# COLLECTION AND CLASSIFICATION OF SERVICES AND THEIR CONTEXT

Arash Khodadadi

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

September 2015

# Abstract

Collection and classification of Services and their context

Arash Khodadadi

SOA provides new means for interoperability of business logic and flexible integration of independent systems by introducing and promoting Web Services. Since its introduction in the previous decade, it has gained a lot of attraction through industry and researchers. However, there are many problems which this novel idea of SOA encounters. One of the initial problems is finding Web Services by the service consumers. Initial design of SOA proposed a service registry between service consumers and service providers but in practice, it was not respected and accepted in the industry and service providers are not registering their services. Many SOA researches assume that such registry exists but, a repository of services is preliminary to the research. The Internet is filled with many Web Services which are being published everyday by different entities and individuals such as companies, public institutions, universities and private developers. Due to the nature of search engines to support all kinds of information, it is difficult for the service consumers to employ them to find their desired services fast and to restrict search results to Web Services. Vertical search engines which focus on Web Services are proposed to be specialized in searching Web Services. Another solution proposed is to use the notion of Brokerage in order to assist the service consumers to find and choose their desired services. A main requirement in both of these solutions is to have a repository of Web Services. In this thesis we exploit methodologies to find services and to create this repository. We survey and harvest three main type of service descriptions: WSDL, WADL, and Web pages describing RESTful services. In this effort, we extract the data from previous known repositories, we query search engines and we use Web Crawlers to find these descriptions.

In order to increase the effectiveness and speed up the task of finding compatible Web Services in the Brokerage when performing service composition or suggesting Web Services to the requests, high-level functionality of the service needs to be determined. Due to the lack of structured support for specifying such functionality, classification of services into a set of abstract categories is necessary. In this thesis we exploit automatic classification of the Web Service descriptions which we harvest. We employ a wide range of Machine Learning and Signal Processing algorithms and techniques in order to find the highest precision achievable in the scope of this thesis for classification of each type of service description. In addition, we complement our approach by showing the importance and effect of contextual information on the classification of the service descriptions and show that it improves the precision. In order to achieve this goal, we gather and store contextual information related to the service descriptions from the sources to the extent of this thesis. Finally, the result of this effort is a repository of classified service descriptions.

# Acknowledgments

I would like to express my gratitude to my supervisor Dr. Joey Paquet who continuously supported me. His guidance and engagement helped me in all the time of research and writing of this thesis. Furthermore, I would like to thank Dr. Serguei Mokhov for all the technical assistance and support of MARF and MARFCAT. This work has been sponsored by NSERC and the Faculty of Engineering and Computer Science of Concordia University, Montréal, Québec, Canada. Last but not least I would like to thank my family for encouraging and strengthening me.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Listings

# Chapter 1

# Introduction

Since SOA was introduced in the previous decade, Web Services have gained an enormous trend in industry and research [10], [11], [12], [13], [14], [15], [16], [17]. SOA paradigm promotes the idea that functionality can be exposed through Web Services. This enables flexible integration of independent systems and applications (e.g., desktop applications as well as Web applications). This way Web Services provide new means for interoperability of business logic. However, it still encounters many problems. Creating and publishing a Web Service is only the first step in the whole Web Service life cycle. The next step for the service consumers is to find the service.

The SOA triangle shown in Figure 1 was suggested to be used to publish Web Services to solve the Web Service location problem. According to the theoretical SOA triangle, service provider would register the service it is offering in a *registry* exposing a *repository* of all services. Then, a service consumer would search the service registry to find its desired service.

A standard describing protocol bindings and message formats which are required to interact with a Web Service registry called *UDDI* (Universal Description, Discovery and Integration) [18] was proposed in 2002 by major companies such as Microsoft, IBM, SAP, Oracle and Intel to implement this model. UDDI defines a universal method for enterprises to dynamically discover and invoke Web Services. Although it was targeted to fill the gap between service consumers and service providers

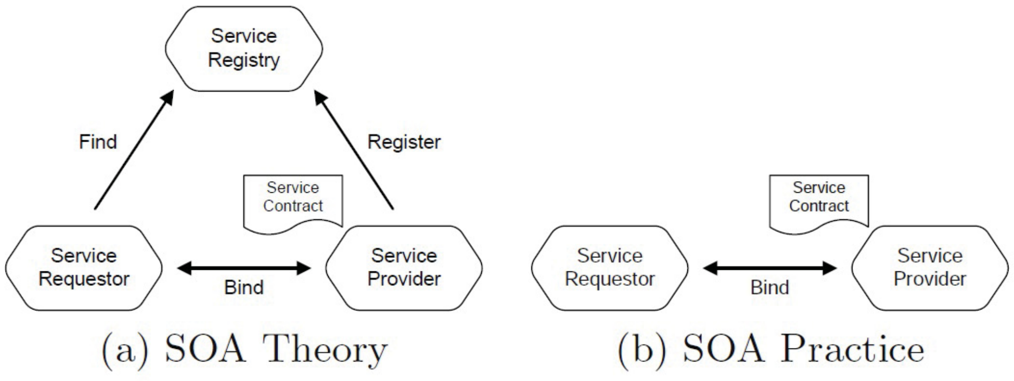


Figure 1: Basic SOA Model - Theory vs. Practice [1]

and provide a universal registry, this standard has not prevailed in the domain of publicly available Web Services and was not accepted by industry mainly because of unsatisfactory quality of data as mentioned in [19]. The authors in [19] surveyed that approximately 67 percent of the registrations were not valid and many of the downloaded WSDL files omitted mandatory elements or contained other syntax errors. As a result, all major UDDI Business registry (UBR) nodes that have once been set up by companies such as IBM, Microsoft and SAP have been discontinued already in 2006 [20]. As a result, as the authors in [1] describe and shown in Figure 1, in practice this approach did not work as intended. The model used in most of today's SOA applications consists of only the service consumer and service provider and service providers do not register their services anymore. As a result, service consumers cannot rely on the data provided by the service registry. Many SOA researches assume that such registry exists but, a repository of services is preliminary to the research.

Nowadays, there are two main ways to publish a Web Service in the Web which will be available for users to find them [20]:

### Publishing it in a portal

There are different private and public portals available which are specialized in listing Web Services, e.g., ProgrammableWeb[1], XMethods[2] or Service-Repository[3]. These are usually Web pages which hold references and descriptions for Web Services and offer search interfaces to retrieve the desired Web Services.

**Providing access to Web Service description**

The other way is to provide access to the service description and describe the the features of the service using Web pages or other documents. This information can then be found by the Web Crawlers (Section 3.2.1) of standard search engines.

According to [21] and [22], the portals only cover small portions of the actually available services and suffer furthermore from the problem of missing or outdated information. Furthermore, this approach needs the service providers to manually register their services.

Using standard search engines with keyword search makes it difficult to find a desired service fast because their nature is to support all kinds of information and it is hard to restrict a search to Web Services.

The authors in [23] and [22] show that general-purpose search engines provide a big coverage of services, but at the same time have drawbacks and are not highly efficient in terms of recall and precision. This is mainly due to the fact that keyword search is done on a pure document level and not on a service level.

To solve this problem, a *vertical* (semantic, focused, or topic driven) *search engine* which is specialized in Web Services is suggested. According to [24] and [25], vertical search engines focus on particular predefined topics, find Web pages related to them and allow the users to search for information related to those topics.

The *Brokerage* notion was introduced between service providers and service consumers to assist the consumers to find the service which suits their needs and for integrating and deploying services. These search engines can be included inside of

[1]`http://www.programmableweb.com/`
[2]`http://www.xmethods.com/`
[3]`http://www.service-repository.com/`

a service *Broker* to be used by the Broker to perform service discovery and eventually abstracting services and sending back the result or to be offered as an interface to the user for finding Web Services. However, Broker can serve multiple purposes besides that. On behalf of consumer, the Broker performs service negotiation. In such circumstances, the Broker can have the power of distributing services across multiple providers for the cost effective solutions and offering the best one to the user. In addition, the Broker can fulfill the considerable need for automatic composition of Web Services to make composed functionality from existing services.

Figure 2 shows the proposed architecture of Brokerage which shows the proposed components inside of a Broker [2]:

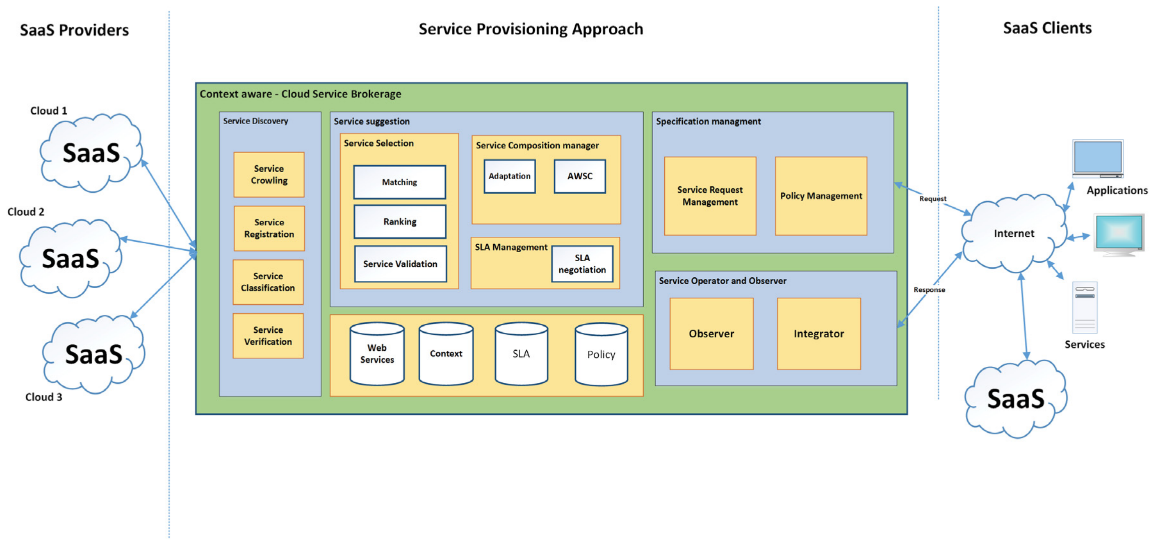


Figure 2: Brokerage Architecture [2]

Whether it is a Broker or a vertical search engine, a repository of Web Services needs to be created automatically. Due to the changing world of Web Services and because Web is changing all the time and Web pages are being added, modified or deleted, to keep the repository updated and fresh, an updatable design is required.

The Broker can perform the task of service composition however, according to [26], interoperability issues arise when composing heterogeneous systems at run-time. The first step in composing such systems is to determine whether, and to what degree, the systems are compatible. As the authors in [6] mentioned, traditional solutions to determine compatibility between systems are rather expensive in terms of computational cost, especially when these are applied to systems in unrelated domains. When two systems (Web Services) want to interact, a compatibility assessment which requires in-depth analysis considering the interface and conversational protocol of them needs to take place. As the authors in [6] argue, one way to speed up the assessment above is to apply Machine Learning methods to automatically classify high-level functionality of a system's interface description, i.e, the highest level of abstraction of what the system does. This will result in restricting the scope of compatibility checks and consequently providing an overall performance gain when looking for matches between systems. There is however no structured support for specifying the abstract class to which the service belongs [6].

Web Service tags are the terms annotated by the users to describe the functionality or quality of Web Services. According to [27], since user tagging is inherently uncontrolled, ambiguous, and overly personalized, and as [28] reveals, many tags provided by the users are imprecise and there are only around 50 percent of the tags actually related to the target object. As a result, these tags can be only treated as collective user knowledge and as a *contextual* information (will be defined in Section 1.1) which needs purification for the Web Services.

In addition to increasing performance of compatibility assessment, the authors in [29] argue that classifying Web Services into different sets based on the tags (clustering) facilitates the task of Service Discovery. Moreover, the result of Service Classification can be very useful to the end-users when selecting services.

There two main approaches towards Service Classification: *manual classification* and *automatic classification*. According to [30], the former methods are very expensive, both in time, effort and consequently financially. The latter methods however are quite inaccurate and do not in general provide quality annotations but cheaper than the former. Although the authors in [30] try to decrease the cost of manual classification by applying crowd sourcing techniques, it is still more expensive than the automatic methods.

## 1.1 Thesis Overview

Due to the fact that cost plays an important role and because of the resources available to us, although we are aware that the annotations in many cases are inaccurate and the automatic classification may not be as accurate as manual methods, we use automatic classification approach as our primary methodology. Accordingly, we try to find the highest accuracy achievable in the scope of this thesis by employing a wide range of *Machine Learning* and *Signal Processing* algorithms and techniques and putting the *context* into practice. Context, in the Web Services environment is any information about the service consumer, service provider, and communication protocols. Hence, content of the service descriptions and any information related to them is considered as a context for the service. However, due to the problems with the current repositories available to us (Section 1.3), we first exploit methods to harvest three main kinds of Web Service descriptions (*WSDL*, *WADL*, and Web pages describing *RESTful* services) from the World Wide Web with respect to updatability of the design. In this effort, we extract the data from previous known repositories, we query search engines and we use Web Crawlers to find the documents describing Web Services (descriptions). In addition, during this process, we gather and store contextual information related to these descriptions to the extend of this thesis to use as a complement to the descriptions for the classification. We show the importance and effect of this context on the classification and show that it improves the accuracy of the results.

The architecture of this thesis is illustrated in Figure 3. Each block will be discussed in details in the next chapters which are outlined in Section 1.6.

## 1.2 Scope

The scope of this thesis is finding and classification of main three types of Web Service descriptions (WSDL, WADL, and Web pages describing *RESTful* services) and their contextual information. We use Web Crawlers, previous known repositories, and

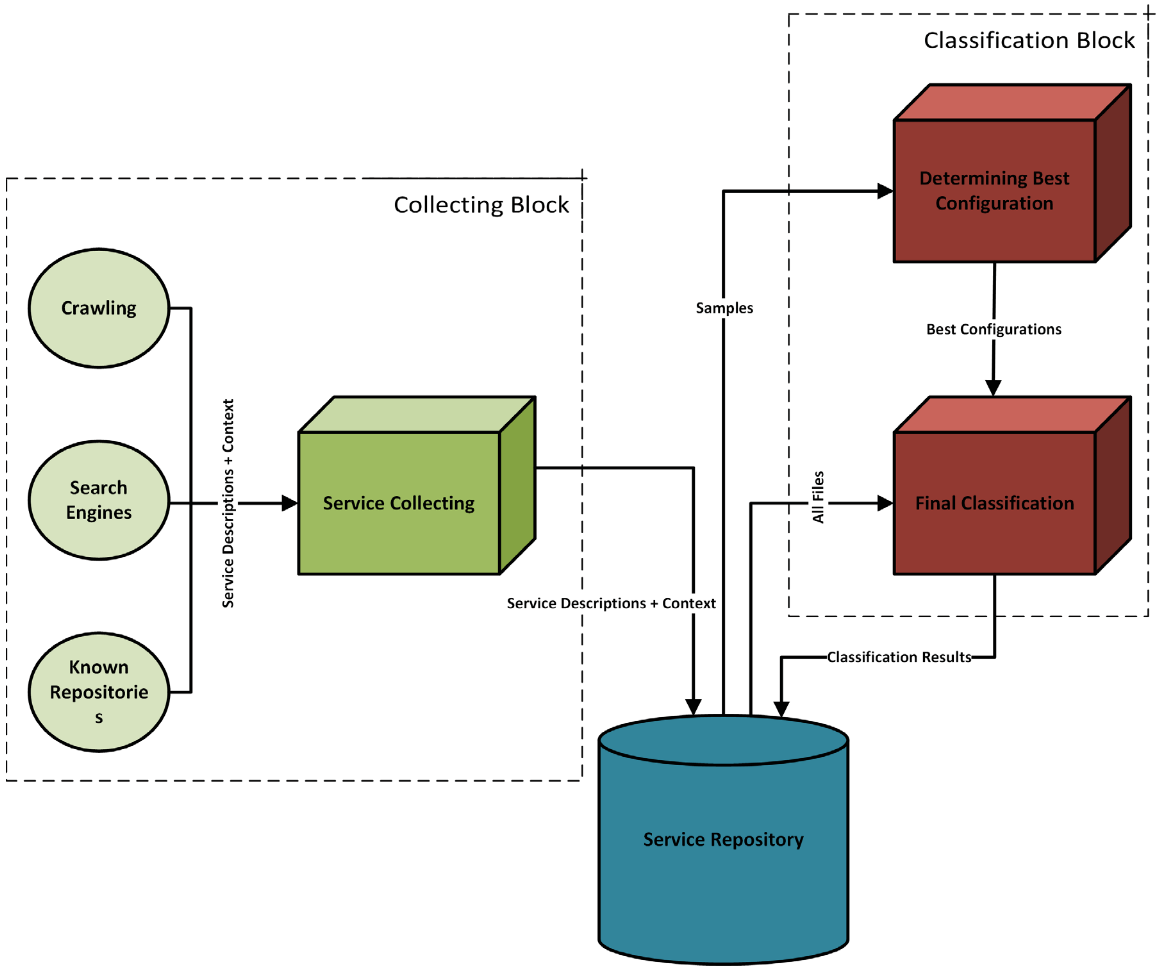


Figure 3: Thesis Architecture

search engines to collect services and their context. However, Web pages describing services, are much more complex to detect and validate. As a result, for simplicity, finding them using Web Crawlers is out of the crawling scope and also, validation of these description pages is out of the validation scope of this thesis. Also, extraction and validation of service endpoints are out of the scope of this thesis.

The scope of the Service Classification in this thesis is to classify the service descriptions and to survey the effect of context using Signal processing and Machine Learning techniques. However, validation and determining the quality of context, context ontology based matching, and also adaptation of the classification process based on the new upcoming services is out of scope of this thesis.

## 1.3 Problem Statement

As discussed previously, from different perspectives a repository of Web Services which is updated regularly is needed particularly in the proposed Broker. However, according to our results in Chapter 6, the current repositories available are not satisfactory to work with. These repositories contain many Web Services which are already outdated and not available anymore, focus on a particular type of Web Service, are not comprehensive and sufficient from both services and context point of view, and/or do not respect the inconsistent nature of Web Services and contain only the actual descriptions and not the source.

On the other hand, we discussed the importance of Service Classification and how it is used in many applications. However, there is no structured support for specifying the abstract class to which the service belongs [6]. An extensive amount of research has been carried out on the topic of automatic classification of a text document which is usually performed by employing Machine Learning methods. However, there is still much research that needs to be done in this area to improve the process specially on combining Signal Processing algorithms and techniques with Machine Learning in the scope of Service Classification. Also, the context of the service descriptions was not considered as a separate entity to prove the importance of it.

## 1.4 Goals and Motivations

Our goal in the service collecting part is to construct a reliable and fresh repository of main kinds of Web Service descriptions and their contextual information with respect to the changing nature of the servers and the services. In addition, our goal in the Service Classification part is to find the best combination(s) (accuracy, recall, and time) of algorithms and options for classification of each type of Web Service description. Moreover, our goal is to show the effect of context on Service Classification and to demonstrate that it improves the accuracy.

Eventually, our goal is to construct a classified repository of Web Service descriptions to serve the need in a Broker or a vertical search engine. This will fill the first step for creating the Broker illustrated in Figure 2 to serve multiple purposes.

## 1.5 Contributions

Firstly, we designed and implemented Web Service collecting tools to find main three type of Web Service descriptions (WSDL, WADL, and Web pages describing *RESTful* services) and their contextual information and stored them with respect to changing nature of Web Services by keeping both sources and actual files based on time:

- Web crawling (except for REST description Web pages)
- Extracting data from known repositories and verification of them
- Using search engines
- Creating the repository with respect to updatability

In this effort we found 72,454 unique service description URLs including 39,288 WSDL URLs, 1,830 WADL URLs, and 31,336 HTML page URLs describing RESTful services. From these URLs we stored 48,161 actual service description files including 16,096 WSDL descriptions, 450 WADL descriptions, and 31,615 HTML files describing RESTful services.

Secondly, we survey a wide range of Machine Learning and Signal Processing algorithms and techniques by designing and implementing a classification method to find the best combination(s) of algorithms and options from the viewpoint of accuracy, recall, and time achievable in the scope of this thesis:

- Analyzing main three types of service descriptions
- Extracting and analyzing only text from Web pages describing services
- Survey the effect of context on each set (training and testing)
- Studying different clustering and frequency options in the classification process

- Showing that context improves the accuracy
- Classification of 48,161 service descriptions

In this effort we surveyed 72 different cases based on the sample types, clustering options, and contextual information (Section 4.4). We tested 864 algorithm permutations in the Signal Processing pipeline for each of these cases.

## 1.6 Outline

In this thesis to begin in Chapter 2 we mention the related and background work. Thereafter, in Chapter 3 we discuss the service collecting approach and procedure and in Chapter 4 we argue the Service Classification process and methodology. Thereafter, in Chapter 5 we discuss the design and implementation of both parts. Finally, in Chapter 6 we illustrate the results and evaluate our work and in Section 7 we conclude our effort and mention the future work.

# Chapter 2

# Background and Related Work

As discussed in the previous chapter, the first step and requirement for creating a Broker or a Vertical search engine specialized in finding Web Services is to have a comprehensive and updated repository of Web Services which brings the notion of collecting services. Furthermore, to facilitate the compatibility assessment between Web Services during composition, Service Discovery, Service Selection, and generally to present more satisfactory results to the end-user, the aforementioned repository needs to be classified. This classification can be leveraged from Machine Learning techniques. In addition, in order to increase the precision of classification, contextual information can be added to the repository. In this chapter we discuss the related work to these areas of research.

## 2.1 SOA, Cloud and Broker

**Service-Oriented Architecture (SOA)** has been a popular research topic from the time it was introduced in the previous decade. There are many papers published in this area specially [10], [12], [14], [11], [13], [31], [32], which focus on the concept. Many definitions were given for SOA.

The *Open Group* (a vendor and technology-neutral industry consortium, currently with over four hundred member organizations [33]) defines *Service-Oriented Architecture*, *Service-Orientation*, and *service* as [34]:

*Service-Oriented Architecture (SOA) is an architectural style that supports service-oriented development (i.e., developing software by reusing existing services as development building blocks).*

*Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services.*

*A service is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports), is self-contained, may be composed of other services, and is a black box to consumers of the service.*

The *OASIS* group (Organization for the Advancement of Structured Information Standards) defines *service-oriented architecture* as [35]:

*A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.*

**Cloud Computing** was defined by *NIST* (The National Institute of Standards and Technology of USA) [36] as [37]: *A model for enabling convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

The authors in [38] approach Cloud Computing as an extension of Service-Oriented Architecture and compared the two paradigms. Accordingly, SOA proposes business solutions to create, manage and reuse the computing components using services where Cloud Computing offers a set of technologies and infrastructure that serves a bigger, more flexible platform for enterprise applications to build their SOA-based solutions. To sum up, SOA and Cloud Computing will co-exist to complement and support each other.

According to [37] and [38], a typical Cloud environment incorporates services in three levels:

**Infrastructure as a Service (IaaS)**

Provides services of virtual computers, operating systems, data centers, and network connectivity.

**Platform as a Service (PaaS)**

Offers computing platforms for designing and developing applications and application-hosting environment without the concern of hardware and storage.

**Software as a Service (SaaS)**

Provides services for software deployment and maintenance. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface.

However, in this thesis we do not distinguish between the services which we find and store and classify all three level services.

**A Cloud Broker** is introduced by the authors in [39] and [40] between Cloud providers in order to assist the service consumers to choose the platform which suits their needs and for integrating and deploying services from multiple Clouds. According to [39], it is time-consuming for consumers to collect the necessary information and analyze all service providers to find their desired service. In addition, from a computational perspective, Broker can be useful because the same computations may be conducted repeatedly by multiple consumers who have similar requirements. A Broker has multiple functionality such as Provider Information Publishing, Ranking, Selection, Composition, and On-Demand Provision Model.

According to the *MarketsandMarkets* (world's No. 2 firm in terms of annually published premium market research reports [41]), the global Cloud service Brokerage market is expected to grow from \$5.24 Billion in 2015 to \$19.16 Billion by 2020, at a Compound Annual Growth Rate (CAGR) of 29.6% [42].

The authors in [43] and [3] propose a framework for SaaS provisioning, which relies on brokered SLAs (Service Level Agreements), between consumers and providers. SLA is the agreement between service consumer and service provider which describes agreed service functionality, cost, and qualities [44]. Figure 4 illustrates the proposed Broker architecture. Service consumers request the Broker for providers that can fulfill their functional and non-functional requirements and the Broker ranks potential providers by matching their offerings against the requirements of the service consumer. Furthermore, the Broker will perform SLA negotiation with the selected SaaS providers, on behalf of service consumers, and perform SLA compliance monitoring.

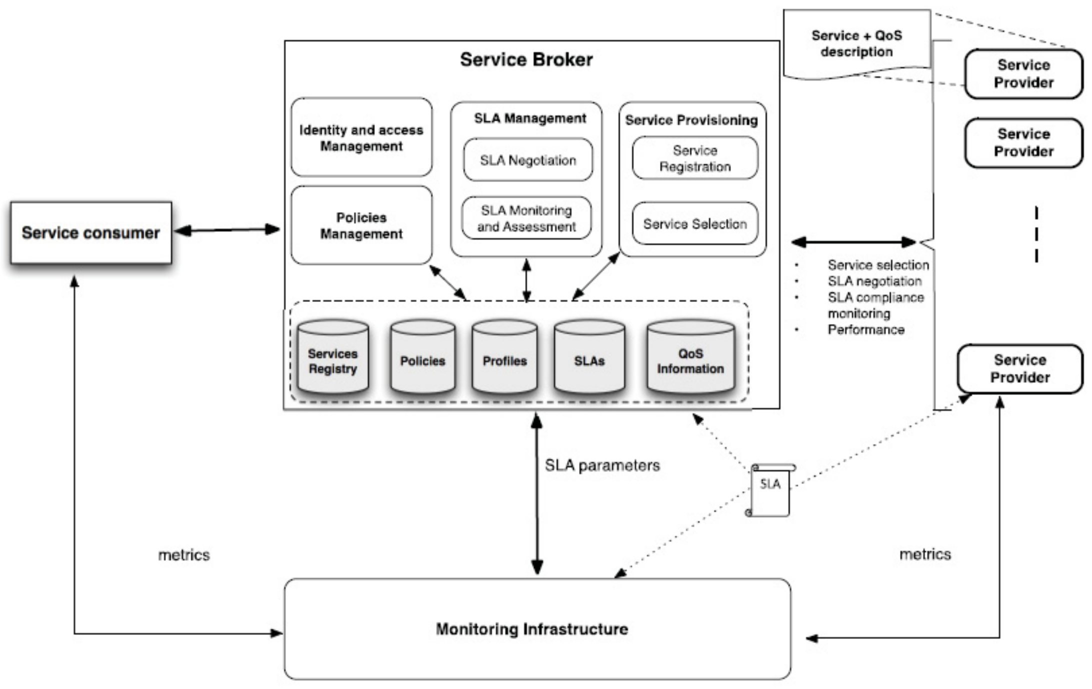


Figure 4: Cloud service Broker [3]

## 2.2 Service Discovery

Service Discovery is the process of detecting a suitable service to fulfill the functional and non-functional requirements which essentially requires a set of service descriptions. Common approaches in Service Discovery mostly work on top of restricted sets of services including either the actual service descriptions as published in the Web or semantic service descriptions that describe the service in terms of its non-functional properties with the aid of Web Ontology Language (OWL-S) [45] and Web Service Modeling Ontology (WSMO) [46].

Semantic Service Discovery focuses on the discovery of semantic service descriptions and it has been an ongoing research topic [47], [48], [49], [50], [51], [52], [53], [54], [55], [56] etc. According to [20], Semantic Service Discovery achieves a high precision, if precise and rather complex descriptions are available for the services. However, such descriptions are presently not available at large scale, but only for restricted sets of services. Semantic Service Discovery needs often complex reasoning methods in the background that are not yet efficient enough to work on top of a real large number of services on Web scale [20].

On the other hand, there are a number of approaches proposed for Non-Semantic Service Discovery. The authors in [57] proposed a clustering algorithm that groups names of parameters of Web Service operations into semantically meaningful concepts which determine similarity of inputs (or outputs) of Web Service operations. Based on their approach they developed a search engine called Woogle which supports similarity search for a set of 1500 Web Services in addition to simple keyword-based retrieval. The authors in [58] clustered services based on search sessions instead of individual queries to take advantage of similar queries by different users. The authors in [59] tried to reflect the underlying semantics of Web Services by fully utilizing the terms within WSDL files. In their approach, the similarity between two services was measured upon the the semantic distance of terms from two compared services. service Clustering and Service Discovery are tightly coupled in the literature which will be mentioned in Section 2.4.

## 2.3 Service Crawling

Preliminary to Service Discovery methods, a populated repository of services is necessary. In order to find the actual service descriptions which are published in the World Wide Web, different approaches are used. The authors in [60] and [61] introduce the Web Service Crawler Engine (WSCE) which is a Web Crawler designed to crawl Web Service repositories and search engines to collect Web Service information. The proposed solution was targeting problems that the centralized service repositories were suffering from, such as single point of failure, bottlenecks and outdated data. In their approach, they automatically collected 5,077 WSDL files from other repositories, search engines and portals to form a new portal. However, Web Services that were not entered to repositories and not available in search engines, remained out of reach for WSCE.

Another Web Service crawler was proposed by [62] based on Pica-Pica Web Service description crawler to search for WSDL files and descriptive data in existing repositories which was able to find 4,148 WSDL files from different repositories. However, the extracting of service related data was done by implementing algorithms explicit to each registry. This approach created a dependency to the existing Web Service repositories and the same problem as in WSCE which would miss services that are not present in repositories.

Some researchers focused on Heritrix[1] which is an open-source archival Web Crawler designed for periodic snapshots of a large portion of the Web. The authors in [20] and its related publications [63], [64] concentrated on methods for automated Web Service crawling based on this crawler and then creation of semantic annotations for the found services. They describe the methodologies for enabling Service Discovery on Web scale and by crawling the Web for WSDL files and related documents, and building unique service objects from multiple Web resources and subsequently enrich the resulting services with simple annotations from basic service information. At the end, they employ semantic-based techniques and aggregating the various real-world information (information that is available about services by analyzing their description, their hyperlink relations, and monitoring information) to rank the services. This work emerged to Seekda[2] Web Service search engine (currently inoperative) with around 28,000 service endpoints extracted from WSDL files from 8,000 service providers. The authors in [65] also followed a similar approach and

[1]http://crawler.archive.org/

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

used Heritrix to crawl and collect 463 WSDL files and then deduced the semantic information from the crawled Web pages and created annotations for each service. Subsequently, these annotations were used to derive a classification for each Web Service into different application domains. The authors in [66] took advantage of the heuristics which were proposed in these papers and improved the performance of the focused crawling (loading only relevant Web pages and discard those out of concern).

The authors in [67] and [68] crawled 21,358 WSDL URL addresses and obtained 16,514 WSDL files from different Web Service portals and search engines such as Seekda for the purpose of Quality of service evaluation and prediction.

The authors in [69], [70], [29], and [27] also introduced a new Web Service search engine named Titan[3] with 15,968 WSDL files which was crawled from Seekda. However, the focus of these papers is more on the tags associated to the services which will be mentioned in the next section.

To conclude, these papers focused mostly on WSDL files and one specific method to find the services and also kept only the actual description files without the source of the data.

## 2.4 Service Clustering and Classification

Service Clustering is an effective solution to boost the performance of Service Discovery and many researchers focused on this task. There are two main approaches towards service clustering which are widely employed.

Semantic service Clustering focuses on clustering of semantic Web Service descriptions. The authors in [71], [72], [73], [74], and [75] particularly focused on semantic service clustering by employing ontology-based approaches. However, as mentioned earlier in Section 2.2, semantic Web Service descriptions are not widely used and supported by the industry circle.

Many researches leveraged from the structure of WSDL files to measure the similarity between Web Services [57], [76], [77], [78], [58], [59]. [76] and [77] proposed

[3]http://ccnt.zju.edu.cn:8080/

to extract and focus on 4 features including content, context, host name, and service name from WSDL files and suggest to put the service Clustering as the pre-processor to Service Discovery. In contrast, the authors in [78], [69], and [29] proposed to extract 5 features including content, types, messages, ports, and service name from WSDL documents to perform service Clustering and Discovery.

In order to facilitate clustering, the authors in [79] allowed users to express their perception on service functionality (after testing or using them). The authors in [30] focused on easing and thus reducing the cost of manual annotations by utilizing crowd sourcing. The authors in [80] proposed to employ Machine Learning techniques and WordNet[4] to automatically annotate tags to services. In order to automate tagging services and make services easily accessible and attractive to the users, the authors in [81] modeled tag prediction problem as a multi-label classification problem. The authors in [69] utilized on both WSDL documents and tags associated to them to recommend tags to the Web Services with few tags according to the tag co-occurrence. Later in [29] the authors proposed a hybrid tag recommendation strategy which employs tag co-occurrence, tag mining, and semantic relevance measurement in order to handle the clustering performance limitation caused by uneven tag distribution and noisy tags. In addition, in [27] they proposed a hybrid mechanism by using WSDL documents and tag network information to compute the relevance scores of tags by employing semantic computation and Hyperlink-Induced Topic Search model, respectively. From another perspective, The authors in [6] argued that the automatic classification of Web Service descriptions into a predefined can considerably speed up the task of finding compatible Web Services due to restriction of computationally-expensive compatibility assessments to systems within the same domain category. In their effort, they employed Machine Learning techniques and developed a method to extract the features from WSDL files to exploit the characteristics of them and infer the categorization function.

Overall, the authors in these papers mostly focused on WSDL files only as one type of service to perform the classification. In addition, there is not much research on

[4]`http://wordnet.princeton.edu/`

employing Machine Learning techniques and also combining the actual descriptions and their contextual information in the classification process.

# Chapter 3

# Service Collecting

The World Wide Web, or simply Web, can be divided into two varieties: ***Surface Web*** also known as the Visible Web or Indexable Web [82], and ***Deep Web*** also known as Invisible Web [83], Deepnet [84] or Hidden Web [85]. Surface Web is data on the Web that is available to the general public and has been crawled and indexed by general-purpose search engines whereas Deep Web can be defined as that part of Web that is not part of the Surface Web.

Search engines construct a repository of the Internet by using programs called *Spiders* or *Web Crawlers* that begin with a list of known Web pages called *seeds* which will be discussed further in Section 3.2.1. For various reasons some pages can not be reached by the Web Crawlers which consists mainly of pages that do not exist until they are created dynamically, password-protected pages or the data that is accessible only via Web Service method calls. These Web pages are referred to as the Deep Web.

In 2001, the authors in [86] estimated that information stored in Deep Web is 400 to 550 times larger than the commonly defined World Wide Web. In 2005, the authors in [87] queried search engines with search terms from 75 different languages and stated that there were over 11.5 billion Web pages in the publicly indexable World Wide Web. As of January 2015, [88] determines that the Surface Web contains at least 46 billion pages.

When the *SOA* (Service-Oriented Architecture) paradigm was introduced in the previous decade, the amount of Web Services available on the Web increased. It started promoting the idea that independent systems and applications should communicate with each other by exposing and using services [89]. Many Web Services, or simply services, are being published in the Internet by different entities and individuals such as companies, public institutions, universities and private developers, using either the *WSDL* (Web Service Description Language) [7] standard or following a *RESTful* (Representational State Transfer) [90] approach.

However, creating and publishing a Web Service is only the first step in the whole Web Service life cycle. The whole Web Service life cycle includes all relevant topics such as Creation, Finding, Discovery, Selection, Ranking and Composition. The next step is to find the location of Web Services.

As discussed in Chapter 1, the initial UDDI model for publishing Web Services was not accepted by industry and service consumers cannot rely on the data provided by the UDDI service registries. Precise and rather complex descriptions are not available at large scale to employ Semantic Service Discovery approaches. Web Service portals need the service providers to manually register and update their services and they suffer from the problem of missing or outdated information. Furthermore, using standard search engines with keyword search makes it difficult to find a desired service fast because their nature is to support all kinds of information and working on a document level rather than a service level.

As mentioned in Chapter 1, whether it is a *Broker* or a *vertical search engine* (focused on a particular predefined topic) searching for desired Web Services, a repository of Web Services which would update its services automatically is needed. Our goal is to combine these methods to provide a comprehensive repository of both kinds of Web Services (WSDL and RESTful) found from the Web with an efficient design with respect to updatability of the repository.

In our effort, as illustrated in Figure 5, we extract the data from previous known repositories (Section 3.2.3), we query Google[1] to find the services from the Surface Web (Section 3.2.2) and we use Web Crawlers to find Web Services which are missed

[1]http://www.google.com

(Section 3.2.1). Instead of using Semantic Service Discovery because of the lack of rich descriptions, our approach rather focuses on the descriptions which are currently used for the numerous already available services and tries to gather related *contextual* information (Section 3.1.4). This contextual information will also play a big role in Service Classification later which will be discussed in Chapter 4.

This chapter will form the collecting block in Figure 3.

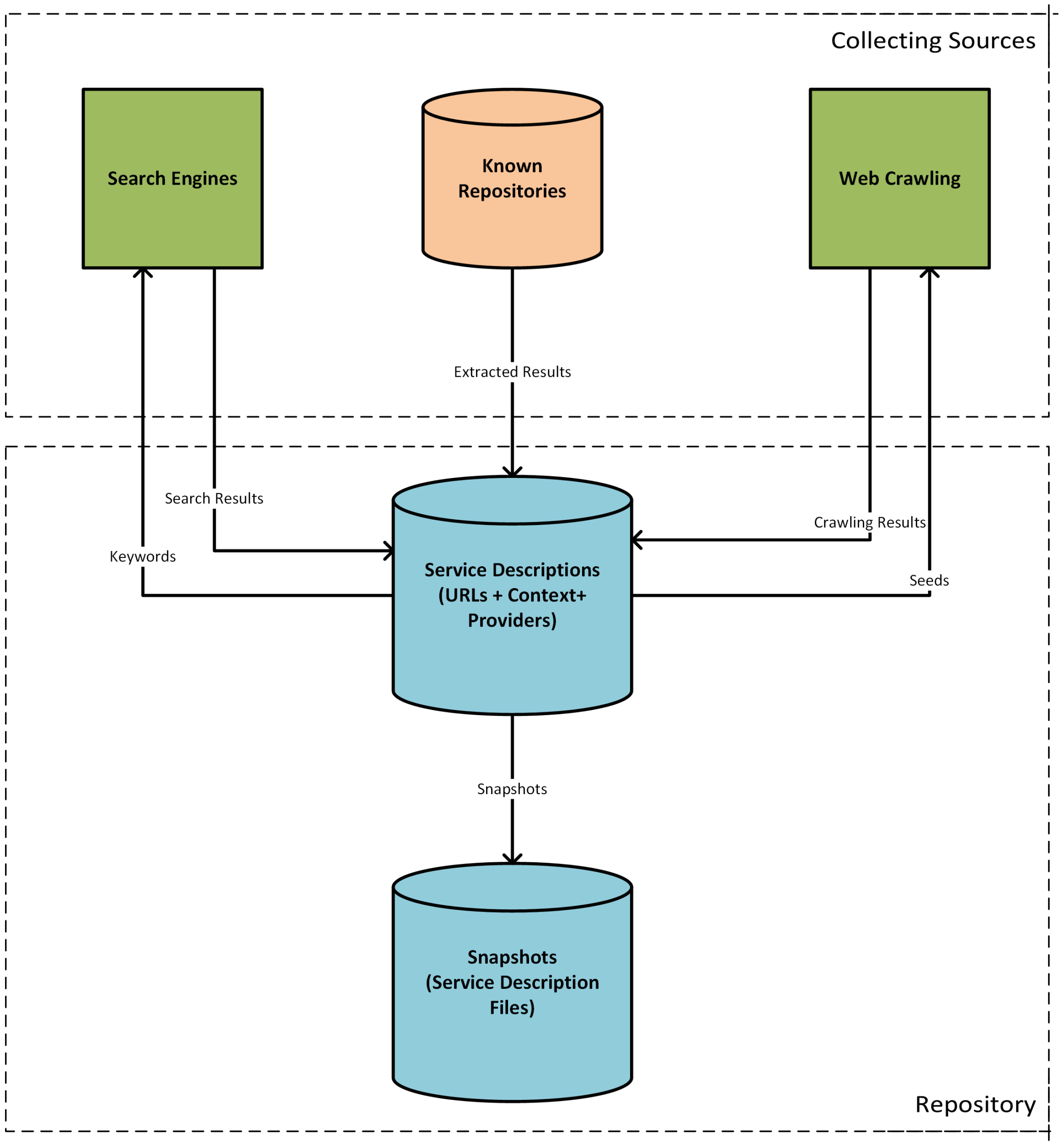


Figure 5: Abstract Service Collecting

## 3.1 Collecting Data

In this section we discuss different kinds of service descriptions and contextual information which we collect using collecting sources which will be discussed in Section 3.2.

### 3.1.1 WSDL Collecting

The SOA paradigm promoted the idea that independent systems and applications should communicate with each other by exposing and using services, the amount of Web Services in the Internet increased [89] and it became increasingly important to standardized and describe communications protocols and message formats in some structured way. **Web Service Description Language**, or simply **WSDL**, addresses this need by defining an *XML* grammar for describing services as collections of communication endpoints capable of exchanging messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services) [7]. In other words, WSDL is a contract between service provider and service consumer for describing Web Services provided. WSDL enables the service provider to separate the description of the abstract functionality offered by a service from concrete details of a service description such as *how* and *where* that functionality is offered [8]. It is platform and language independent and also extensible, to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. However, it is highly used in conjunction with *SOAP* (*Simple Object Access Protocol*). There are currently two main versions of WSDL used world wide: WSDL 1.1 [7] and WSDL 2.0 [8]. To see the difference and examples of both files refer to Appendix A.1.

WSDL files are generated using tools by different service providers and published through the Web. Our goal is to find and store them in a repository with respect to updatability. We also store the URLs which point to the WSDL file instead of retrieving and storing WSDL directly. Then from these URLs we retrieve the actual WSDL file and store its *snapshot* in another repository. We will discuss about this further in Section 3.3.2.

For achieving this goal, we extract the data from previous known repositories (Section 3.2.3), we query Google (Section 3.2.2) and we use Web Crawlers to find WSDL files which are missed (Section 3.2.1).

### 3.1.2 REST Collecting

*Representational State Transfer*, or simply *REST*, was first introduced in 2000 by Roy Thomas Fielding at the University of California, Irvine, in his academic dissertation [90] which analyzes a set of software architecture principles that use the Web as a platform for distributed computing. Now, years after its introduction, REST has gained widespread acceptance across the Web as a simpler alternative to SOAP and WSDL-based Web Services. Key evidence of this shift in interface design is the adoption of REST by mainstream Web 2.0 service providers such as Yahoo, Google, and Facebook who have deprecated SOAP and WSDL-based interfaces in favor of RESTful services [91].

While REST is a software architecture style consisting of guidelines and best practices for networked hypermedia applications, it is primarily used to build Web Services that are lightweight, maintainable, and scalable [92]. It relies on a stateless, client-server, cache-able communications protocol and in virtually all cases, the HTTP (Hyper Text Transfer Protocol) is used. To see an example of a REST resource refer to Appendix A.3.

Whereas WSDL descriptions are well-structured by using XML meta-language standards and allow an easy recognition, RESTful services do not follow any strict structured description rules. Therefore, they are described in different formats and most of the time, in different Web pages by different service providers. As a result, RESTful services are much harder to detect and validate hence, using Web Crawlers to discover REST services (except WADL-based ones) are out of the scope of this thesis and will not be discussed. However, our goal is to find and store different kinds of Web Services in the repository, therefore we extract the data from previous known repositories (Section 3.2.3) and we query Google (Section 3.2.2) to find RESTful services. Similar to WSDL approach, we store each step's results separately. We also store the URLs which point to the description of RESTful service and from these URLs we retrieve the description and store its *snapshot* in another repository. We will discuss about this further in Section 3.3.2.

### 3.1.3 WADL Collecting

*Web Application Description Language*, or simply *WADL*, was introduced by SUN Microsystems Inc. on 31 August 2009 as a structured description for REST Web Services. WADL allows the description of HTTP-based Web applications (typically REST Web Services), putting the emphasis on the basic description of those services from the HTTP interaction standpoint, while allowing different grammars and formalizations to describe the payloads and parameters used during the interaction [93] [9]. Similar to WSDL, WADL is platform and language independent and aims to be machine readable and well-structured using *XML* standards. To see an example of a WADL file refer to Appendix A.2.

In contrast, in some respects, WADL is not as flexible as WSDL (e.g., WADL cannot express binding to SMTP servers), but it is sufficient for any REST service and much less verbose and following the REST approach. WADL is lightweight, easier to understand and easier to write than WSDL. However, it is still not very popular through service providers and there are no current plans to standardize it [93]. As a result, REST services are not widely described using WADL in the Web and no specific portals are dedicated to it. We query Google (Section 3.2.2) and use Web Crawlers to find this kind of RESTful descriptions. Similar to the other types, we also store the URLs which point to WADL files and from these URLs we retrieve the service description and store its *snapshot* in another repository. We will discuss about this further in Section 3.3.2.

### 3.1.4 Context Collecting

It is a challenging task to define the word *context* as many researchers tried to find their own definition for what context actually includes. In literature, the term *context-aware* appeared in [94] for the first time where the authors described context as location, identities of nearby people, objects and changes to those objects. Many researchers and developers consider only physical context in contrast with logical context. Physical or external context refers to context that can be measured by hardware sensors, e.g., location, light, sound, movement, touch, temperature or air pressure. On the other hand, the logical or internal context is mostly specified by the user or captured by monitoring user interactions, e.g., the user's goals, tasks, work context, business processes, or the user's emotional state [95] [96]. According to [97], in general, context can be defined as any vital information that can be used to characterize the situational state of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves.

Accordingly, we define context as shared information that can enhance the comprehension and awareness of a system and help the system to enhance the operation. More specifically, in the Web Services environment it could be any information about the service consumer, service provider, and communication protocols. Hence, any information and description related to a specific Web Service is considered as contextual information which will be also used in Service Classification in Chapter 4. We find the following contextual information for the Web Services in addition to their description files:

- The domain of the URL which the service description is found from (service provider domain)
- Annotations, categories or tags assigned to the service
- Text describing the service
- Name of the service

## 3.2 Collecting Sources

In this section we discuss different kinds of sources and approaches which we use to collect different kinds of service descriptions and contextual information which was discussed in Section 3.1.

### 3.2.1 Web Crawling

A *Web Crawler*, also called Spider [98] or Automatic Indexer [99] is a software program that systematically browses the World Wide Web and visits Web pages. Search engines make the most widespread use of Web Crawlers when they collect Web pages on the Web to build their indexes [20]. The most well-known crawler is called Googlebot which is used by Google [100]. A Web Crawler starts with a set of URLs called seeds, fetches the documents (e.g., HTML pages, service descriptions, images and audio files) available at those pages, identifies and extracts all URLs from the documents fetched in the previous step and adds them to the list of URLs to visit which is called the Crawl Frontier. Thereafter, it starts over the same process recursively to move from one page to another [101] [20] [100]. Different crawling strategies are used to perform this process. Crawler types are thus related to the different intentions they pursue when crawling the Web. The main crawl types are [20]:

**Broad or Universal Crawling**

Large crawling where the Web Crawler fetches a large number of Web pages and goes as well into a high depth on each crawled site which results in a high bandwidth usage.

**Focused or Topical Crawling**

Scope of crawling is limited by a number of criteria such as domain, number or topic of the pages which are loaded, i.e., a focused crawler is oriented to cover information corresponding to a specific domain.

**Continuous Crawling**

Web Crawler continuously visits all URLs in its frontier, i.e., the frontier cannot grow fast and the crawl should be scoped.

A Web Crawler usually implements a set of policies to apply these strategies and to define its behavior [102]:

**Selection Policy** States which pages to retrieve.

**Re-Visit Policy** States when to check for changes in a previously visited page.

**Politeness Policy** States the delay between each request to avoid overloading servers.

**Parallelization Policy** States how to distribute the crawling among Web Crawlers.

Our goal is to develop a crawling strategy for Web Services. We do not want to crawl through the whole World Wide Web. Instead, we need to focus our crawl for the Web Service domain thus, we apply *Focused Crawling* technique and during a crawl, we apply methods to identify the service descriptions. However, as mentioned before in Section 3.1.2, RESTful services which are not defined using WADL, are much harder to detect and validate when using Web Crawlers and it is out of the scope of this thesis. Therefore, we only focus our crawling task on WSDL and WADL files because, they are well-structured and can be identified and validated. For applying such strategy, we define *Selection Policy* in our focused crawl to reject a lot of content by default especially binary files such as images, audio or video files and to just fetch HTML pages, XML files and other text documents, i.e., all types of files that could be either a service description or lead to a service description. When a result is found by the Web Crawler, we validate it through WSDL and WADL definitions. If it is a valid description, we store its URL in the service repository and create a copy of the file as its snapshot in the snapshot repository. Moreover, we define *Politeness Policy* using *Politeness Delay* to make delays between the requests to avoid overloading the servers while performing the crawling and *Number of Crawlers* to define the number of crawler threads which will be created simultaneously as *Parallelization Policy.*

This criteria and algorithm will be discussed later in Section 5.3.1 and elaborated in Algorithm 1.

#### Crawling Seeds

As defined above, Web Crawlers start the search from a pre-defined set of URLs which are called *seeds*. These seeds play a very important role in every crawling task particularly when it has a focused scope. To be able to focus our crawls on Web Services, we need to start with a good set of seeds. These seeds ideally need to direct the Web Crawler to the Web pages where the Web Services are published in fewer iterations. Therefore, in our case, we start from the Web pages where we know that Web Services are published by analyzing the results of our other approaches, i.e., we analyze the results from Google Search (Section 3.2.2) and known repositories (Section 3.2.3). Initially, we analyze the results to find the service providers and then we analyze them to find the most common ones to start the crawling from. In other words, the seeds will become a set of provider domain URLs from which a high number of Web Services has been found.

### 3.2.2 Search Engines

*Search engines* are web-based software systems which crawl the Web and index them in a repository to provide search results more efficiently for users. Whenever a user submits a query, it is compared with indexed pages and related pages are loaded as a result. Search engines index many Web pages involving a comparable number of distinct terms and they answer many queries every day. The success of search engines is highly dependent on innovative and effective solutions for Web crawling. Search engines make the most widespread use of Web Crawlers when they collect Web pages on the Web to build their indexes [20].

eBizMBA[2] reports the top 15 most popular search engines in [103] which are the results from a continually updated average of each website's Alexa Global Traffic

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

Rank[3], and U.S. Traffic Rank from both Compete[4] and Quantcast[5]. Google [6] holds the first place in search with an enormous difference of more than 40 percent from second in place Bing[7]. According to the latest comScore[8] report [104], Google led the explicit core search market in with more than 65 percent of search queries conducted.

We use Google as our search engine to query and to find the Web Services from. Google provides two important factors to restrict the search in each request: **query** and **file type**. Query determines the keywords for the search and for the index matching and file type determines the type of the results. For providing the keywords in each query to find the Web Services, we use the previously assigned annotations and tags which we find from known repositories in Section 3.2.3, the categories that we use in Service Classification in Chapter 4, and also we provide many keywords manually which are related to Web Services from semantic, business or technology point of view. Each of these keywords will shape a different request with query specified to them and also, they will become contextual information for the resulting services as discussed in Section 3.1.4.

In file type to find the SOAP-based services, we use WSDL to find the results and for the few REST services which use WADL, we use WADL as the file type in the request. However, as discussed in Section 3.1.2, the most of RESTful services do not follow any well-structured format and thus, there is no file type which we can use to restrict the search and we only use the query. We follow the Politeness Policy between each request to Google. Each of these requests, will return a set of results including the URL of the result and the information which Google provides. This information includes a title and a description about the result which will become extra context in addition to the query for the resulting services. We store all of these URLs and their context in our repository without any validation in this step to follow our goal and to respect the updatability.

[3]http://www.alexa.com/
[4]https://siteanalytics.compete.com/
[5]https://www.quantcast.com/
[6]http://www.google.com
[7]http://www.bing.com/
[8]http://www.comscore.com/

### 3.2.3 Known Repositories

Different portals and registries are available on the Internet which are specialized on listing Web Services. service providers register and introduce their Web Services through these portals to be used by service consumers. These portals are usually Web pages which hold references and descriptions for Web Services and offer search interfaces to retrieve the desired Web Services. The main difference between these portals and general-purpose search engines is that the search engines center their information around the service description document such as WSDL similar to any HTML document while service registries use the services and their providers as central notions, not the underlying documents [20].

In addition to these portals, some researchers made their crawling and classification results available in the Web. These portals and previous results hold valuable services and information about them, therefore we extract the data from them to build our initial repository. These services will help us to find the starting points of our crawling (seeds) as discussed in Section 3.2.1. In addition, the information describing them and the tags, annotations and categories assigned to them are contextual information which help us to do Service Classification in Chapter 4 as discussed in Section 3.1.4.

We extract the data from three different service portals. These portals contain services which are registered manually by the service providers:

- ProgrammableWeb[9]
- XMethods[10]
- Service-Repository[11]

We also extract the data from the repositories available at `http://www.zjujason.com/data.html` which is used in [69], [70], [29], and [27] with 15,968 WSDL files which was crawled from Seekda (Section 2.3). In addition, we extract the repository

[9] `http://www.programmableweb.com/`
[10] `http://www.xmethods.com/`
[11] `http://www.service-repository.com/`

available at http://www.wsdream.net/dataset.html which is used in [67] and [68] containing 16,514 WSDL files from different Web Service portals and search engines such as Seekda (Section 2.3).

However, these portals and repositories contain services which are not available anymore or changed (Section 6.1). We validate them during snapshots creation (Section 3.3.2).

## 3.3 Collecting Strategies

In Section 3.2 we discussed different kinds of sources and approaches which we use to collect different kinds of service descriptions and contextual information which was discussed in Section 3.1. In this section we discuss different strategies concerning this collecting process and how we process the collected information.

### 3.3.1 Duplications

In our approach as discussed, we find the service description URLs and store them in the service repository. However, there are many duplications from different sources of the data. Therefore, in each step of crawling, searching and extracting we need to check whether the URL which is found is a duplicate. In addition, we do not want to discard the duplications because their context may be useful, if they are from a different source. As a result, each time a service description URL is found, we first check whether it is already available in the repository. If it already exists in the repository, we update the context and the source of the data otherwise, we store it as a new service. For instance, a service description which is found from Google Search, may be already added to the repository from a known repository. However, its context such as the description could be different and needs to be merged to the previous description. This approach's algorithm will be discussed in Section 5.3.2 and elaborated in Algorithm 2.

### 3.3.2 Snapshots

In the previous sections we argued our approach to find and store service description URLs and their related context. For the Service Classification, we need the actual descriptions to analyze their content. Therefore, we request the URLs and retrieve the description and archive a copy of the file in a file repository. However, as our goal is to keep the results with respect to changing nature of services, each time we start to create snapshots, we send the request and retrieve the file for all URLs in the service repository and tag the result with the time because at the time of request the services may change. In other words, each snapshot is a copy of the service description at the time of snapping and it is the information we have at that time. Accordingly, when we receive the result, it is first validated and then if it is a valid result, we compare it with the last snapshot to check whether it is updated. If it is updated, this will be a new snapshot for the service otherwise, we just update the availability of the service. On the other hand, if a URL does not respond or returns a result which is not a valid result, we still keep the URL in the service repository for further requests because, the service provider server may not be working at that specific time. Another reason for keeping these snapshots is that, they can be analyzed to find the service availability and to find the changes of a service to report and also to notify the Broker, if it is already using the service in different demands. In the WSDL and WADL case we retrieve the XML file and save a copy of it as a snapshot however, as discussed before in Section 3.1.2, REST services do not follow any well-structured format besides WADL. As a result, we retrieve the document or HTML page that the URL is pointing at and store it as a snapshot. In order to keep these results, the snapshot repository consists of a file archive which holds the actual files and another repository which contains the file addresses, the time of the snapping and some extra information which algorithm will be discussed in Section 5.3.3 and elaborated in Algorithm 3.

## 3.4 Summary

As discussed in Chapter 1, a repository of Web Services which updates its services automatically is necessary inside a Broker or a vertical search engine focused on Web Services. In this chapter we discussed our methodology of collecting both kinds of Web Services (WSDL and RESTful) from the Web and storing them in a comprehensive and updatable repository. This chapter formed the collecting block in Figure 3. Accordingly, in Section 3.1, we discussed different kinds of service descriptions (WSDL, WADL, and Web pages describing RESTful services) and their contextual information which we collect using different collecting sources. In Section 3.2 we discussed different kinds of sources and approaches which we use to collect these descriptions and their contextual information. We discussed our methodology to extract the data from previous known repositories, to query Google, and to use Web Crawlers to find Web Services which are missed. Finally, in Section 3.3 we discussed different strategies concerning the collecting process and how we process the collected information and store it in the repository.

In the next chapter we will discuss our Service Classification methodology and how we classify the service descriptions and their contextual information which we find.

# Chapter 4

# Service Classification

In the previous chapter we discussed harvesting and storing Web Service descriptions and their contextual information from different sources. In this chapter we argue the next step, automatic ***Service Classification***. Service Classification or Categorization is the task of associating Web Service descriptions to a predefined set of categories which can considerably speed up and increase the effectiveness of the task of finding compatible Web Services in Brokerage or suggesting Web Services to the requests [6].

Categories or classes specify the purpose of the service and what it does at a high level. However, there is no structured support for specifying the abstract category to which the service belongs [6]. As a result, this classification task needs to be done manually or automatically.

As argued in [30], human intervention provides more quality annotations which requires more effort. The authors in [30] present the idea of crowd sourcing to decrease the costs and effort. However, it is still much more expensive than automatic classification even though the precision is considerably higher.

In this thesis due to the high number of service descriptions (Section 6.1) and the lack of resources, we use automatic classification. We build on the considerable amount of research that has been carried out on the topic of automatic classification of a text document which has many practical applications [105].

The task of automatic classification of documents is usually tackled by applying

*Machine Learning* techniques. These techniques use classifiers that have been automatically induced by estimation on a collection of documents which is called the *training set* [106]. Machine Learning methods can be divided into two broad categories:

**Supervised Learning**

Each document in the training set is already associated with a category by a human supervisor.

**Unsupervised Learning**

Documents are not associated with a category prior to the learning process and the Machine Learning method must find a meaningful division into categories.

In this thesis we focus on the former method, which has generally been much more successful in most studies as pointed out in [6].

We use the open-source MARF framework and its MARFCAT application because they are designed as an input media type-independent investigation platform to execute a considerable number of experiments in a short amount of time and to assist selecting the best combinations of different available algorithms. In this application, we use signal processing techniques which use character-level (bi-grams) processing rather than syntax and semantic levels and we treat the descriptions as a *signal* which will be discussed in details in Section 4.1. In Section 4.1.3 and Section 4.1.4 we discuss different algorithms and options available in MARFCAT.

Using MARFCAT as our investigation platform, we systematically test and select the best (a tradeoff between accuracy, recall, and speed) combination(s) of algorithm implementations (configuration) available to us for each type of service descriptions (Section 4.2) and then use only those for the final classification of all service descriptions based on the classes defined in Section 4.3. We will discuss our methodology in Section 4.4.

This chapter forms the classification block in Figure 3.

## 4.1 MARF and MARFCAT

*Modular Audio Recognition Framework* [107], or short termed as *MARF*, is an open-source collection of pattern recognition APIs and their implementation for unsupervised and supervised Machine Learning and classification. MARF was designed to act as a testbed to verify and test common and novel algorithms found in literature for sample loading, pre-processing, feature extraction, and training and classification, which constitute a typical pattern recognition pipeline [4]. During the years that MARF was introduced, it accumulated a fair number of implementations for each of the pipeline stages which allows us to execute reasonably comprehensive comparative studies of algorithm combinations for the Service Classification purpose.

The conceptual pattern recognition pipeline shown in Figure 6 depicts the core of the data flow and transformation between the stages of the pipeline in MARF.

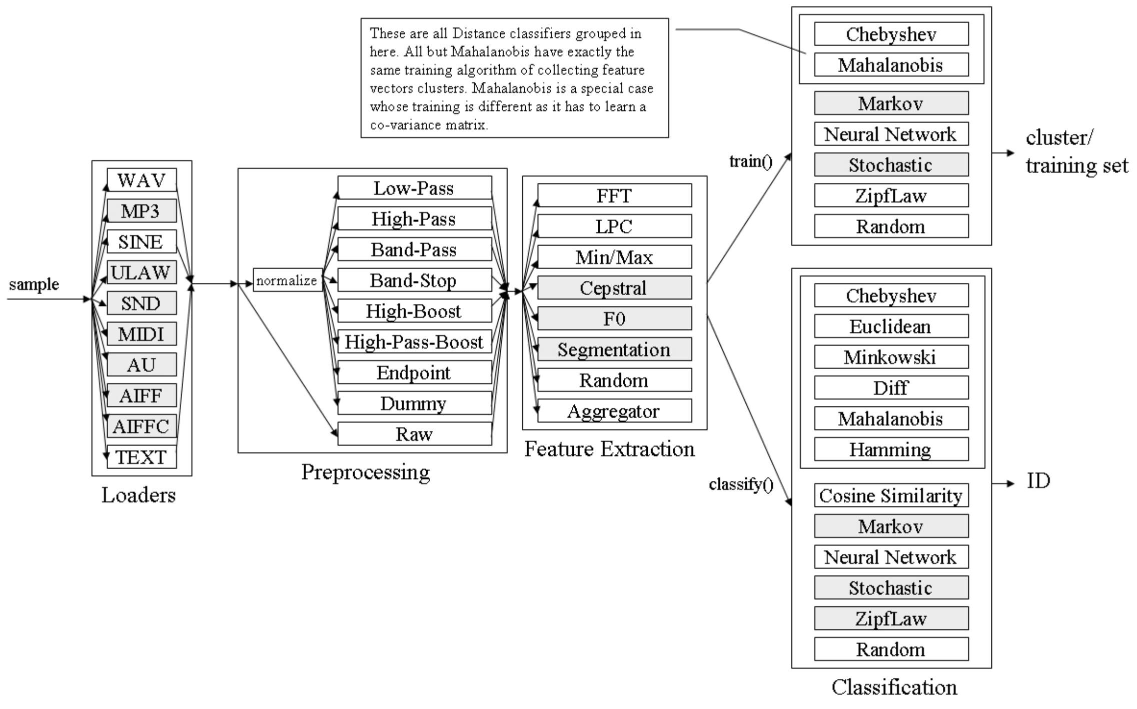


Figure 6: MARF pattern recognition pipeline [4]

The pattern recognition process starts by **loading** a sample (e.g., an audio recording, a text, or image file), removing noisy and/or silent data and other unwanted elements (**pre-processing**), then extracting the most prominent features from it (**feature extraction**), and finally either **training** the system such that the system learns a new set of features of a given subject or **classifying** what the subject is. The outcome of the training process is either a collection of some form of feature vectors or their mean or median clusters, which are stored for every subject learned. The outcome of classification is the class that the system believes the subject belongs to [4] and a score attached to it.

MARF and its application MARFCAT's design is independent on the input media being analyzed. The input is treated as a signal, equivalent to binary, where each $n$-gram ($n = 2$ by default, i.e., two consecutive characters or, more generally, bytes) are used to construct a sample amplitude value in the signal.

### 4.1.1 Signal Pipeline

The loading in MARF starts with the interpretation of the files being scanned in terms of bytes forming amplitude values in a signal using either unigram, bi-gram, or tri-gram approach. Then, the pre-processing allows to be *none-at-all* (raw, or the fastest), *normalization*, traditional frequency domain filters, wavelet-based filters, etc. Feature extraction involves reducing an arbitrary length signal to a fixed length feature vector of what is thought to be the most relevant features in the signal, e.g., spectral features in FFT, LPC, min-max amplitudes, etc. The classification stage is then separated either to train by learning the incoming feature vectors (usually $k$-means clusters, median clusters, or plain feature vector collection, combined with neural network training) or testing them against the previously learned models [108].

*MARFCAT* is a MARF-based Code Analysis Tool, which was first exhibited at the Static Analysis Tool Exposition (SATE) workshop in 2010 [109, 110]. MARFCAT, as any MARF application, can be used for a wide array of recognition tasks, not only applicable to audio, but rather to general pattern recognition for various applications, such as in digital forensic analysis, writer identification, natural language processing (NLP) [111], and others. In particular, MARFCAT was used to analyze source and byte code to fingerprint, detect, and classify vulnerabilities and weaknesses in [112, 113, 114] and do the same for network packet traces [115]. The authors point out MARFCAT's advantages and shortcomings:

- Advantages
  - Relatively fast (e.g., 2400 files to train and test in about 3 minutes) on a now-commodity 7-year old desktop or a laptop.
  - Input data type-independent (e.g., sound files, binary and source code, images, and natural language text)
  - Can automatically learn a large knowledge-base to test on known and unknown cases.
  - A wide range of algorithms and their combinations can be investigated to select the best ones for a particular task.
- Shortcomings
  - Interpreting a signal is less intuitive by humans in the output.
  - Accuracy depends on the quality of the knowledge-base (training sets) collected. Some of this collection and annotation is manual; hence, error-prone and a subject to over-fitting.
  - No nice GUI. Presently the application is script/command-line based (however, a scalable web service-based UI development is in progress).

### 4.1.2 Motivation to Use MARFCAT

The following are primary motivations justifying the use of MARFCAT in this work:

1. MARFCAT was successfully used in related source code and text analysis tasks, for specific vulnerabilities and defects as well as more general weakness categories as referenced earlier. At its introduction in 2010, it was arguably the first time such an approach was applied to text analysis and was deemed novel in these types of tasks. The most significant advantage of it was the processing speed compared to other code analysis tools.

By extension this applied it to Web Services descriptions in various formats.

2. MARFCAT supports both signal processing and NLP pipelines. However, the signal pipeline was found by an order of magnitude faster than most parsing and NLP approaches [114, 113].

   Thus, spectral analysis was proven beneficial in code analysis, source, and binary as well as network packet traces, and natural language processing. It is analogous to analyze the signal from a distant star, breaking it down into spectrum of emitted light in order to classify the chemical composition in terms of elements present in the star, i.e., to fingerprint them. MARFCAT similarly fingerprints a spectrum of text or any other media into bins related to different categories it was shown to learn from.

3. MARFCAT is very easy to quickly setup and do preliminary testing in search for good algorithms. It can also be used as a front-end for semantic- and ontology-based parsing classifiers to prioritize their work [112].

4. MARFCAT author was readily available to consult on issues of its uses and operation.

### 4.1.3 Algorithms Used by MARFCAT

The specific algorithms come from the classical literature and other sources and are detailed in [4], [107], [115]. The below is a summary of some algorithms corresponding to Figure 6 with a brief description:

**Fast Fourier Transform (FFT)**

A version of the Discrete Fourier Transform used in FFT-based filtering as well as feature extraction [116]. It is also used in FFT-based filters (both forward and inverse FFT to reconstruct the signal after filtering). Uses 512 frequencies by default (empirically determined by the MARF project).

**Linear Predictive Coding (LPC)**

Used in feature extraction, which evaluates windowed sections of signals and determines a set of coefficients approximating the amplitude vs. frequency function. Uses 20 poles by default.

**Distance Classifiers**

Various distance classifiers (Chebyshev, Euclidean, and Minkowski [117], Mahalanobis [118], Diff (internally developed within MARF, roughly similar in behavior to the UNIX/Linux diff utility [119]), and Hamming [120]).

**Cosine Similarity Measure**

Cosine similarity measure [121], [122] was thoroughly discussed in [123] and often produces the best or near best accuracy in MARF in many configurations.

### 4.1.4 MARFCAT Options

As we mentioned before, the loading in MARF starts with the interpretation of the files being scanned in terms of 2-byte sequences (bi-grams) forming amplitude values in a signal over time. All mentioned algorithms are selected as options in a scripted manner exhaustively at the first stage in order to select the candidate best options for subsequent classification. Not all combinations are necessarily optimal or have effect together (e.g., noise removal uses low pass filter at pre-pre-processing and then if low pass filter is applied, it doubles the work, without additional filtering effect), but they are easy to automate and there is no dependency assumptions between algorithms at different stages keeping them decoupled and re-usable. Other options, such as clustering type, choice of signal or NLP pipeline, loaders, are also selected. We survey some of these options to find the best configuration for each type of service description which will be discussed in Section 4.4.

In order to be able to classify samples into different classes, an automatic classifier determines the salient properties of the samples and puts them into different feature vectors. This process is called feature extraction [6]. In MARF there are different ways of storing and matching feature vectors that MARFCAT takes advantages of from a specific class. These are referred to as clustering options in MARFCAT and can be customized:

- k-means clusters (mean option)
- median clusters (median option)
- plain feature vector collection (no clustering option)

We report our results with all three clustering options to find the best configuration for each type of service description which will be discussed in Section 4.4.

In terms of some most prominent algorithms producing the best results in the algorithm selection stage include, but not limited to:

**Preparation**

- `-noise` does noise removal by applying as an FFT (Fast Fourier Transform) low-pass filter effectively removing high-frequency occurring material.
- `-silence` removes near-zero gaps from the data. It is important to apply silence removal after the noise removal since noise filtering may produce more silence gaps. The gaps are removed by compression of the input data into a smaller sized array by cutting out and concatenating non-silent portions.
- `-silence-noise` combines the noise and silence removal. It helped selecting best low-frequency non-zero local minimums and maximum features in classification of less structured samples such as REST descriptions.

**Preprocessing**

- `-endp` is endpointing which collects all *local* minimums and maximums from the signal. It worked best with `-minmax`.
- `-low` FFT filter that removes approximately the upper 1/3 band of frequency spectrum by applying a zero frequency response on that portion effectively removing most high-frequency bigram material.

It is redundant to apply `-low` and `-noise` together under the current implementation of `-noise`, but scripting facilities do not make such intelligent guesses.

- `-bandstop` keeps approximately the lowest and highest 1/3 bands of the spectrum and removing the middle third. Combined together with the low-pass filter this effectively means 2/3 upper frequencies are removed keeping 1/3 of the lower-frequency band.

**Feature Extraction**

- `-minmax` picks a hundred features from the vector, where 50 are minimums and 50 are maximums. If there are less than 100 values, the gap is filled with zeros. Worked best with `-endp` to select the 100 local minimum and maximums.
- `-lpc` is Linear Predictive Coding which works on a spectral envelope of coefficients representing the spectrum curve. In MARF, the empirical default is 20 coefficients. It works well with compressed form of signal, such as with local minimums and maximums with silence removed. The feature vectors are as a result small – 20 features making distance calculation faster (as opposed to `-fft`'s 512 features).

**Classification**

- `-cheb` is Chebyshev distance classifier which appears to work best with the `-endp` and `-minmax` selected local extremes due to their block nature that provides enough discriminatory power for highly varied and overlapping data sets such as REST descriptions. It is also the fastest classifier.
- `-eucl` is Euclidean distance which works better with less varied 20-sized vectors, such as produced by LPC combined with endpointing. `-silence`, `-endp`, `-lpc` with Euclidean distance appear to produce one of the best configurations during search for algorithms to use for WSDL descriptions.

Other options and their algorithms, and their complexity are discussed in [4, 115] and related and are omitted here for brevity.

## 4.2 Data Samples

In Chapter 3 we discussed different types of Web Service descriptions and their contextual information. We stored each type's description and their context separately. Due to the nature of each type's description's characteristics and features, and to compare and analyze the results separately, we survey each type's classification process independently.

For WSDL and WADL files we use the descriptions directly which we found from the previous step. On the other hand, for HTML files, descriptions regarding to REST services, we take an additional step before feeding them to MARFCAT because of the nature of these files which contain too much noise, e.g., *script* codes.

In this step we remove all the HTML tags and unnecessary sections and only keep the raw text inside and store it in a separate text file and consider it as a new type of sample. Likewise, this step can be applied to WSDL and WADL files to remove all their tags. However, they do not contain much noise and MARFCAT will take care of noise removal in the pre-processing step. Therefore, this task were postponed to the future work because it will multiply the number of the tests to be applied.

As a result, our repository will contain four general types of samples:

- WSDL files (.wsdl)
- WADL files (.wadl)
- HTML files (.html)
- Tags-Filtered description files (.txt)

Another dimension which is added to each of these types is their contextual information. In order to show the effect of context on the classification and to find the best configuration, we define three type of samples with respect to the contextual information for each of general types defined above:

- Plain files (description files without any context added to them)
- Combined with context files (plain descriptions + context)
- Only context files (files containing only the contextual information of service descriptions)

Figure 7 illustrates abstract data model of the samples:

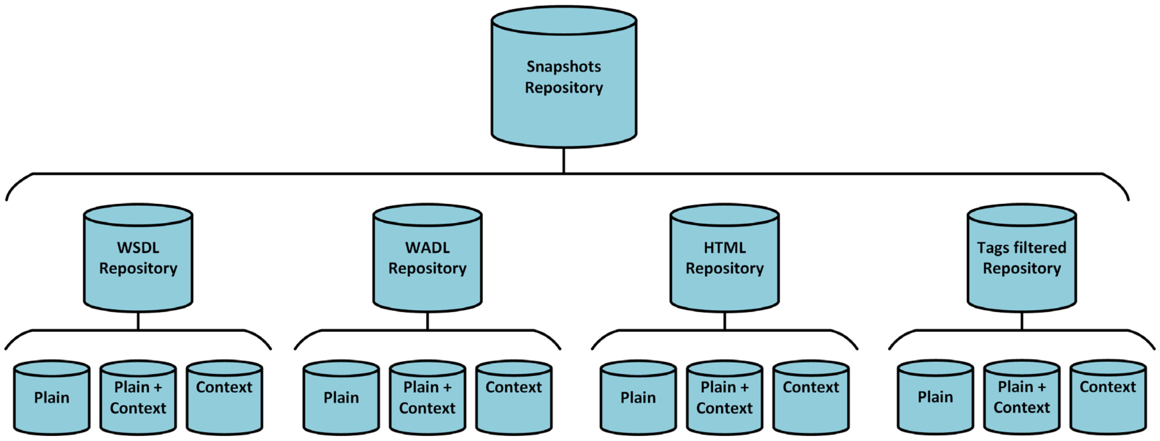


Figure 7: Abstract Snapshots Data Model

The concrete data model is shown in Figure 13. These samples will be loaded into MARF in the data flow shown in Figure 14.

## 4.3 Classes and Training Sets

We use 5 classes for the classification with respect to previous research [70], [27], [67], the most popular categories in ProgrammableWeb[1] and more importantly the nature of Web Service descriptions in the repository and their intersections:

- Weather
- Social

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

- Tourism
- Entertainment
- Financial

As mentioned in Section 4.1, in order to classify the service descriptions which are stored in the repository, we need *training sets* for each of these classes. These sets need to be chosen with minimal intersections to be definite candidates for the class. In addition, for the testing purposes to find the best combination of configurations which will be discussed in Section 4.4, we need *testing sets* for each of the classes. Therefore, we manually classify 500 instances (100 per each class) for WSDL and REST files. However, for WADL files because of the inadequacy of the files in the repository as discussed in Section 3.1.3, only for weather, social and financial 10 definite matches can be found. For tourism 3 candidates and for entertainment only 2 candidates are chosen.

## 4.4 Testing Methodology

As discussed before in Section 4.1, we use MARF and its application MARFCAT to find the best algorithm combinations for each service description types which were mentioned in Section 4.2. In order to perform this task, MARFCAT defines two processes which were discussed in Section 4.1:

**Learning Process**

In this process the feature vectors are extracted and the system learns the classes from the *training set*.

**Testing or Classification Process**

In this process the *testing set* is classified based on the previously learned models.

In order to evaluate the performance of the classifiers, we compute different evaluation measures. These measures are usually presented as percentages. Consider for a given class $C$, $n_t$ samples are expected, i.e., are labeled with $C$. The classification system classifies (labels) $n_s$ samples as $C$ including $n_c$ correct samples (*true positives*) and $n_n$ incorrect samples (*false positives*).

**Total Accuracy**

Total accuracy is defined as the fraction of the samples which were classified in the same class as expected in total:

$$\frac{\sum n_c}{\sum n_t}$$

**Precision**

Precision is defined for each class as the fraction of classified items which are relevant, i.e., expected in that class:

$$\frac{n_c}{n_s}$$

We also compute the *macro precision* which is the average of precision over all classes.

**Recall**

Recall (also called sensitivity) is defined for each class as the fraction of relevant items which are classified:

$$\frac{n_c}{n_t}$$

We also compute the *macro recall* which is the average of recall over all classes.

**F-Measure**

F-Measure is defined for each class as the harmonic mean of precision and recall:

$$2 * \frac{precision * recall}{precision + recall}$$

We also compute the *macro F-Measure* which is the average of f-measure over all classes.

#### Classification Time

Classification time is the execution time of the classification process.

Appendix B contains examples of the reports.

In our methodology, initially we survey each description type (Section 4.2) independently in order to find the best algorithm combinations considering the measurements.

In addition, there are also other options which MARFCAT provides and were referred in Section 4.1.4 as clustering and frequency options. We test all three clustering options in all cases to find the best algorithm combinations. On the other hand, because the frequency change did not have any effect on the precision (see Chapter 6) when tested in the best case, we ignored it for the other cases.

Therefore, we find the best configuration of MARFCAT for each sample types which consists of an algorithm combination using a specific clustering method.

As argued in Section 4.3, we chose 5 classes and we manually classified 500 instances (100 per class) for each service description type. As discussed in Section 4.2, contextual information adds another dimension to the samples and adds 2 more sample types (plain + context, and only context) for each of the service description types.

In order to completely survey the possible cases and find the best configuration(s), we train and test on all type of samples exhaustively.

Figure 8 illustrates our methodology which forms 72 different cases based on the sample types and clustering options. Each case consists of one row from each of the blocks; one description type, training and testing on which type of file considering contextual information. We test 864 algorithm permutations in the Signal Processing pipeline for each case which will be discussed in Section 5.3.4 and illustrated in Figure 14.

In this exhaustive test process we choose randomly a smaller set of the manually classified instances in order to keep the tests simple and applicable in a shorter amount of time (62,208 runs).

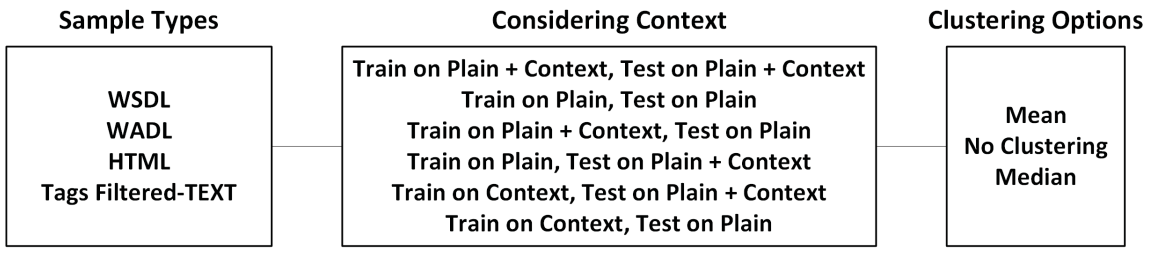

Sample Types
WSDL
WADL
HTML
Tags Filtered-TEXT
Considering Context
Train on Plain + Context, Test on Plain + Context
Train on Plain, Test on Plain
Train on Plain + Context, Test on Plain
Train on Plain, Test on Plain + Context
Train on Context, Test on Plain + Context
Train on Context, Test on Plain
Clustering Options
Mean
No Clustering
Median

Figure 8: Testing Methodology

After finding the best cases, we increase the sets and use all 500 instances for each of service description types. Using these sets we perform another exhaustive search on the algorithm combinations in order to find the best algorithm combination.

Using the best case and the best algorithm combination, we perform a 10-fold cross-validation in order to give an insight on how the model will generalize to an independent dataset and to reduce variability. In this procedure, we split the data randomly into 10 pieces and run the classification 10 times using one of the pieces (10%) as the testing set and the rest (90%) as the training set in a way that each sample is present once and only once in the testing set among the runs and then, average all the results.

In order to show the effect of the use of context on the classification, we study the best configuration without any contextual information added, and the best one through all other cases with context. As a result, we perform the 10-fold cross-validation for each of these cases and compare the results in order to illustrate the effect of context.

At the end, after finding the best configuration for each sample type, we create the sets from all the service descriptions in the repository which are not classified yet and perform the classification for them and store the results in the repository.

## 4.5 Summary

As discussed in Chapter 1, Service Classification can considerably speed up and increase the effectiveness of the task of finding compatible Web Services in Brokerage or suggesting Web Services to the requests. In this chapter we discussed our methodology of combining Machine Learning and Signal Processing techniques and employing contextual information in order to automatically classify Web Service descriptions. This chapter formed the classification block in Figure 3. Accordingly, in Section 4.1 we introduced the open-source MARF framework and its MARFCAT application which we use as our investigation platform in order to execute a considerable number of experiments in a short amount of time and to assist selecting the best combinations of different algorithms. In Section 4.2 we discussed different type of samples which we use for the classification including the service descriptions and tags-filtered version of them. We argued adding another dimension to the descriptions and defining samples with respect to the contextual information. In Section 4.3 we discussed choosing the classes and the training and testing sets for the classification. Finally, in Section 4.4 we discussed our methodology to find the best combinations of algorithms and options from the viewpoint of accuracy, recall, and time achievable in the scope of this thesis and to survey the effect of context on the Service Classification. We defined 72 different cases based on the sample types, clustering options, and the contextual information which we survey 864 combinations of algorithms and techniques in each.

In the next chapter we will discuss the architecture, models, and the implementation of this thesis.

# Chapter 5

# Prototype

In Chapter 3 we discussed the concepts involved in how we collect Web Service descriptions and their contextual information. In Chapter 4 we argued our methodology to find the best classification configuration and classifying the repository. In this chapter we discuss the architecture, models, and the implementation of these processes.

## 5.1 Architecture

Figure 3 illustrates the whole architecture and how the two steps of service collecting and Service Classification are connected.

### 5.1.1 Service Collecting Architecture

Figure 9 demonstrates the service collecting concrete architecture and components. As discussed in Chapter 3, we collect service descriptions from three sources. For each of these sources we define independent components:

**Known Repositories Extractor**

This component extracts service description URLs, tags, annotations, categories, descriptions and titles from the result of previous research and service portals available in the Web as discussed in Section 3.2.3.

**Google Search**

This component employs Google Search using a query based on keywords and different file types which were discussed in Section 3.2.2. Accordingly, it extracts URLs, descriptions and titles from Google results.

**Web Crawler**

This component crawls the Internet in order to find Web Service descriptions. As discussed in Section 3.2.1, it starts from the crawling seeds which are populated with a set of provider domain URLs from which a high number of Web Services has been found.

In addition, the *Service Descriptions Merger* is responsible to merge the results of each of these components and store them in the service descriptions repository. This component prevents duplications in the repository and aggregates contextual information from different sources as discussed in Section 3.3.1.

The *Provider Extractor* extracts service provider domain URLs from service description URLs and stores them based on the number services which they offer.

The *Snapshots Creator* sends a request to each of the service description URLs and as discussed in Section 3.3.2, if the result is valid, it archives a copy of the file in the snapshots repository and stores the accessed time and the reference to the file in the snapshots info part of the service descriptions repository.

### 5.1.2 Service Classification Architecture

Figure 10 depicts the Service Classification concrete architecture and components. As discussed in Section 4.4, initially we test each sample type (Section 4.2) independently in order to find the best configuration of MARFCAT for that sample type.

At the end, after finding the best configuration for each sample type, we create the testing sets for each type from all the service descriptions in the repository which are not classified yet. Then we perform the final classification for each type based on the best configurations (algorithm combination + clustering method) found from

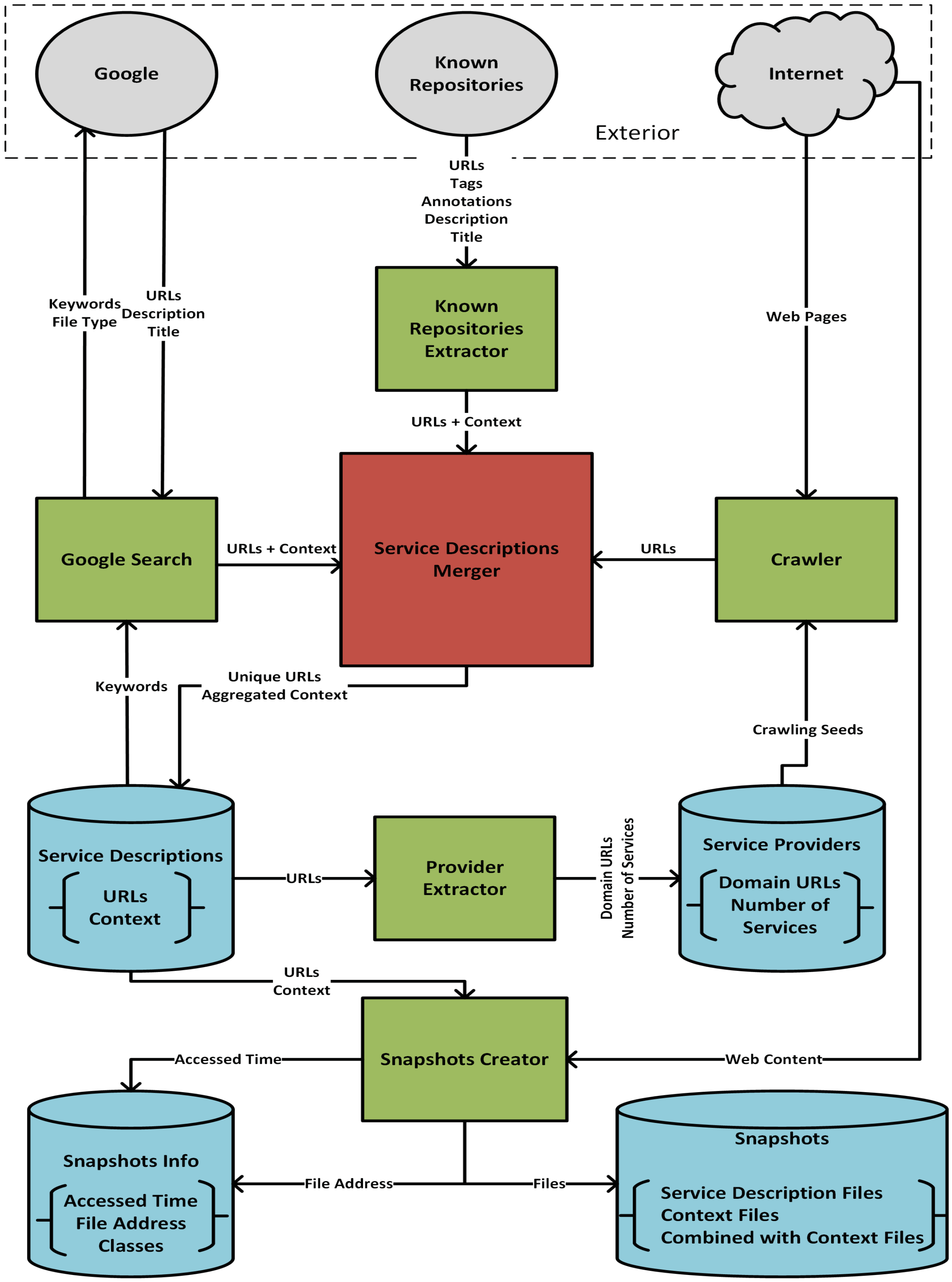


Figure 9: Concrete Service Collecting Architecture

the previous step and store the resulting classes associated with each snapshot in the snapshots info part of the service descriptions repository.

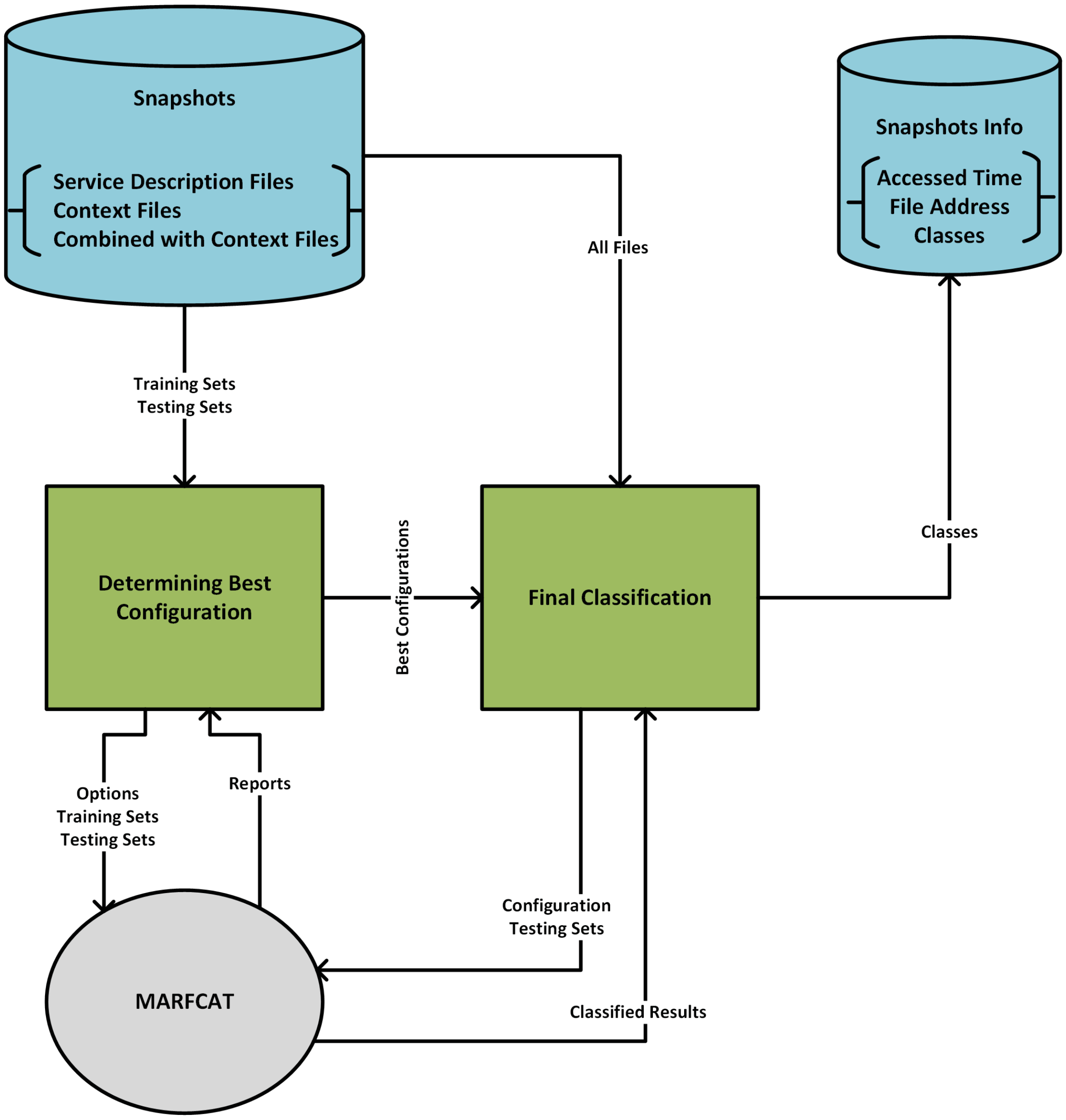


Figure 10: Concrete Service Classification Architecture

## 5.2 Data Model

Our service repository consists of two main repositories:

**Service Descriptions Repository**

Stores service description URLs, providers, contextual information, and snapshots information and references. This repository is implemented as a SQL database. The main characteristics of this repository is illustrated in Figure 11 and the concrete entity relationship diagram is illustrated in Figure 12.

**Snapshots Repository**

This repository is a file-based repository which stores snapshots of service descriptions, context files, and snapshots of service descriptions combined with their context files. Each snapshot is stored in a folder named with its service provider URL and linked to Snapshots Info in service descriptions repository. The abstract illustration of this repository is shown in Figure 7 and the concrete data model is illustrated in Figure 13.

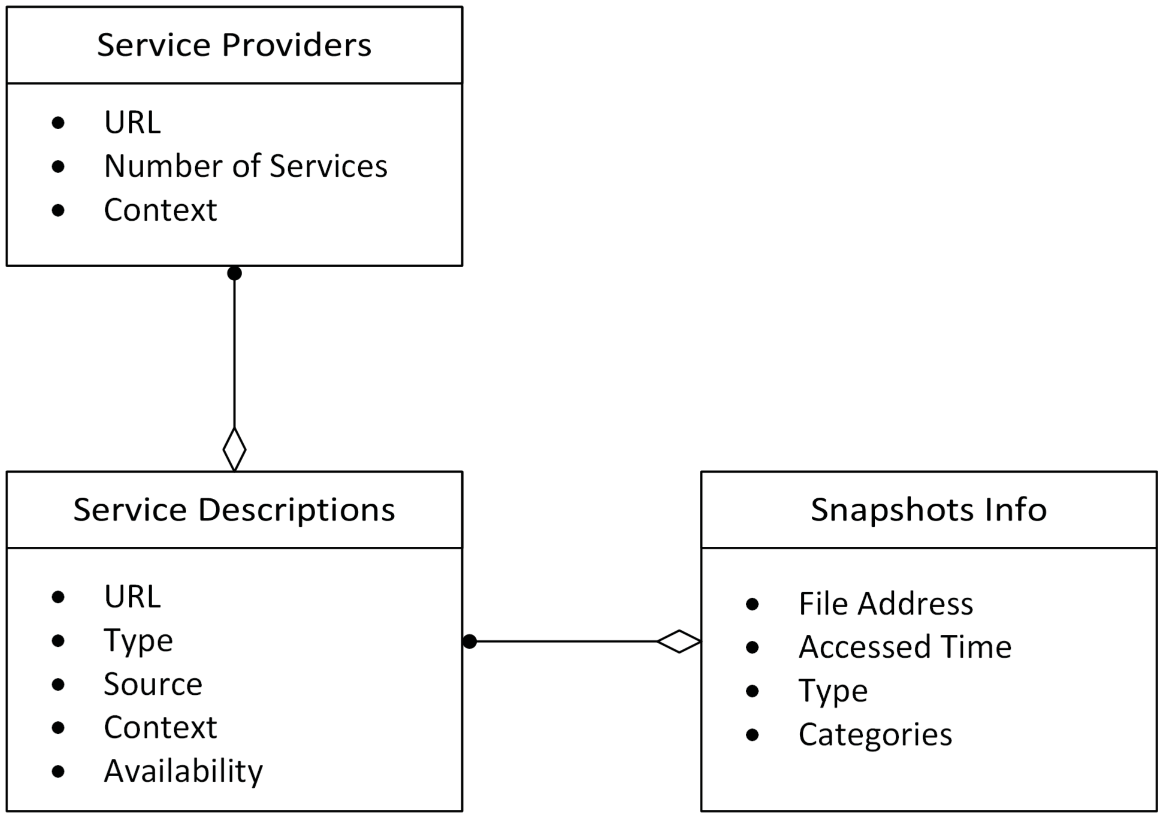


Figure 11: Abstract Entity Relationship Diagram

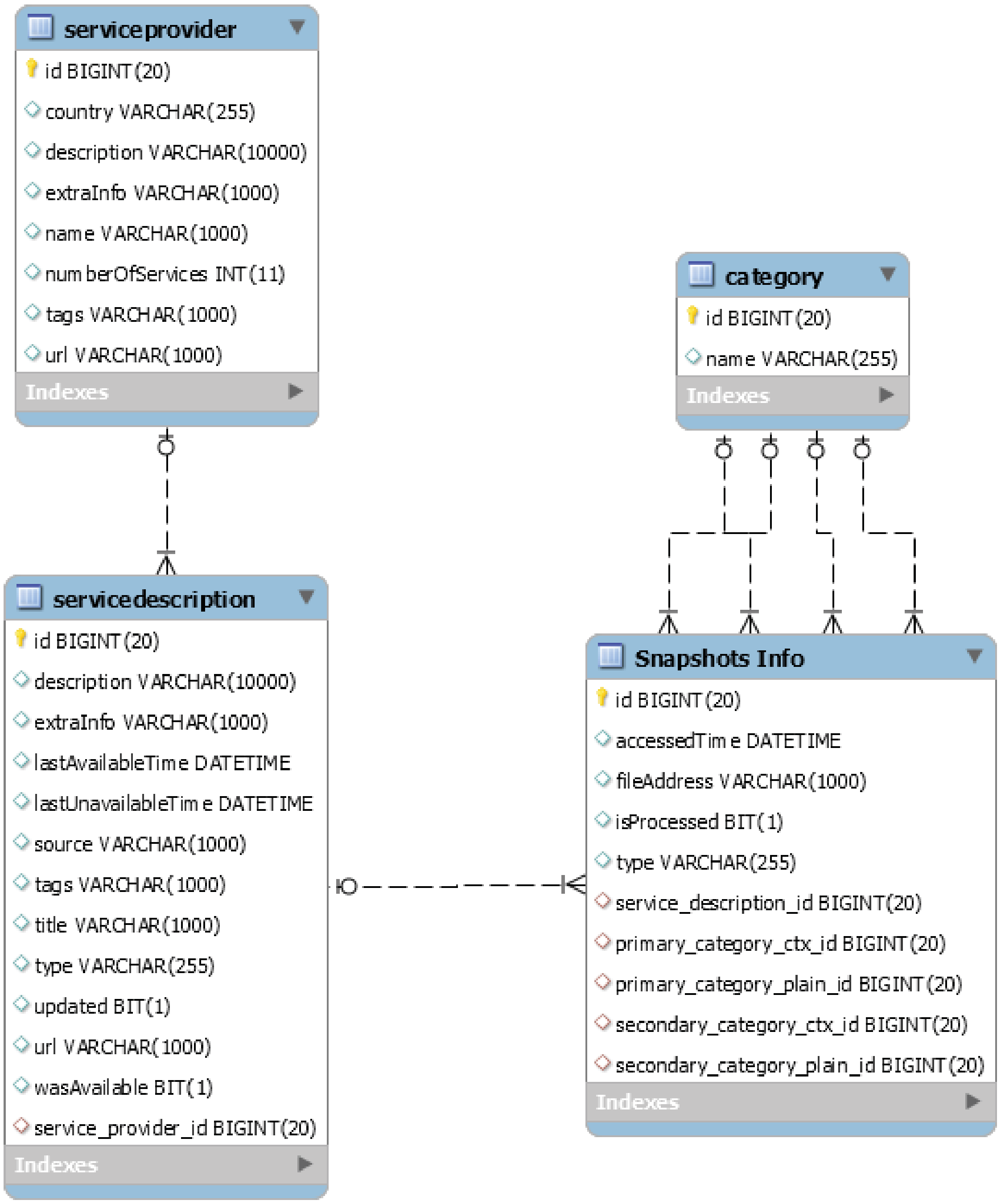


Figure 12: Entity Relationship Diagram

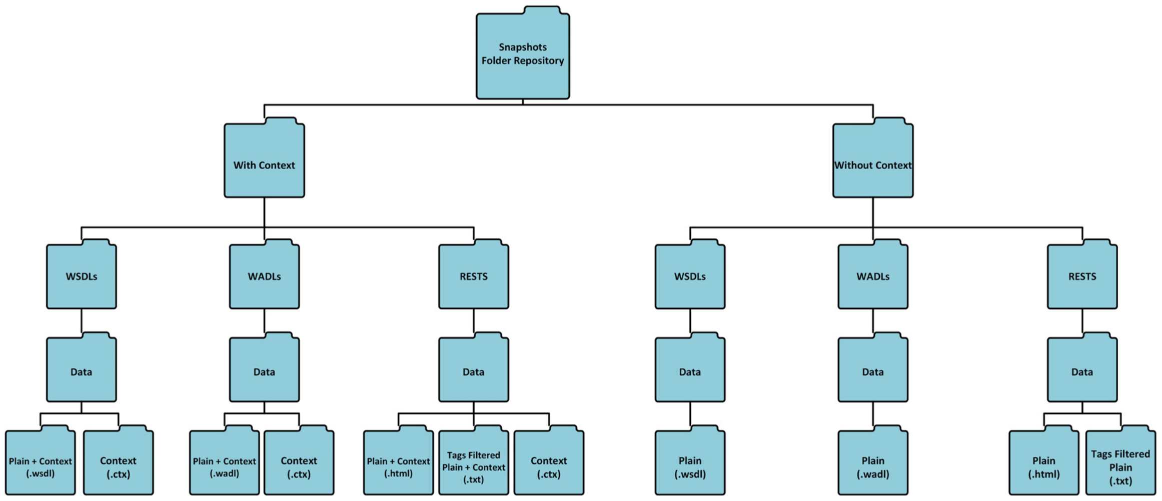


Figure 13: Concrete Snapshots Data Model

## 5.3 Implementation

All of the components are implemented in Java. The Service Descriptions Repository was created using MySQL and Hibernate as its ORM (Object-Relational Mapping) framework. In this section we go further into the details of the components which need more discussion.

### 5.3.1 Web Crawling

Our Web crawling methodology was discussed in Section 3.2.1. In order to perform this task, we use Crawler4j[1], an open-source Web Crawler for Java which provides a simple interface for crawling the Web. This crawler is easy to setup and customize and supports all of our crawling policies defined in Section 3.2.1.

In order to define the Politeness Policy and Parallelization Policy, we set the configuration of Crawler4j with a specific Politeness Delay and Number of Crawlers. This crawler offers different methods to be overridden in order to inject the policies of crawling. In this section we discuss two of main functions and the algorithms implemented in them.

#### Should Visit

[1]https://github.com/yasserg/crawler4j

This function decides whether the given URL should be crawled or not. In this function we define our Selection Policy to reject a lot of content by default especially binary files such as images, audio or video files and to just fetch HTML pages, XML files and other text documents, i.e., all types of files that could be either a service description or lead to a service description (WADL or WSDL).

**Visit**

This function is called after the content of a URL is downloaded successfully. In this function we first determine whether the downloaded content is a valid XML and then validate it through WSDL and WADL schema definitions. If it is a valid description and not already stored in the repository, we store its URL in the service descriptions part of the repository or update its context and availability as illustrated in Algorithm 2.

Algorithm 1 elaborates a simplified abstraction of these crawling algorithms.

**Algorithm 1** Crawling Algorithm

```
Require: url, filters
1: if url matches filters then
2:     Discard url
3: else
4:     content ← Download url Content
5:     if content type is xml then
6:         if content is a valid wsdl or content is a valid wadl then
7:             call Algorithm 2 with url
8:         end if
9:     end if
10: end if
```

## 5.3.2 Merging and Duplications

As defined in Section 5.1.1, the Service Descriptions Merger is responsible to merge the results of the three different collector components. We discussed how we prevent duplications and how we aggregate contextual information from different sources in

Section 3.3.1. In order to achieve this goal, Algorithm 2 is defined to handle saving or updating a URLs and its context. This algorithm is also used during merge and migration of repositories created by each collector, if they were running in a different environment, i.e., in a different server or with a different version of repository.

**Algorithm 2** Saving or Updating Service Descriptions Algorithm

```
Require: $newUrl$, $newContext$
 1: if $newUrl \in$ Service Descriptions Repository then
 2:     $existingContext \leftarrow$ context of $newUrl$ in Service Descriptions Repository
 3:     if $newContext \equiv existingContext$ then
 4:         Discard $newUrl$
 5:     else
 6:         $existingContext \leftarrow newContext \cup existingContext$
 7:     end if
 8: else
 9:     Save $newUrl$ with $newContext$ in Service Descriptions Repository
10: end if
```

### 5.3.3 Creating Snapshots

As defined in Section 5.1.1, the Snapshots Creator is responsible to create and update snapshots from the URLs saved in the repository. Algorithm 3 elaborates a simplified abstraction of the algorithm which is used to create or update snapshots and availability.

### 5.3.4 Service Classification

As mentioned in Chapter 4, we use MARF and its application MARFCAT to find the best algorithm combinations for each sample type which were mentioned in Section 4.2. We use the *fast* script of MARFCAT which performs the algorithms illustrated in Figure 14 in each step of its pipeline: 1 Loader, 4 techniques in the Preparation stage, 9 algorithms in the Pre-processing stage, 4 algorithms in the Feature Extraction stage, and 6 Distance Classifier algorithms. The combination of these algorithms will result in 864 permutations which we test in each of our cases as discussed in Section 4.4.

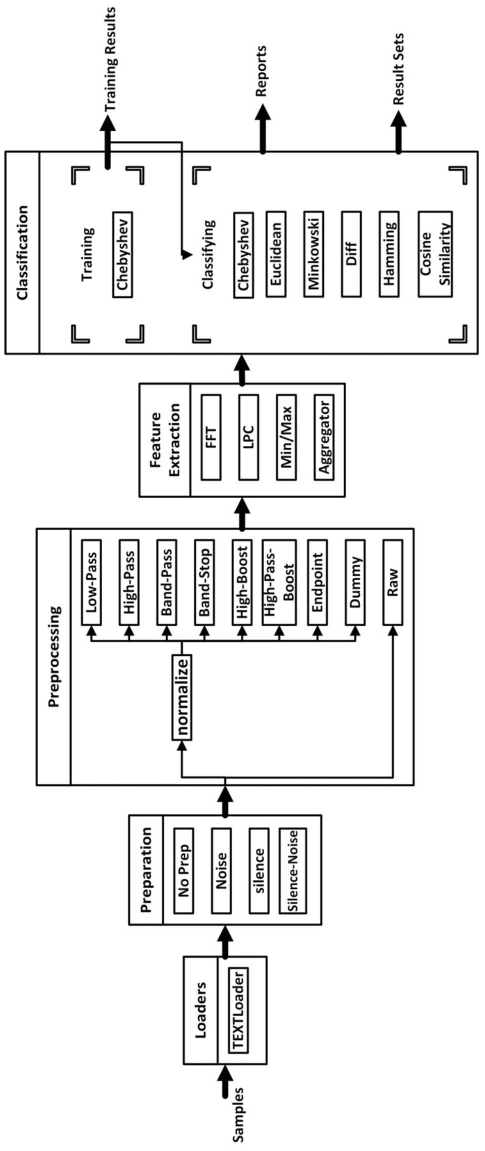


Figure 14: Service Classification Data Flow in MARF

**Algorithm 3** Creating Snapshots Algorithm

```
1: for all $url \in$ Service Descriptions Repository do
2:   $content \leftarrow$ Download $url$ Content
3:   if $content$ is valid then
4:     $lastAvailableTime$ of $url \leftarrow now$
5:     $infos \leftarrow$ Sort Snapshot Infos based on $accessedTime$
6:     $lastInfo \leftarrow$ First of $infos$
7:     $lastContent \leftarrow$ Load Content from $address$ of $lastInfo$ from Snapshots Repository
8:     if $lastContent \neq content$ or $\neg$ ($\exists$ $lastContent$) then
9:       $newInfo \leftarrow$ Create new Snapshot Info
10:      $accessedTime$ of $newInfo \leftarrow now$
11:      $newFile \leftarrow$ Create file with $content$ in Snapshots Repository
12:      $address$ of $newInfo \leftarrow$ Address of $newFile$
13:      Save $newInfo$ in Service Descriptions Repository
14:    else
15:      $accessedTime$ of $lastInfo \leftarrow now$
16:    end if
17:  else
18:    $lastUnavailableTime$ of $url \leftarrow now$
19:  end if
20: end for
```

## 5.4 Summary

In this chapter we discussed how we implement the prototype of Service Collecting which was discussed in Chapter 3 and Service Classification which was discussed in Chapter 4. Accordingly, in Section 5.1 we proposed the architecture of collecting and classification parts. We defined different components which are used in the architecture and illustrated the relationship and connection between them. In Section 5.2 we proposed the data model and the structure of the service repository which consists of two main repositories: Service Descriptions Repository and SnapshotsRepository. In Section 5.3 we discussed the details of the implementation and proposed different algorithms in order to implement the components.

In the next chapter we will present the results of Service Collecting and Service Classification and evaluate them.

# Chapter 6

# Results and Evaluation

In this chapter we present the results of Service Collection discussed in Chapter 3 and the Service Classification results discussed in Chapter 4 using the prototype which was implemented using the concepts argued in Chapter 5. Accordingly, in Section 6.1 we measure and illustrate the number of Web Service descriptions and their providers which we found and stored in the repository in total and based on each source. As discussed in Chapter 3, our first hypothesis is:

**Hypothesis 1**

By using different sources and combining the techniques, we shall create a comprehensive repository from all three main types of service descriptions and find more available service descriptions in comparison with the current repositories accessible.

As discussed in our methodology in Section 4.4, in order to classify the service descriptions which we found, we first find the best configuration of MARFCAT (best algorithm combination + clustering option) for each type of service description. In addition, we add the contextual information to the classification and our second hypothesis is:

**Hypothesis 2**

Adding contextual information to the files will improve the performance of the classification.

In Section 6.2 we measure and illustrate the resulting accuracies of 72 different cases based on the clustering options and adding contextual information, cross-validated results for the best cases, the effect of context on the classification, and we compare our results with the literature where it is available.

## 6.1 Service Collecting Results

Table 1 depicts the total results of service collecting including all three types of service descriptions (WSDL, WADL, and HTML pages describing REST services). Table 2, Table 3, and Table 4 present the results for each type of service description individually.

**Unique URLs**

This column contains the number of unique service description URLs which were found from each source.

**Available**

This column includes the number of service description URLs from each source to which we could send a request during snapshot creation and receive a valid response, i.e., successfully validated against WADL schema definition, WSDL schema definition, or a valid HTML page.

**Snapshots**

This column represents the number of snapshots which we stored from the URLs found from each source.

**Service providers**

This columns depicts the number of service providers which we extracted from the URLs found from each source.

The last row contains the total number of each column's concept regardless of the source of the data.

Table 1: Total Service Collecting Results

| | **Unique URLs** | **Available** | **Snapshots** | **Service Providers** |
|---|---|---|---|---|
| **Known Repositories** | 23474 | 10080 | 10158 | 11250 |
| **Search Engines** | 43494 | 27969 | 32554 | 8073 |
| **Web Crawling** | 6634 | 6634 | 6634 | 31 |
| **Total** | 72454 | 43576 | 48161 | 18494 |

Table 2: WSDLs Collecting Results

| | **Unique URLs** | **Available** | **Snapshots** | **Service Providers** |
|---|---|---|---|---|
| **Known Repositories** | 16955 | 4988 | 4988 | 5507 |
| **Search Engines** | 16543 | 5299 | 5299 | 1383 |
| **Web Crawling** | 6622 | 6622 | 6622 | 29 |
| **Total** | 39288 | 16096 | 16096 | 6791 |

Table 3: WADLs Collecting Results

| | **Unique URLs** | **Available** | **Snapshots** | **Service Providers** |
|---|---|---|---|---|
| **Known Repositories** | 0 | 0 | 0 | 0 |
| **Search Engines** | 1828 | 448 | 448 | 244 |
| **Web Crawling** | 12 | 12 | 12 | 2 |
| **Total** | 1830 | 450 | 450 | 246 |

Table 4: RESTs Collecting Results

| | **Unique URLs** | **Available** | **Snapshots** | **Service Providers** |
|---|---|---|---|---|
| **Known Repositories** | 6519 | 5092 | 5170 | 5787 |
| **Search Engines** | 25123 | 22222 | 26807 | 6587 |
| **Web Crawling** | 0 | 0 | 0 | 0 |
| **Total** | 31336 | 27030 | 31615 | 11680 |

Table 5 illustrates top 20 service providers which was extracted from service description URLs and the number of services each one offers.

### 6.1.1 Evaluation

There were 1,148 same URLs which were found from different sources due to the intersections between the sources. As a result, the total number (last row) is not equal to the sum of individual rows in some cases.

As discussed in Section 3.1.3, we were not able to find any known repositories which lists WADL file and as discussed in Section 3.1.2, because RESTful services are much harder to detect and validate, using Web Crawlers to discover HTML pages

Table 5: Top 20 service providers

| Provider | Number Of Services |
|---|---|
| data.serviceplatform.org | 6869 |
| github.com | 3888 |
| code.google.com | 2203 |
| stackoverflow.com | 1904 |
| programmableweb.com | 1101 |
| biomoby.org | 1048 |
| generalinterface.org | 1005 |
| svn.wso2.org | 505 |
| wsembnet.vital-it.ch | 484 |
| svn.apache.org | 456 |
| msdn.microsoft.com | 454 |
| wso2.org | 423 |
| books.google.ca | 355 |
| docs.aws.amazon.com | 315 |
| developer.atlassian.com | 303 |
| docs.atlassian.com | 282 |
| slideshare.net | 272 |
| community.workday.com | 267 |
| phoebus.cs.man.ac.uk | 242 |

describing REST services is out of the scope of this thesis. As a result, the Known Repositories row in Table 3 and the Web Crawling row in Table 4 is empty.

As Table 1 depicts, only around half of service description URLs which were found from known repositories were available and it shows that they are outdated. Similarly, only approximately 64 percent of service description URLs which were found from search engines were available and it shows that they are not guaranteed to find currently working Web Services. On the other hand, as we discussed in Section 5.3.1, as soon as we find a URL from Web crawling, we have to validate it to check whether it is a valid description URL (by checking its content). As a result, the Unique URLs column is the same as the Available column when Web Crawling is the source.

The task of storing snapshots as defined in Section 3.3.2, was performed at three different instances of time. The content from WSDL and WADL URLs did not change and as a result, there is no difference between the number of snapshots and the available URLs for them. On the other hand, because the HTML pages tend to change a lot over time, the content from the URLs which were pointing to HTML pages describing REST services changed in some cases and as a result, the number of snapshots is higher than the number of available URLs for them.

## 6.2 Service Classification Results

As discussed in Section 4.4, in order to classify the service descriptions, we first find the best configuration of MARFCAT (best algorithm combination + clustering option). In this approach as illustrated in Figure 4.4, based on the sample types, considering the contextual information, and the clustering options, we survey 72 different cases. As discussed in Section 5.3.4 and illustrated in Figure 14, we exhaustively test 864 algorithm permutations for each case. As discussed in Section 4.2, we classify and survey each service description type individually. For each service type, as discussed in Section 4.4, we survey 18 different cases in order to find the highest accuracy achievable without considering any contextual information (training on *Plain* files and testing on *Plain* files) and with the contextual information in effect (e.g., training on *Plain + Context* files and testing on *Plain + Context* files). In each case we use MARFCAT to test 864 algorithm permutations in order to find the highest accuracy. As mentioned in Section 4.4, after finding the best configuration, we perform two 10-fold cross-validations; one with the best case without considering any contextual information and one with the context added to the files. We compare the evaluation measures (total accuracy, macro recall and precision, macro F-Measure, and the classification time) in order to show the effect of context. In addition, we compare the performance of our classification for WSDL files with the literature in order to give an insight on how close our classification stands. However, we could not find any related work for classification of REST descriptions to compare.

### 6.2.1 WSDLs Classification Results

Table 6 depicts the highest accuracy achievable by exhaustively testing 864 algorithm permutations for each case of WSDL files using different clustering options which were mentioned in Section 4.1.4.

Table 6: The highest accuracy of the cases for WSDLs

| | **TrainPlainCtx TestPlainCtx** | **TrainPlain TestPlain** | **TrainPlainCtx TestPlain** | **TrainPlain TestPlainCtx** | **TrainCtx TestPlainCtx** | **TrainCtx TestPlain** |
|---|---|---|---|---|---|---|
| **Mean** | 60 | 52 | 48 | 52 | 36 | 28 |
| **No Clustering** | 68 | 64 | 76 | 64 | 36 | 36 |
| **Median** | 56 | 56 | 52 | 52 | 28 | 32 |

The highest accuracy without considering context (Train on Plain-Test on Plain column) is achieved by using the *No Clustering* option: 64 percent. The highest accuracy with considering context (other columns) is achieved by using the *No Clustering* option and using *Plain + Context* files in the training and *Plain* files in the testing: 76 percent.

For these cases as discussed in Section 4.4, we change the *frequency* option of MARFCAT which was mentioned in Section 4.1.4 in order to find better accuracy. However, as the results depicted in Appendix B.1, there is no change in the accuracy and we ignore this option for the types and cases.

As discussed in Section 4.4, after finding the best case which is using the *No Clustering* option and using *Plain + Context* files in the training and *Plain* files in the testing, we increase the sets and perform another exhaustive search in order to find the best algorithm combination.

As the results depicted, the best result for the classification of WSDL files is achieved by training on *Plain + Context* files and testing on *Plain* files using the following configuration:

- No Clustering option (discussed in Section 4.1.4)
- `-silence` for preparation, `-endp` for pre-processing, `-lpc` for feature extraction, `-eucl` for classification

As mentioned in Section 4.1.4, `-silence` option removes near-zero gaps from the data. There are usually many white-spaces and empty parts in the WSDL files that are normalized close to zero or silence gaps appear due to low-pass filtering. As a result this preparation technique helped to improve the overall classification combination. Additionally, as mentioned in Section 4.1.4 from a theoretical point of view, LPC works well with compressed form of signal, such as with local minimums and maximums with silence removed. In addition, Euclidean distance (which is sensitive to high-dimensional vectors) works better with less varied 20-sized vectors, such as produced by LPC combined with endpointing.

As discussed in Section 4.4, we perform 10-fold cross-validation based on this configuration in order to give an insight on how the model will generalize to an independent dataset and to reduce variability. Table 7 depicts the cross-validated results including the evaluation measures which were defined in Section 4.4. The class-wise results are listed in Appendix B.1.

Table 7: WSDLs cross-validated results with context

| **Total Accuracy** | **Macro Precision** | **Macro Recall** | **Macro F-Measure** | **Classification Time (ms)** |
|---|---|---|---|---|
| 59.00 | 59.65 | 59.00 | 58.62 | 3432.80 |

We compare the performance of our classification for WSDL files with the literature in order to give an insight on how close our classification process is to the literature. However, most of the research has been carried out for semantically-defined files (using Ontology Language (OWL-S) [45] and Web Service Modeling Ontology (WSMO) [46]) which are not available at a large scale and are a small subset of available service description files. The authors in [6] used different techniques for feature extraction such as Bag of Words variances and different algorithms for machine learning such as Support Vector Machines (SVM) variances and compared them in order to find the best classification performance for WSDL files. Finally, the best combination was the result of employing Support Vector Machines and use a feature extractor that is tailored to the task of WSDL classification by using its structure, in particular the identifiers. Table 8 depicts the results of their work including the same evaluation measures except the classification time which is not presented.

Table 8: WSDLs literature results form [6]

| **Total Accuracy** | **Macro Precision** | **Macro Recall** | **Macro F-Measure** |
|---|---|---|---|
| 59.40 | 58.00 | 52.80 | 55.30 |

Although the data sets from which the tests are performed are different, we can conclude that our classification accuracy is very close to the best result from the literature without any customization on the preprocessing, feature extraction, and classification based on the WSDL files. However, as the goal of this thesis was not to find the best classification tool available, and as such improving the classification accuracy is postponed to the future work. MARFCAT offers a good tradeoff between precision and speed and helps us to validate the hypothesis on the positive effect of context on classification results.

In order to show the effect of contextual information, as discussed in Section 4.4, we perform another 10-fold cross-validation on the same configuration without considering any contextual information and training on the *Plain* files and testing on the *Plain* files. Table 9 depicts the cross-validated results including the evaluation measures without considering any contextual information. The class-wise results are listed in Appendix B.1.

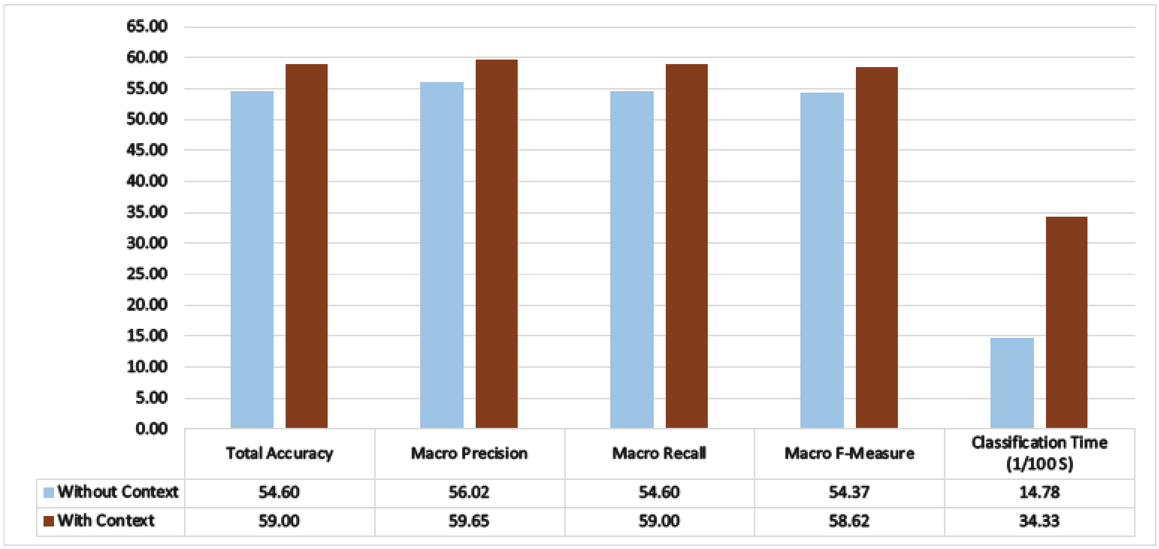
Table 9: WSDLs cross-validated results without context

| **Total Accuracy** | **Macro Precision** | **Macro Recall** | **Macro F-Measure** | **Classification Time (ms)** |
|---|---|---|---|---|
| 54.60 | 56.02 | 54.60 | 54.37 | 1478.00 |

Figure 15 illustrates the effect of adding contextual information to the WSDL files on total accuracy, macro precision, macro recall, macro F-Measure, and classification time.

The results depicts that contextual information is improving the performance of the classification even though it is increasing the classification time due to increase in the file sizes because of the context which is added to them.

Finally, we use the best configuration (using contextual information) in order to perform the final classification of all WSDL files which class is unknown. Table 10 depicts the number of instances which was classified inside of each class. These results are based on the performance of the current classification tool. Currently, the results cannot be verified because the actual classes are not known. In order to validate, all the instances need be classified by human contribution by using approaches such as crowd-sourcing.



| | Total Accuracy | Macro Precision | Macro Recall | Macro F-Measure | Classification Time (1/100 S) |
|---|---|---|---|---|---|
| Without Context | 54.60 | 56.02 | 54.60 | 54.37 | 14.78 |
| With Context | 59.00 | 59.65 | 59.00 | 58.62 | 34.33 |

Figure 15: Effect of context on classification of WSDLs

Table 10: Final WSDL classification results

| Category Name | Number of Instances |
|---|---|
| **Weather** | 2782 |
| **Social** | 2747 |
| **Tourism** | 3581 |
| **Financial** | 2376 |
| **Entertainment** | 4110 |

### 6.2.2 WADLs Classification Results

As discussed in Section 3.1.3, WADL descriptions are not popular through service providers and REST services are not widely described using WADL in the Web. As a result, as depicted in Table 3, we could not find as many instances for them as we could for WSDL and REST descriptions. Consequently, as discussed in Section 4.3, we could not find the same number of samples for WADL files. The scale of the samples are not as much as the WSDL and REST samples and we discard the 10-fold cross-validation for these files. However, for the sake of completeness we perform a 2-fold cross-validation (swapping training and testing set and averaging the results) and we compare the results with the context and without considering any contextual information. The results are depicted in Appendix B.2.

Table 11 depicts the highest accuracy achievable by testing algorithm permutations for each case of WADL files using different clustering options.

Table 11: The highest accuracy of the cases for WADLs

| | **TrainPlainCtx TestPlainCtx** | **TrainPlain TestPlain** | **TrainPlainCtx TestPlain** | **TrainPlain TestPlainCtx** | **TrainCtx TestPlainCtx** | **TrainCtx TestPlain** |
|---|---|---|---|---|---|---|
| **Mean** | 64.71 | 58.82 | 64.71 | 58.82 | 58.82 | 52.94 |
| **No Clustering** | 76.47 | 64.71 | 58.82 | 52.94 | 58.82 | 47.06 |
| **Median** | 58.82 | 58.82 | 58.82 | 58.82 | 52.94 | 41.18 |

The highest accuracy without considering context (Train on Plain-Test on Plain column) is achieved by using the *No Clustering* option: 64.71 percent. The highest accuracy with considering context (other columns) is achieved by using the *No Clustering* option and using *Plain + Context* files in both training and testing: 76.47 percent.

### 6.2.3 REST Files Classification Results

As discussed in Section 4.2, because HTML files contain too much noise, e.g., *script* codes, we define a new type of sample for REST HTML files and remove all the tags and unnecessary sections and only keep the raw text inside and store it in a separate text file. We survey both sample types in order to find the best case for classification of REST service descriptions and use the sample type with the highest accuracy in the final classification.

#### REST HTML Files Classification Results

Table 12 depicts the highest accuracy achievable by testing algorithm permutations for each case of HTML files describing RESTful services using different clustering options.

Table 12: The highest accuracy of the cases for the REST HTML files

| | **TrainPlainCtx TestPlainCtx** | **TrainPlain TestPlain** | **TrainPlainCtx TestPlain** | **TrainPlain TestPlainCtx** | **TrainCtx TestPlainCtx** | **TrainCtx TestPlain** |
|---|---|---|---|---|---|---|
| **Mean** | 40 | 40 | 40 | 40 | 32 | 40 |
| **No Clustering** | 40 | 44 | 48 | 36 | 40 | 36 |
| **Median** | 40 | 40 | 40 | 40 | 36 | 32 |

The highest accuracy without considering context (Train on Plain-Test on Plain column) is achieved by using the *No Clustering* option: 44 percent. The highest accuracy with considering context (other columns) is achieved by using the *No Clustering* option and using *Plain + Context* files in the training and *Plain* files in the testing: 48 percent.

## REST Tags-Filtered Files Classification Results

Table 13 depicts the highest accuracy achievable by testing algorithm permutations for each case of tags-filtered files describing RESTful services using different clustering options.

Table 13: The highest accuracy of the cases for the REST tags-filtered files

| | **TrainPlainCtx TestPlainCtx** | **TrainPlain TestPlain** | **TrainPlainCtx TestPlain** | **TrainPlain TestPlainCtx** | **TrainCtx TestPlainCtx** | **TrainCtx TestPlain** |
|---|---|---|---|---|---|---|
| **Mean** | 48 | 40 | 36 | 40 | 48 | 40 |
| **No Clustering** | 44 | 48 | 52 | 44 | 48 | 40 |
| **Median** | 40 | 44 | 48 | 40 | 44 | 40 |

The highest accuracy without considering context (Train on Plain-Test on Plain column) is achieved by using the *No Clustering* option: 48 percent. The highest accuracy with considering context (other columns) is achieved by using the the *No Clustering* option and using *Plain + Context* files in the training and *Plain* files in the testing: 52 percent.

## REST Files Best Configuration

As the results depicted, the best case for the classification of REST files is achieved by using the *tags-filtered* sample type and training on *Plain + Context* files and testing on *Plain* files and using the *No Clustering* option. As discussed in Section 4.4, after finding the best case, we increase the sets and perform another exhaustive search in order to find the best algorithm combination.

As the results depicted, the best result for the classification of REST files is achieved by using the following configuration:

- No Clustering option (discussed in Section 4.1.4)

- `-silence-noise` for preparation, `-endp` for pre-processing, `-minmax` for feature extraction, `-cheb` for classification (discussed in Section 4.1)

As mentioned in Section 4.1.4, `-noise` removes noise (high-frequency occurring material) by applying an FFT low-pass filter. It is important to apply silence removal after the noise removal since noise filtering may produce more silence gaps. As a result, `-silence-noise` which combines the noise and silence removal helped selecting best low-frequency non-zero local minimums and maximum features in classification of these non-uniformly structured files. Also, as mentioned in Section 4.1.4 from a theoretical point of view, `-minmax` which picks a hundred features from the data, where 50 are minimums and 50 are maximums, works best with `-endp` in order to select the 100 local minimum and maximums. In addition, Chebyshev distance classifier appears to work better with the higher-dimensionality of 100 features from `-endp` and `-minmax` selected local extremes due to their nature that provides enough discriminatory power for highly varied and overlapping RESTful services.

As discussed in Section 4.4 and similar to WSDL files, we perform 10-fold cross-validation based on this configuration in order to give an insight on how the model will generalize to an independent dataset and to reduce variability. Table 14 depicts the cross-validated results including the evaluation measures which were defined in Section 4.4. The class-wise results are listed in Appendix B.3.

Table 14: RESTs cross-validated results with context

| Total Accuracy | Macro Precision | Macro Recall | Macro F-Measure | Classification Time (ms) |
|---|---|---|---|---|
| 29.60 | 29.67 | 29.60 | 29.12 | 3807.8 |

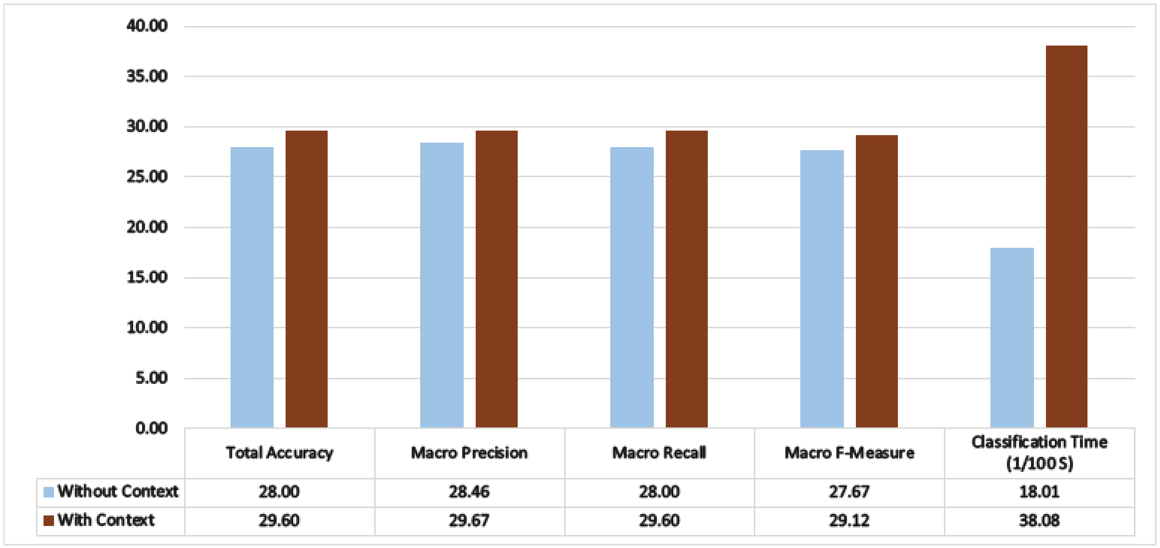
The performance is lower in comparison with WSDL files due to the lack of common structure and high variability of these pages that describe RESTful services in different structures and terminologies and not in a structured format specific to describing Web Services like WSDL files. However, the performance is still significantly higher than the random baseline would have been. Unlike WSDL files, we could not find any related work for classification of REST descriptions in the literature in order to compare with. As far as we know, this work is the initial step towards the classification of REST descriptions. We discuss in Section 6.2.4 how the performance of classification of REST descriptions could be improved.

In order to show the effect of contextual information, as discussed in Section 4.4, we perform another 10-fold cross-validation on the same configuration without considering any contextual information and training on the *Plain* files and testing on the *Plain* files. Table 15 depicts the cross-validated results including the evaluation measures without considering any contextual information. The class-wise results are listed in Appendix B.3.

Table 15: RESTs cross-validated results without context

| **Total Accuracy** | **Macro Precision** | **Macro Recall** | **Macro F-Measure** | **Classification Time (ms)** |
|---|---|---|---|---|
| 28.00 | 28.46 | 28.00 | 27.67 | 1801.4 |

Figure 15 illustrates the effect of adding contextual information to the the REST *tags-filtered* files on total accuracy, macro precision, macro recall, macro F-Measure, and classification time.



| | Total Accuracy | Macro Precision | Macro Recall | Macro F-Measure | Classification Time (1/100 S) |
|---|---|---|---|---|---|
| 🟦 Without Context | 28.00 | 28.46 | 28.00 | 27.67 | 18.01 |
| 🟫 With Context | 29.60 | 29.67 | 29.60 | 29.12 | 38.08 |

Figure 16: Effect of context on classification of REST tags-filtered files

The results depict that contextual information improves the performance of the classification even though it is increasing the classification time due to increase in the file sizes because of the context which is added to them. However, for REST tags-filtered files it is not improving the accuracy as much as for WSDL files due to the nature of these descriptions, which are not defined in a structured format specific to describing Web Services. In other words, because they contain phrases which are more similar to the contextual information phrases, context is not adding much discriminant features to the REST description files.

As a result, we use this configuration to perform the final classification of all REST files. We use the same training result which is the result of training on *Plain + Context* REST *tags-filtered* files and the *Plain* REST *tags-filtered* files for the testing set to be classified.

Finally, we use the best configuration (using contextual information) in order to perform the final classification of all REST description files which class is unknown. Table 16 depicts the number of instances, which were classified inside of each class. Similar to the WSDL files as discussed in Section 6.2.1, these results are based on the performance of the current classification tool. Currently, the full complete classification results cannot be verified because the actual classes are not known. In order to validate our complete data set, all the instances need be classified by human contribution by using approaches such as crowd-sourcing.

Table 16: Final REST classification results

| **Category Name** | **Number of Instances** |
|---|---|
| **Weather** | 6414 |
| **Social** | 6323 |
| **Tourism** | 5560 |
| **Financial** | 6047 |
| **Entertainment** | 6212 |

### 6.2.4 Evaluation

As discussed in Section 3.1.3, WADL descriptions are not popular on the Web and REST services are not widely described using WADL descriptions. As a result, as discussed in Section 4.3, because of the low number of WADL files in the repository, we were not able to find the same number of samples for the classes in comparison with other sample types. The scale of the samples are not as much as the WSDL and REST samples and we discard the 10-fold cross-validation for these files. However, for the sake of completeness we performed a 2-fold cross-validation (swapping training and testing set and averaging the results) and we compared the results with the context and without considering any contextual information. Despite having to rely on limited data sets, our results show that the use contextual information does increase the effectiveness of WADL classification.

As the results depict, the accuracy is generally lower for REST HTML files in comparison with WSDL files because they have more noise, e.g., *JavaScript* and *markup* code, and natural language segments. However, after filtering the tags and cleaning-up these files and storing the raw text inside in a separate text file, the accuracy increased in general and it helped the classification accuracy. Although, the performance for them is still generally lower in contrast with WSDL files due to the nature of these descriptions which are not defined in a structure format specific to describing Web Services and as a result, have high variability due to using different structures and terminologies embedded in the HTML documents. However, the resulting performance is still higher than the random baseline after 10-fold crossvalidation. Unlike WSDL files we could not find any related work for classification of REST descriptions in the literature in order to compare with. As far as we know, this work is the initial step towards the classification of REST descriptions. The performance of the classification for these files can be improved using an approach to extract the most prominent features specific to these files before performing the classification. One way to achieve such goal is to extract all resource URIs using a regular expression extraction process. However, because the URIs are also defined in different structures, formats, and shortcuts and have variability in the files, it requires significantly more experimentation to be done at the semantic-level processing. On the other hand, as the goal of this thesis is not to find the best classification tool for all file types, improving the classification performance is postponed to the future work.

*No Clustering* option, which was mentioned in Section 4.1.4, is found as the best clustering option in the best configurations of MARFCAT for all of the three types of service descriptions. This option disables clustering the training and testing sets' individual class's feature vectors in MARF, i.e., it uses all of the feature vectors of the instances which we passed for a specific class in the training set and calculates their distance to all of the feature vectors of the instances which we passed for a specific class in the testing set instead of using only one feature vector (mean or median). As a result, the space and the time complexity increases. However, because our priority in finding the best configuration is the highest accuracy, we chose this option.

As the results depict, the algorithm combinations which are found as the best combinations in the best configurations of MARFCAT vary throughout the different types of service descriptions. The reason is that this experiment is data-driven and the results is based on the input data. As a result, because the structure and nature of each of these types is different and also due to the manual choosing of training and testing sets for one type regardless of the other types, the aforementioned algorithm combinations vary.

The effect of adding contextual information to the WSDL files, and the REST *tags-filtered* files is illustrated in Figure 15, and Figure 16 respectively. The results depicts that contextual information is improving the performance of the classification for both cases even though it is increasing the classification time due to increase in the file sizes because of the context which is added to them. The context has less effect on the precision of REST *tags-filtered* files in comparison with WSDL files due to the nature of these descriptions which are not defined in a structure format specific to describing Web Services.

In other words, because they contain phrases which are more similar to the contextual information phrases, context is not adding much discriminant features to these files.

In order to perform the final classification of all description files which class is unknown, we used the best configuration (using contextual information) which we could find in the scope of this thesis for each service description type. As a result, these results are based on the performance of the current classification tool. Currently, the results cannot be verified because the actual classes are not known. In order to validate, all the instances need be classified by human contribution by using approaches such as crowd-sourcing.

## 6.3 Summary

In this chapter we measured and illustrated the results of Service Collecting in Section 6.1 including the number of Web Service descriptions and their providers which we found and stored in the repository in total and based on each source. We were able to find a considerable number of URLs which are potentially service descriptions and also stored all three main types of service descriptions. In literature as mentioned in Section 2.3, the researchers focused only on WSDL files. The most number of WSDL URL addresses which was found is 21,358 and from them the most number of WSDL files obtained is 16,514 WSDL files. However, as the results depicts in this chapter, the known repositories are outdated and only around half of service description URLs in them are available. In comparison, in this effort we found 72,454 unique service description URLs including 39,288 WSDL URLs, 1,830 WADL URLs, and 31,336 HTML page URLs describing RESTful services. From these URLs we stored 48,161 availanle service description files including 16,096 WSDL descriptions, 450 WADL descriptions, and 31,615 HTML files describing RESTful services. As a result, we validated our hypothesis 1 which is defined in the beginning of this chapter.

In Section 6.2 we measured and illustrated the results of Service Classification including the resulting accuracies of 72 different cases based on the clustering options and adding contextual information, cross-validated results for the best cases, the effect of adding contextual information to the samples on the classification. In addition, for WSDL files we compared the same evaluation measures with the literature in order to give an insight on how close our classification process is to the literature. Unlike WSDL files we could not find any related work for classification of REST descriptions in the literature in order to compare. As far as we know, this work is the initial step towards the classification of REST descriptions.

We found and presented the best configuration of MARFCAT (best algorithm combination + clustering option) which will result in the highest precision for each type of service description. In addition, we added the contextual information to the classification and showed that it improves the performance of the classification and validated our hypothesis 2 which is defined in the beginning of this chapter.

In the next chapter we will conclude our work and discuss the limitations and the future work.

# Chapter 7

# Conclusion and Future Work

In the scope of this thesis we have developed methodologies to first collect service descriptions and their contextual information and to build a comprehensive repository of them. Subsequently, we surveyed a wide range of Machine Learning and Signal Processing algorithms and techniques in order to find the highest precision achievable in the scope of this thesis for the classification of service descriptions. In this process we exploited the contextual information and illustrated its effect on the Service Classification. At the end, we classified all service descriptions by practicing the best set of algorithms and options which we found. In the following we summarize the contributions and achievements of this thesis and provide some insights to potential future work in the addressed area.

## 7.1 Service Collecting

We discussed the importance of a repository of Web Service descriptions in Chapter 1 from different perspectives, particularly inside of a Broker and a vertical search engine. As discussed in Section 2.3, in the literature, researchers focused on finding WSDL files only. In most of the efforts, the researches only stored the snapshots of WSDL files and did not store URL addresses which point to them. As a result, it is not possible to evaluate these files to determine whether they are still available. The highest number of WSDL URL addresses which was found in the literature is 21,358 and from them the highest number of WSDL files obtained is 16,514. However, as discussed in Section 1.3, the current repositories available are not usable in real-life situation as, according to our results in Section 6.1, only half of service descriptions inside of known repositories are still available.

In Chapter 3 we first exploited methods to harvest three main kinds of Web Service descriptions (*WSDL*, *WADL*, and Web pages describing *RESTful* services) and their context from the World Wide Web with respect to updatability of the design from three different sources. We extracted the data from previous known repositories, we queried search engines and we used Web Crawlers to find service descriptions. We found 72,454 unique service description URLs including 39,288 WSDL URLs, 1,830 WADL URLs, and 31,336 HTML page URLs describing RESTful services.

From these URLs we stored 48,161 actual service description files including 16,096 WSDL descriptions, 450 WADL descriptions, and 31,615 HTML files describing RESTful services. In addition, we extracted 18,494 provider domain URLs from which the Web Service descriptions had been found in order to supply the seeds for Web crawling as discussed in Section 3.2.1. These results exceed, by far, any other similar solution for service collection.

## Limitations and Future Work

We could not find any repositories which lists WADL descriptions and we only used search engines and Web crawling in order to find this kind of descriptions. However, as discussed in Section 3.1.3, REST services are not widely described using WADL in the Web and as a result, the amount of WADLs in comparison with other type of service descriptions which we could find, is much lower.

As discussed in Section 3.1.2, RESTful services do not follow any strict structured description rules and they are described in different formats and most of the time, in different Web pages by different service providers. As a result, RESTful services are much harder to detect and validate hence, using Web Crawlers to find Web pages describing REST services and also validating these pages which we could find from other sources (known repositories and search engines) were not discussed in this thesis and is postponed to the future work.

## 7.2 Service Classification

We argued in Chapter 1 that the classification of high-level functionality of Web Services into pre-defined set of classes, i.e, the highest level of abstraction of the kind of service they perform, can assist and improve the task of compatibility assessment when composing them in Brokerage. In addition, we discussed that it facilitates the task of Service Discovery as well as suggesting Web Services to map to specific service requests. As discussed in Section 1.3, there is currently no consensus on structured support for specifying the abstract class to which the service belongs.

As discussed in Chapter 4, we built on the considerable amount of research that has been carried out on the topic of automatic classification of text documents. We leveraged from combining Machine Learning and Signal Processing techniques and employed contextual information to classify Web Service descriptions. We first surveyed 864 combinations of algorithms and techniques for each of 72 different cases defined based on the sample types, clustering options, and the contextual information (Section 4.4) in order to find the best combination(s) of algorithms and options from the viewpoint of time, accuracy, precision and recall achievable in the scope of this thesis. In this process we analyzed each service description type separately and found the best combination for each of them.

After finding the best configuration, we performed two 10-fold cross-validations; one with the best case without considering any contextual information and one with the context added to the files. We compared the evaluation measures (total accuracy, macro recall and precision, macro F-Measure, and the classification time) and showed that contextual information always improves the performance of the classification. In addition, we compared the performance of our classification for WSDL files and showed that our classification performance is very close to the best combination from the literature without any customization on the preprocessing, feature extraction, and classification based on the WSDL files. At the end, we classified 48,161 service descriptions with the best combinations.

## Limitations and Future Work

As discussed in Section 7.1, we could not find as many WADL files as we could for WSDL and REST descriptions. Consequently, as discussed in Section 4.3, we could not find the required number of samples for WADL files (i.e. 100 for each class). The scale of the samples for these files were not as much as the WSDL and REST samples and we had to discard the 10-fold cross-validation for these files. However, for the sake of completeness we performed a 2-fold cross-validation (swapping training and testing set and averaging the results) and we compared the results with the context and without considering any contextual information. Even with these limitations, we found that using contextual information did improve the quality of classification.

Due to the variety of the source of the data in the repository, the terminology and the quality of the contextual information in the repository may vary through all of the instances. Also as mentioned in Chapter 1, user tags which are part of the contextual information, may not be accurate. As a result, before the classification process, the context needs to be analyzed and purified. One way of achieving this goal is to measure the relevance of the contextual information, i.e., determine whether and how the context is related to the description. In [27] the authors discuss the modeling and exploiting tag relevance which can be employed to validate the context. However, it is out of the scope of this thesis and we use the raw contextual information as we found.

We used the open-source MARF framework and its MARFCAT application as our investigation platform. MARF has a pipeline for character-level NLP processing which we also surveyed for all service description types to find the best algorithm combinations. However, because this pipeline is currently only working on character-level and not lexeme-level and due to low precision, we discarded this pipeline and did not present the results and methodology. In the future work NLP lexeme-level processing can be added to MARF and surveyed to find best algorithm combinations in that area of research.

The performance of the classification for the REST descriptions is generally lower in contrast with WSDL files due to the nature of these descriptions which are not defined in a structured format specific to describing Web Services and as a result, have high variability due to using different structures and terminologies. However, the resulting classification performance is still higher than random. Unlike WSDL files we could not find any related work for classification of REST descriptions in the literature in order to compare. As far as we know, this work is the initial step towards the classification of REST descriptions. The performance of the classification for these files can be improved using an approach to extract the most prominent features specific to these files before performing the classification. One way to achieve such goal is to extract all resource URIs using a pattern recognition process. However, because the URIs are also defined in different structures and have variability in the files, it needs more research to be done on the semantic-level processing. On the other hand, as the goal of this thesis is not to find the best classification tool for all file types, improving the classification performance of REST descriptions is postponed to the future work.

On another perspective, Web Services are being published every day and new services will be found and added to the repository. Due to the changing world of Web Services, the specific classifiers used arenot applicable to all cases in the real world because the best configuration for each sample type was determined according to the specific data set available in the repository. In order to deal with a changing data set, a mechanism to measure the variation of the new incoming services with regards to the existing ones can be applied to determine a threshold above which the training needs to be re-done, or a new configuration for the classifier may need to be found that is adapted to the evolution of the data set. This is also out of the scope of this thesis and can be addressed in the future work.

# References

[1] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar, "Towards recovering the broken SOA triangle: A software engineering perspective," in *2Nd International Workshop on Service Oriented Software Engineering: In Conjunction with the 6th ESEC/FSE Joint Meeting*, IW-SOSWE'07, (Dubrovnik, Croatia), pp. 22–28, ACM, 2007.

[2] T. Laleh, S. A. Mokhov, J. Paquet, and Y. Yan, "Context-aware cloud service brokerage: A solution to the problem of data integration among SaaS providers," in *Proceedings of C3S2E'15*, July 2015. To appear.

[3] E. Badidi and M. El Koutbi, "A framework for automated SLAs management in service-oriented environments," in *International Conference on Multimedia Computing and Systems (ICMCS)*, pp. 601–606, IEEE, April 2014.

[4] S. A. Mokhov, "Study of best algorithm combinations for speech processing tasks in machine learning using median vs. mean clusters in MARF," in *Proceedings of C3S2E'08* (B. C. Desai, ed.), (Montreal, Quebec, Canada), pp. 29–43, ACM, May 2008.

[5] S. Tilkov, "A brief introduction to REST." [online], Accessed March, 2015. http://www.infoq.com/articles/rest-introduction.

[6] A. Bennaceur, V. Issarny, R. Johansson, A. Moschitti, D. Sykes, and R. Spalazzese, "Machine learning for automatic classification of web service interface descriptions," in *Leveraging Applications of Formal Methods, Verification, and Validation* (R. Hahnle, J. Knoop, T. Margaria, D. Schreiner,

and B. Steffen, eds.), Communications in Computer and Information Science, pp. 220–231, Springer Berlin Heidelberg, 2012.

[7] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, and The World Wide Web Consortium (W3C), “Web Services Description Language (WSDL) 1.1.” [online], Accessed November, 2014, Mar. 2001. http://www.w3.org/TR/wsdl.

[8] The World Wide Web Consortium (W3C), “Web Services Description Language (WSDL).” [online], Accessed November, 2014. http://www.w3.org/TR/2003/WD-wsdl20-20031110/.

[9] World Wide Web Consortium (W3C), “Web application description language.” [online] http://www.w3.org/Submission/wadl/. Accessed January, 2015.

[10] R. Perrey and M. Lycett, “Service-oriented architecture,” in *Proceedings 2003 Symposium on Applications and the Internet Workshops (SAINT 2003 Workshops)*, (Orlando, Florida, USA), pp. 116–119, IEEE, Jan. 2003.

[11] M. Endrei, J. Ang, A. Arsanjani, *et al.*, *Patterns: Service-Oriented Architecture and Web Services*. IBM, 2004. IBM Red Book; online at http://www.redbooks.ibm.com/abstracts/sg246303.html.

[12] M. P. Singh and M. N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*. West Sussex, England: John Wiley & Sons, Ltd, 2005.

[13] S. Kambhampaty, *Service-Oriented Architecture for Enterprise Applications*. John Wiley & Sons, 2008.

[14] Y. Chen and W.-T. Tsai, *Service-Oriented Computing and Web Data Management: From Principles to Development*. Kendall/Hunt Publishing Company, first ed., 2008. ISBN: 978-0-7575-7747-5. Online at http://www.public.asu.edu/~ychen10/book/DSOSD.pdf.

[15] Y. Wei and M. B. Blake, “Service-oriented computing and cloud computing: Challenges and opportunities,” *IEEE Internet Computing*, vol. 14, no. 6, pp. 72–75, 2010.

[16] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: service-oriented architecture and design strategies*. John Wiley & Sons, 2012.

[17] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing and verification in service-oriented architecture: a survey," *Software Testing, Verification and Reliability*, vol. 23, no. 4, pp. 261–313, 2013.

[18] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, *et al.*, "UDDI version 3.0," *Published specification, OASIS*, vol. 5, pp. 16–18, 2002.

[19] S. M. Kim and M.-C. Rosu, "A survey of public web services," in *E-commerce and web technologies*, pp. 96–105, Springer, 2004.

[20] N. Steinmetz, "Web service location," Master's thesis, University of Innsbruck, Semantic Technology Institute (STI) Innsbruck, Aug. 2010. http://www.nathaliesteinmetz.net/master_thesis_nathalie_steinmetz.pdf.

[21] D. Bachlechner, K. Siorpaes, D. Fensel, and I. Toma, "Web service discovery-a reality check," in *3rd European Semantic Web Conference*, vol. 308, (Budva, Montenegro), 2006.

[22] H. Lausen and T. Haselwanter, "Finding web services," in *1st European Semantic Technology Conference*, vol. 2007, (Vienna, Austria), Citeseer, 2007.

[23] H. Lausen and N. Steinmetz, "Survey of current means to discover web services," Tech. Rep. STI TR 2008-08-08, Semantic Technology Institute (STI), University of Innsbruck, Innsbruck, Austria, August 2008.

[24] K. Curran and J. Mc Glinchey, "Vertical search engines," *ITB Journal*, vol. 16, no. 3, pp. 22–26, 2007.

[25] G. Almpanidis, C. Kotropoulos, and I. Pitas, "Combining text and link analysis for focused crawling-an application for vertical search engines," *Information Systems*, vol. 32, no. 6, pp. 886–908, 2007.

[26] A. Bennaceur, G. S. Blair, F. Chauvel, N. Georgantas, P. Grace, V. Issarny, V. Nundloll, M. Paolucci, R. Saadi, and D. Sykes, “Intermediate CONNECT Architecture,” Research Report inria-00584911, HAL - INRIA, Feb. 2011. https://hal.inria.fr/inria-00584911/file/Connect_WP1_D12.pdf.

[27] L. Chen, J. Wu, Z. Zheng, M. R. Lyu, and Z. Wu, “Modeling and exploiting tag relevance for web service mining,” *Knowledge and Information Systems*, vol. 39, no. 1, pp. 153–173, 2014. http://dx.doi.org/10.1007/s10115-013-0703-1.

[28] L. S. Kennedy, S.-F. Chang, and I. V. Kozintsev, “To search or to label?: predicting the performance of search-based automatic image classifiers,” in *Proceedings of the 8th ACM international workshop on Multimedia information retrieval*, pp. 249–258, ACM, 2006.

[29] J. Wu, L. Chen, Z. Zheng, M. R. Lyu, and Z. Wu, “Clustering web services to facilitate service discovery,” *Knowledge and Information Systems*, vol. 38, no. 1, pp. 207–229, 2014. http://dx.doi.org/10.1007/s10115-013-0623-0.

[30] J. Scicluna, C. Blank, N. Steinmetz, and E. Simperl, “Crowd sourcing web service annotations,” in *AAAI Spring Symposium: Intelligent Web Services Meet Social Computing*, Association for the Advancement of Artificial Intelligence (AAAI), 2012. http://www.aaai.org/ocs/index.php/SSS/SSS12/paper/viewFile/4316/4664.

[31] T. Erl, *SOA: principles of service design*, vol. 1. Prentice Hall Upper Saddle River, 2008.

[32] Q. Duan, Y. Yan, and A. V. Vasilakos, “A survey on service-oriented network virtualization toward convergence of networking and cloud computing,” *IEEE Transactions on Network and Service Management*, vol. 9, pp. 373–392, Dec. 2012.

[33] Open Group, "The Open Group: Leading the development of open, vendor-neutral IT standards and certifications." [online], Accessed November, 2014. http://www.opengroup.org/aboutus.

[34] Open Group, "Service oriented architecture." [online], Accessed November, 2014. http://www.opengroup.org/soa/source-book/soa/soa.htm.

[35] OASIS, "OASIS SOA reference model TC." [online], Accessed November, 2014. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

[36] NIST, "About NIST, the national institute of standards and technology of USA." [online], Accessed November, 2014. http://www.nist.gov/public_affairs/nandyou.cfm.

[37] P. Mell and T. Grance, "The NIST definition of cloud computing," Tech. Rep. NIST SP 500-145, NIST, Sept. 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[38] W.-T. Tsai, X. Sun, and J. Balasooriya, "Service-oriented cloud computing architecture," in *Seventh International Conference on Information Technology: New Generations (ITNG)*, pp. 684–689, IEEE, 2010.

[39] S. Sundareswaran, A. Squicciarini, and D. Lin, "A brokerage-based approach for cloud service selection," in *5th International Conference on Cloud Computing (CLOUD)*, pp. 558–565, IEEE, June 2012.

[40] S. G. Grivas, T. U. Kumar, and H. Wache, "Cloud broker: Bringing intelligence into the cloud," in *3rd International Conference on Cloud Computing (CLOUD)*, (Las Vegas, NV, USA), pp. 544–545, IEEE, 2010.

[41] MarketsandMarkets, "Dallas, TX Market Research Company and Consulting Firm." [online], Accessed July, 2015. http://www.marketsandmarkets.com/AboutUs-8.html.

[42] MarketsandMarkets, "Cloud Services Brokerage Market by Types - Global Forecast to 2020." [online], Accessed July, 2015. http://www.marketsandmarkets.com/Market-Reports/cloud-brokerage-market-771.html.

[43] E. Badidi, "A framework for brokered service level agreements in SOA environments," in *7th International Conference on Next Generation Web Services Practices (NWeSP)*, pp. 37–42, IEEE, Oct 2011.

[44] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, "Web services on demand: WSLA-driven automated management," *IBM systems journal*, vol. 43, no. 1, pp. 136–158, 2004.

[45] D. Martin, M. Burstein, J. Hobbs, and The World Wide Web Consortium (W3C), "OWL-S: Semantic Markup for Web Services." [online], Accessed February, 2015. http://www.w3.org/Submission/OWL-S/.

[46] The World Wide Web Consortium (W3C), "Web Service Modeling Ontology (WSMO)." [online], Accessed February, 2015. http://www.w3.org/Submission/WSMO/.

[47] U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel, "Automatic location of services," in *The Semantic Web: Research and Applications*, pp. 1–16, Springer, 2005.

[48] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *The Semantic Web ISWC 2002*, pp. 333–347, Springer, 2002.

[49] U. Keller, H. Lausen, and M. Stollberg, *On the semantics of functional descriptions of web services*. Springer, 2006.

[50] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, "Web service modeling ontology," *Applied ontology*, vol. 1, no. 1, pp. 77–106, 2005.

[51] M. Klusch, B. Fries, and K. Sycara, "Automated semantic web service discovery with owls-mx," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, (Hakodate, Japan), pp. 915–922, ACM, 2006.

[52] N. Srinivasan, M. Paolucci, and K. Sycara, "Semantic web service discovery in the owl-s ide," in *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, vol. 6, (Honolulu, Hawaii), pp. 109b–109b, IEEE, 2006.

[53] J. Kang and K. M. Sim, "Towards agents and ontology for cloud service discovery," in *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, (Beijing, China), pp. 483–490, IEEE, Oct. 2011.

[54] A. V. Paliwal, B. Shafiq, J. Vaidya, H. Xiong, and N. Adam, "Semantics-based automated service discovery," *Services Computing, IEEE Transactions on*, vol. 5, no. 2, pp. 260–275, 2012.

[55] L. D. Ngan and R. Kanagasabai, "Semantic web service discovery: state-of-the-art and research challenges," *Personal and ubiquitous computing*, vol. 17, no. 8, pp. 1741–1752, 2013.

[56] G. Cassar, P. Barnaghi, and K. Moessner, "Probabilistic matchmaking methods for automated service discovery," *IEEE Transactions on Services Computing*, vol. 7, no. 4, pp. 654–666, 2014.

[57] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for web services," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, (Toronto, Canada), pp. 372–383, VLDB Endowment, 2004.

[58] R. Nayak, “Data mining in web services discovery and monitoring,” *Web Services Research for Emerging Applications: Discoveries and Trends*, p. 270, 2010.

[59] F. Liu, Y. Shi, J. Yu, T. Wang, and J. Wu, “Measuring similarity of web services based on wsdl,” in *IEEE International Conference on Web Services (ICWS)*, (Miami, FL, USA), pp. 155–162, IEEE, 2010.

[60] E. Al-Masri and Q. H. Mahmoud, “WSCE: A crawler engine for large-scale discovery of web services,” in *IEEE International Conference on Web Services (ICWS)*, (Salt Lake City, UT, USA), pp. 1104–1111, IEEE, 2007.

[61] E. Al-Masri and Q. H. Mahmoud, “Investigating Web Services on the World Wide Web,” in *Proceedings of the 17th International Conference on World Wide Web*, WWW 08, (Beijing, China), pp. 795–804, ACM, 2008.

[62] D. Dehua, “Deep web services crawler,” *Dresden University of Technology*, 2010.

[63] N. Steinmetz, H. Lausen, and M. Brunner, “Web service search on large scale,” in *Service-Oriented Computing* (L. Baresi, C.-H. Chi, and J. Suzuki, eds.), vol. 5900 of *Lecture Notes in Computer Science*, pp. 437–444, Stockholm, Sweden: Springer Berlin Heidelberg, 2009.

[64] N. Steinmetz and H. Lausen, “Ontology-based feature aggregation for multi-valued ranking,” in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, vol. 6275 of *Lecture Notes in Computer Science*, pp. 258–268, Springer Berlin Heidelberg, 2010.

[65] M. AbuJarour, F. Naumann, and M. Craculeac, “Collecting, annotating, and classifying public web services,” in *On the Move to Meaningful Internet Systems: OTM 2010* (R. Meersman, T. Dillon, and P. Herrero, eds.), vol. 6426 of *Lecture Notes in Computer Science*, pp. 256–272, Springer Berlin Heidelberg, 2010.

[66] T. Pold, "Heuristics for crawling wsdl descriptions of web service interfaces - the heritrix case," bachelor's thesis, University of Tartu, Faculty of Mathematics and Computer Science, 2012.

[67] Z. Zheng, Y. Zhang, and M. R. Lyu, "Distributed QoS evaluation for real-world web services," in *2010 IEEE International Conference on Web Services (ICWS)*, (Miami, FL, USA), pp. 83–90, IEEE, July 2010.

[68] Y. Zhang, Z. Zheng, and M. R. Lyu, "Exploring latent features for memory-based qos prediction in cloud computing," in *2011 30th IEEE International Symposium on Reliable Distributed Systems*, (Madrid, Spain), pp. 1–10, IEEE, Oct. 2011.

[69] L. Chen, L. Hu, Z. Zheng, J. Wu, J. Yin, Y. Li, and S. Deng, "WTCluster: Utilizing tags for web services clustering," in *Service-Oriented Computing* (G. Kappel, Z. Maamar, and H. Motahari-Nezhad, eds.), vol. 7084 of *Lecture Notes in Computer Science*, pp. 204–218, Berlin, Germany: Springer Berlin, 2011. http://dx.doi.org/10.1007/978-3-642-25535-9_14.

[70] J. Wu, L. Chen, Y. Xie, and Z. Zheng, "Titan: a system for effective web service discovery," in *Proceedings of the 21st international conference companion on World Wide Web*, (Lyon, France), pp. 441–444, ACM, International World Wide Web Conference Committee(IW3C2), Apr. 2012. http://dl.acm.org/citation.cfm?id=2188069.

[71] P. Sun and C. Jiang, "Using service clustering to facilitate process-oriented semantic web service discovery," *Chinese Journal of Computers*, vol. 31, no. 8, pp. 1340–1353, 2008.

[72] C. Pop, V. Chifu, I. Salomie, M. Dinsoreanu, T. David, and V. Acretoaie, "Semantic web service clustering for efficient discovery using an ant-based method," in *Intelligent Distributed Computing IV* (M. Essaaidi, M. Malgeri, and C. Badica, eds.), vol. 315 of *Studies in Computational Intelligence*, pp. 23–33, Springer Berlin Heidelberg, 2010.

[73] S. Dasgupta, S. Bhat, and Y. Lee, "Taxonomic clustering of web service for efficient discovery," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, (Toronto, ON, Canada), pp. 1617–1620, ACM, 2010.

[74] L. li Xie, F. zan Chen, and J.-S. Kou, "Ontology-based semantic web services clustering," in *2011 IEEE 18Th International Conference on Industrial Engineering and Engineering Management (IE EM)*, vol. Part 3, (Changchun, China), pp. 2075–2079, IEEE, Sept. 2011.

[75] B. Kumara, I. Paik, and W. Chen, "Web-service clustering with a hybrid of ontology learning and information-retrieval-based term similarity," in *2013 IEEE 20th International Conference on Web Services (ICWS)*, (Santa Clara, CA, USA), pp. 340–347, IEEE, June 2013.

[76] W. Liu and W. Wong, "Discovering homogenous service communities through web service clustering," in *Service-Oriented Computing: Agents, Semantics, and Engineering* (R. Kowalczyk, M. Huhns, M. Klusch, Z. Maamar, and Q. Vo, eds.), vol. 5006 of *Lecture Notes in Computer Science*, pp. 69–82, Springer Berlin Heidelberg, 2008.

[77] W. Liu and W. Wong, "Web service clustering using text mining techniques," *International Journal of Agent-Oriented Software Engineering*, vol. 3, no. 1, pp. 6–26, 2009.

[78] K. Elgazzar, A. E. Hassan, and P. Martin, "Clustering wsdl documents to bootstrap the discovery of web services," in *Web Services (ICWS), 2010 IEEE International Conference on*, (Miami, FL, USA), pp. 147–154, IEEE, 2010.

[79] U. Chukmol, A.-N. Benharkat, and Y. Amghar, "Bringing socialized semantics into web services based on user-centric collaborative tagging and usage experience," in *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, (Jeju Island, South Korea), pp. 450–455, IEEE, Dec. 2011.

[80] Z. Azmeh, J.-R. Falleri, M. Huchard, and C. Tibermacine, "Automatic web service tagging using machine learning and wordnet synsets," in *Web Information Systems and Technologies* (J. Filipe and J. Cordeiro, eds.), vol. 75 of *Lecture Notes in Business Information Processing*, pp. 46–59, Springer Berlin Heidelberg, 2011.

[81] I. Katakis, G. Pallis, M. D. Dikaiakos, and O. Onoufriou, "Automated tagging for the retrieval of software resources in grid and cloud infrastructures," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, (Ottawa, ON, Canada), pp. 628–635, IEEE, 2012.

[82] S. Lawrence and C. L. Giles, "Accessibility of information on the web," *Nature*, vol. 400, no. 6740, pp. 107–107, 1999.

[83] J. Devine and F. Egger Sider, "Beyond Google the invisible web in the academic library," *The Journal of Academic Librarianship*, vol. 30, no. 4, pp. 265–269, 2004.

[84] N. Hamilton, "The mechanics of a deep net metasearch engine," in *Proceedings of the 12th International World Wide Web Conference*, (Budapest, Hungary), Citeseer, ACM, 2003.

[85] S. Raghavan and H. Garcia-Molina, "Crawling the hidden web," Technical Report 2000-36, Stanford InfoLab, 2000. http://ilpubs.stanford.edu:8090/456/.

[86] M. K. Bergman, "White paper: the deep web: surfacing hidden value," *Journal of electronic publishing*, vol. 7, no. 1, 2001.

[87] A. Gulli and A. Signorini, "The indexable web is more than 11.5 billion pages," in *Special interest tracks and posters of the 14th international conference on World Wide Web*, (Chiba, Japan), pp. 902–903, ACM, ACM, 2005.

[88] ILK Research Group and Tilburg University, "The size of the World Wide Web (The Internet)." [online], Accessed January, 2015. http://www.worldwidewebsize.com/.

[89] N. M. Josuttis, *SOA in practice: the art of distributed system design*. O'Reilly Media, Inc., 2007.

[90] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, CA, USA, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[91] A. Rodriguez, "Restful web services: The basics," tech. rep., IBM developerWorks, 2008. http://www.ibm.com/developerworks/library/ws-restful/.

[92] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.

[93] World Wide Web Consortium (W3C), "Team comment on the Web Application Description Language submission." [online], Accessed January, 2015. http://www.w3.org/Submission/2009/03/Comment.

[94] B. N. Schilit and M. M. Theimer, "Disseminating active map information to mobile hosts," *Network, IEEE*, vol. 8, no. 5, pp. 22–32, 1994.

[95] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[96] H.-L. Truong and S. Dustdar, "A survey on context-aware web service systems," *International Journal of Web Information Systems*, vol. 5, no. 1, pp. 5–31, 2009.

[97] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: aiding the development of context-enabled applications," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 434–441, ACM, 1999.

[98] D. H. Chau, S. Pandit, S. Wang, and C. Faloutsos, “Parallel crawling for online social networks,” in *Proceedings of the 16th International Conference on World Wide Web*, WWW ’07, (New York, NY, USA), pp. 1283–1284, ACM, 2007.

[99] M. Kobayashi and K. Takeda, “Information retrieval on the web,” *ACM Computing Surveys (CSUR)*, vol. 32, no. 2, pp. 144–173, 2000.

[100] Google, “Crawling and indexing.” [online], Accessed January, 2015. http://www.google.ca/insidesearch/howsearchworks/crawling-indexing.html.

[101] J. Masanes, *Web archiving*. Springer, 2006.

[102] C. Castillo, “Effective web crawling,” in *ACM SIGIR Forum*, vol. 39, pp. 55–56, ACM, 2005.

[103] eBizMBA Inc., “Top 15 most popular search engines february 2015.” [online], Accessed February, 2015. http://www.ebizmba.com/articles/search-engines.

[104] comScore Inc., “Desktop search engine rankings.” [online], Accessed January, 2015. http://www.comscore.com/Insights/Market-Rankings/comScore-Releases-December-2014-US-Desktop-Search-Engine-Rankings.

[105] R. Basili and A. Moschitti, “Automatic text categorization,” *From information retrieval to support vector learning*, 2005.

[106] B. Mitchell and R. Gaizauskas, *A Comparison of Machine Learning Algorithms for Prepositional Phrase Attachment*. University of Sheffield, 2002.

[107] The MARF Research and Development Group, “The Modular Audio Recognition Framework and its Applications.” [online], 2002–2014. http://marf.sf.net and http://arxiv.org/abs/0905.1235, last viewed May 2015.

[108] S. A. Mokhov, J. Paquet, M. Debbabi, and Y. Sun, “MARFCAT: Transitioning to binary and larger data sets of SATE IV.” [online], May 2012–2014. Online at http://arxiv.org/abs/1207.3718.

[109] V. Okun, A. Delaitre, P. E. Black, and NIST SAMATE, "Static Analysis Tool Exposition (SATE) 2010." [online], 2010. See http://samate.nist.gov/SATE2010Workshop.html.

[110] V. Okun, A. Delaitre, P. E. Black, and NIST SAMATE, "Static Analysis Tool Exposition (SATE) IV." [online], Mar. 2012. See http://samate.nist.gov/SATE.html.

[111] S. A. Mokhov, "Evolution of MARF and its NLP framework," in *Proceedings of C3S2E'10*, pp. 118–122, ACM, May 2010.

[112] S. A. Mokhov, "The use of machine learning with signal- and NLP processing of source code to fingerprint, detect, and classify vulnerabilities and weaknesses with MARFCAT," Tech. Rep. NIST SP 500-283, NIST, Oct. 2011. Report: http://www.nist.gov/manuscript-publication-search.cfm?pub_id=909407, online e-print at http://arxiv.org/abs/1010.2511.

[113] S. A. Mokhov, J. Paquet, and M. Debbabi, "MARFCAT: Fast code analysis for defects and vulnerabilities," in *Proceedings of SWAN'15* (O. Baysal and L. Guerrouj, eds.), pp. 35–38, IEEE, Mar. 2015.

[114] S. A. Mokhov, J. Paquet, and M. Debbabi, "The use of NLP techniques in static code analysis to detect weaknesses and vulnerabilities," in *Proceedings of Canadian Conference on AI'14* (M. Sokolova and P. van Beek, eds.), vol. 8436 of *LNAI*, pp. 326–332, Springer, May 2014. Short paper.

[115] A. Boukhtouta, S. A. Mokhov, N.-E. Lakhdari, M. Debbabi, and J. Paquet, "Network malware classification comparison using DPI and flow packet headers," *Journal of Computer Virology and Hacking Techniques*, pp. 1–32, July 2014–2015.

[116] S. M. Bernsee, "The DFT "à pied": Mastering the Fourier transform in one day." [online], Sept. 1999–2010. http://www.dspdimension.com/admin/dft-a-pied/.

[117] H. Abdi, "Distance," in *Encyclopedia of Measurement and Statistics* (N. J. Salkind, ed.), (Thousand Oaks (CA): Sage), 2007.

[118] P. C. Mahalanobis, "On the generalised distance in statistics," in *Proceedings of the National Institute of Science of India 12*, pp. 49–55, 1936. Online at http://en.wikipedia.org/wiki/Mahalanobis_distance.

[119] D. Mackenzie, P. Eggert, and R. Stallman, "Comparing and merging files." [online], 2002. http://www.gnu.org/software/diffutils/manual/ps/diff.ps.gz.

[120] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 26, no. 2, pp. 147–160, 1950. See also http://en.wikipedia.org/wiki/Hamming_distance.

[121] E. Garcia, "Cosine similarity and term weight tutorial." [online], 2006. http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html.

[122] A. Kishore, "Similarity measure: Cosine similarity or euclidean distance or both." [online], Feb. 2007. http://semanticvoid.com/blog/2007/02/23/similarity-measure-cosine-similarity-or-euclidean-distance-or-both/.

[123] M. Khalifé, "Examining orthogonal concepts-based micro-classifiers and their correlations with noun-phrase coreference chains," Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2004.

# Appendix A

# Examples

## A.1 WSDL Examples

Listing A.1: WSDL 1.1 Example [7]

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="TradePriceRequest">
              <complexType>
                  <all>
                      <element name="tickerSymbol" type="string"/>
                  </all>
              </complexType>
           </element>
           <element name="TradePrice">
```

```
                <complexType>
                    <all>
                        <element name="price" type="float"/>
                    </all>
                </complexType>
            </element>
        </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
            <input message="tns:GetLastTradePriceInput"/>
            <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="
        tns:StockQuotePortType">
        <soap:binding style="document" transport="http://schemas.
            xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
            <soap:operation soapAction="http://example.com/
                GetLastTradePrice"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
            <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```

Listing A.2: WSDL 2.0 Example [8]

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions">

  <documentation>
    This document describes the GreatH Web service.  Additional
    application-level requirements for use of this service --
    beyond what WSDL 2.0 is able to describe -- are available
    at http://greath.example.com/2004/reservation-documentation.html
  </documentation>

  <types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://greath.example.com/2004/schemas/
            resSvc"
        xmlns="http://greath.example.com/2004/schemas/resSvc">

      <xs:element name="checkAvailability" type="tCheckAvailability"
          />
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element  name="checkInDate" type="xs:date"/>
          <xs:element  name="checkOutDate" type="xs:date"/>
          <xs:element  name="roomType" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>

      <xs:element name="checkAvailabilityResponse" type="xs:double"/
          >

      <xs:element name="invalidDataError" type="xs:string"/>

    </xs:schema>
  </types>

  <interface  name = "reservationInterface" >

    <fault name = "invalidDataFault"
            element = "ghns:invalidDataError"/>

    <operation name="opCheckAvailability"
            pattern="http://www.w3.org/ns/wsdl/in-out"
            style="http://www.w3.org/ns/wsdl/style/iri"
            wsdlx:safe = "true">
        <input messageLabel="In"
              element="ghns:checkAvailability" />
        <output messageLabel="Out"
              element="ghns:checkAvailabilityResponse" />
        <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
    </operation>

  </interface>

  <binding name="reservationSOAPBinding"
           interface="tns:reservationInterface"
           type="http://www.w3.org/ns/wsdl/soap"
           wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/
              HTTP/">

    <fault ref="tns:invalidDataFault"
      wsoap:code="soap:Sender"/>

    <operation ref="tns:opCheckAvailability"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

  </binding>

  <service name="reservationService"
        interface="tns:reservationInterface">

     <endpoint name="reservationEndpoint"
               binding="tns:reservationSOAPBinding"
               address ="http://greath.example.com/2004/reservation"
                  />

  </service>

</description>
```

## A.2 WADL Example

Listing A.3: WADL Example [9]

```
<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
```

```
xmlns:tns="urn:yahoo:yn"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:yn="urn:yahoo:yn"
xmlns:ya="urn:yahoo:api"
xmlns="http://wadl.dev.java.net/2009/02">
 <grammars>
   <include
     href="NewsSearchResponse.xsd"/>
   <include
     href="Error.xsd"/>
 </grammars>

 <resources base="http://api.search.yahoo.com/NewsSearchService/V1/
     ">
   <resource path="newsSearch">
     <method name="GET" id="search">
       <request>
         <param name="appid" type="xsd:string"
           style="query" required="true"/>
         <param name="query" type="xsd:string"
           style="query" required="true"/>
         <param name="type" style="query" default="all">
           <option value="all"/>
           <option value="any"/>
           <option value="phrase"/>
         </param>
         <param name="results" style="query" type="xsd:int" default
             ="10"/>
         <param name="start" style="query" type="xsd:int" default="
             1"/>
         <param name="sort" style="query" default="rank">
           <option value="rank"/>
           <option value="date"/>
         </param>
         <param name="language" style="query" type="xsd:string"/>
       </request>
```

```
          <response status="200">
            <representation mediaType="application/xml"
              element="yn:ResultSet"/>
          </response>
          <response status="400">
            <representation mediaType="application/xml"
              element="ya:Error"/>
          </response>
        </method>
      </resource>
    </resources>

</application>
```

## A.3 REST Example

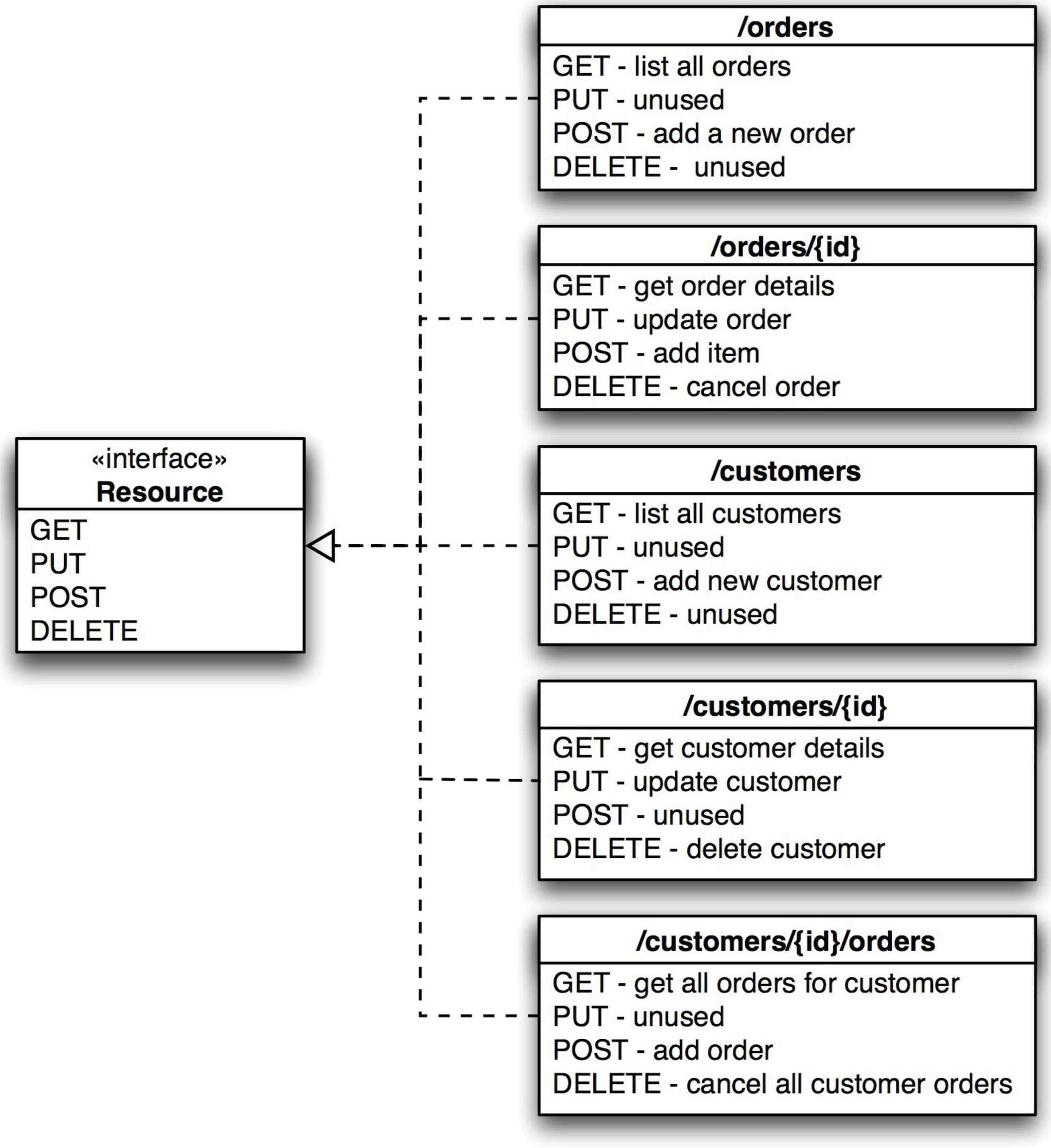


Figure 17: REST Example [5]

# Appendix B

# Classification Results

## B.1 WSDLs Results

Table 17: WSDLs cross-validated class-wise results with context

| Category Name | True Positives | False Negatives | False Positives | Precision | Recall | F-Measure | Classification Time |
|---|---|---|---|---|---|---|---|
| **Weather** | 6.5 | 3.5 | 2.3 | 73.97 | 65.00 | 69.00 | 635.6 |
| **Social** | 5.5 | 4.5 | 3.8 | 59.23 | 55.00 | 56.60 | 636.8 |
| **Tourism** | 6.6 | 3.4 | 5.1 | 57.45 | 66.00 | 60.80 | 564.4 |
| **Financial** | 6.1 | 3.9 | 4.2 | 57.73 | 61.00 | 58.56 | 756.2 |
| **Entertainment** | 4.8 | 5.2 | 5.1 | 49.86 | 48.00 | 48.12 | 839.8 |

Table 18: WSDLs cross-validated class-wise results without context

| Category Name | True Positives | False Negatives | False Positives | Precision | Recall | F-Measure | Classification Time |
|---|---|---|---|---|---|---|---|
| **Weather** | 4.7 | 5.3 | 3.8 | 57.05 | 47.00 | 51.26 | 221.8 |
| **Social** | 5.8 | 4.2 | 4.7 | 57.64 | 58.00 | 56.90 | 293.4 |
| **Tourism** | 6 | 4 | 5.7 | 51.44 | 60.00 | 54.60 | 289.2 |
| **Financial** | 5.9 | 4.1 | 3.4 | 63.68 | 59.00 | 59.88 | 252.8 |
| **Entertainment** | 4.9 | 5.1 | 5.1 | 50.30 | 49.00 | 49.20 | 420.8 |

Table 19: Effect of changing frequency on the accuracy

| | **8kHz** | **16kHz** | **24kHz** | **44kHz** |
|---|---|---|---|---|
| **TrainPlain TestPlain No Clustering** | 64.00 | 64.00 | 64.00 | 64.00 |
| **TrainPlainCtx TestPlain No Clustering** | 76.00 | 76.00 | 76.00 | 76.00 |

## B.2 WADLs Results

We perform a two-fold cross-validation and average the results for the best cases which are presented in Section 6.2.2 in order to reduce variability. Table 20 depicts the average accuracies of the results of the best cases.

Table 20: WADLs best cases cross-validated results

| | **TrainPlainCtx TestPlainCtx** | **TrainPlain TestPlain** |
|---|---|---|
| **No Clustering** | 63.235 | 62.91 |

As the results depicted, the best result for the classification of WADL files is achieved by both training and testing on *Plain + Context* files using the following MARFCAT configuration:

- No Clustering option (discussed in Section 4.1.4)
- `-noise`, `-bandstop`, `-fft`, `-eucl` (discussed in Section 4.1)

## B.3 REST files Results

Table 21: RESTs cross-validated class-wise results with context

| Category Name | True Positives | False Negatives | False Positives | Precision | Recall | F-Measure | Classification Time |
|---|---|---|---|---|---|---|---|
| **Weather** | 3.6 | 6.4 | 7.5 | 32.74 | 36.00 | 34.07 | 1011.2 |
| **Social** | 2.2 | 7.8 | 7.1 | 23.23 | 22.00 | 22.20 | 692.8 |
| **Tourism** | 2.9 | 7.1 | 7.6 | 28.06 | 29.00 | 28.31 | 707.6 |
| **Financial** | 3.4 | 6.6 | 6.3 | 33.99 | 34.00 | 33.20 | 791.2 |
| **Entertainment** | 2.7 | 7.3 | 6.7 | 30.31 | 27.00 | 27.81 | 605.0 |

Table 22: RESTs cross-validated class-wise results without context

| Category Name | True Positives | False Negatives | False Positives | Precision | Recall | F-Measure | Classification Time |
|---|---|---|---|---|---|---|---|
| **Weather** | 3 | 7 | 6.2 | 32.98 | 30.00 | 30.42 | 605.8 |
| **Tourism** | 2.5 | 7.5 | 7.4 | 25.52 | 25.00 | 24.69 | 336.6 |
| **Entertainment** | 3.1 | 6.9 | 8.7 | 26.37 | 31.00 | 28.24 | 268.8 |
| **Financial** | 3.4 | 6.6 | 6.3 | 35.66 | 34.00 | 34.36 | 291.4 |
| **Social** | 2 | 8 | 7.4 | 21.78 | 20.00 | 20.62 | 298.8 |

# Index