



S	
E1	
E2	
For Office Use Only	

Masters Project Final Report
(MCS)
2019

Project Title	REST SERVICE DISCOVERY BASED ON ONTOLOGY MODEL
Student Name	SHANALIE SILVA
Registration No. & Index No.	2017/MCS/076 17440763
Supervisor's Name	DR. D.D. KARUNARATNE

For Office Use ONLY



REST Service Discovery Based on Ontology Model

**A dissertation submitted for the Degree of Master of
Computer Science**

**S. M. SILVA
University of Colombo School of Computing
2019**



Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge, it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: Shanalie Silva

Registration Number: 17440763

Index Number: 2017/MCS/076

Date: 21/6/2020

Signature:

This is to certify that this thesis is based on the work of

Mr./Ms.

under my supervision. The thesis has been prepared according to the format stipulated and is of an acceptable standard.

Certified by:

Supervisor Name:

Signature:

Date:

ABSTRACT

RESTful web services are the new drift that is used for most of the present technical solutions when creating stateless web services. At present world moves towards RESTful web services, the number of available web services is rapidly increasing thereby service discovery can be a challenging task. The present service discovery approaches have deficiencies in semantics, lack of inefficiency, and lower precision rates.

The web service usage can be categorized as 'Provider Developers' and 'Consumer Developers'. Provider Developers develop APIs and publish along with Open API specification. Published web services are consumed by Consumer Developers. Consumer Developers integrate third-party web services in their solutions yet face difficulties in finding the best suitable service for their needs.

Most traditional service discovery approaches depend on semantic specifications like OWL-S. OWL-S is the universal ontology specification to represent RESTful service semantics. This specification was initially created for WSDL and up to the present, there are only a few conversions available for RESTful service to OWL-S. To convert a REST service to a semantic specification, the syntactic of the RESTful service needs to be available. The objective of this **Research Study** is to propose a solution to reducing the present gap between syntactic and semantics of RESTful service.

At present most RESTful service developers incorporate Open API specification formerly known as Swagger to provide syntactically. The main use of the Open API Specification document provides guided documentation for consumers on how to invoke web service. In this document, synthetics are hidden, which can be extremely useful for service discovery, but mostly not used at present. The proposed study aims towards the use of Open API specification to extract syntactic and map syntactic to OWL-S semantic specification. The approach further proposes to provide a keyword-based search engine that can be used to search web services based on IOPE conditions by leveraging the generated semantic specification. As proof of concept for this proposed approach, Lufthansa Swagger API was used to generate semantic specification and service grounding. The search engine was developed to query by keyword and Input/ Output condition based.

ACKNOWLEDGEMENT

I sincerely like to thank everyone who supported me throughout this MCS project. Especially, I would like to thank the Architects at Virtusa for giving me valuable advice in practically implementing this solution.

I would like to express my sincere gratitude to my supervisor Dr. D.D. Karunaratne for providing his invaluable guidance, comments, and suggestions throughout the project.

Thank You
Shanalie Silva
17440763 (2017 | MCS | 076)

TABLE OF CONTENTS

ABSTRACT.....	i
ACKNOWLEDGEMENT	ii
ABBREVIATION.....	v
CHAPTER 1: INTRODUCTION	1
1.1 OVERVIEW.....	1
1.2 PROBLEM STATEMENT	1
1.2 PROBLEM DEFINITION	2
1.4 MOTIVATION.....	3
1.5 COMPUTER SCIENCE PROBLEM	4
1.6 RESEARCH CONTRIBUTION	4
1.6.1 GOAL.....	4
1.6.2 OBJECTIVES OF THE STUDY	5
1.7 SCOPE.....	6
1.8 RESEARCH CONTRIBUTION	6
1.9 CHAPTER STRUCTURE	7
CHAPTER 2: LITERATURE REVIEW	8
2.1 REST.....	8
2.2 ONTOLOGY.....	9
2.3 REPRESENTING SEMANTICS AND DERIVING SYNTACTIC	9
2.4 RESEARCH GAP AND LIMITATIONS IN EXISTING APPROACHES	23
CHAPTER 3: METHODOLOGY.....	23
3.1.PROBLEM ANALYSIS	23
3.2 PROPOSING MODEL/DESIGN	29
CHAPTER 4: EVALUATION PROCESS	44
4.1 EVALUATION SCOPE	44
4.2 EVALUATION DATASET.....	44
4.3 EVALUATION APPROACH.....	45
4.4 EVALUATION RESULTS.....	47
CHAPTER 5: CONCLUSION AND FUTURE WORK.....	49
5.1 CONCLUSION	49
5.2 FUTURE WORK.....	50
REFERENCES.....	51

LIST OF FIGURES

Figure 1: Abstract view currently existing semantic description implementations and approaches	10
Figure 2: An annotated API document HTML.	12
Figure 3: An example of hREST how the tags are showed in class level.....	12
Figure 4: MircoWSMO layered overview.	13
Figure 5: Unifying sawsdl and Microwsmo through wsmo-Lite service.....	14
Figure 6: Top level of the service ontology	16
Figure 7: Select the class and properties of the profile.....	17
Figure 8: OWL syntax to represent IOPE.....	18
Figure 9: Select classes and properties of Service Process	18
Figure 10 OWL-S and WSDL.....	19
Figure 11: The Structure of Concept Operation Ontology Model (COOM).....	21
Figure 12: Hierarchical capability-based matching.....	22
Figure 13: List of verbs supported by HTTP protocol for REST	24
Figure 14: UI of Swagger	26
Figure 15: UI of the Spring doc	26
Figure 16: Code snippet of implemented annotation approach.....	28
Figure 17: High-level Architecture Diagram	30
Figure 18: Example of Java code that is annotated with Swagger annotations.....	31
Figure 19: Example of the above service shown in Swagger UI.	32
Figure 20: How Schema.org ontology gets populated using HTML meta tags in web pages	33
Figure 21: Abstract view of Schema.org ontology.....	33
Figure 22: Protégé view of instances for a service.....	35
Figure 23: Object Properties of Service instance	35
Figure 24: Protégé view of Service Profile subclass for a new service call Operation Flight	36
Figure 25: Protégé view of the agent instance.	36
Figure 26: illustrates how Swagger objects are mapped to OWL-S Process concepts.....	37
Figure 27: Protégé view of Atomic process instance with multiple input parameters.	38
Figure 28: Protégé view of an Input Process instance	38
Figure 29: Protégé view of Output Process instance	39
Figure 30: Protégé view of Grounding instance.....	39
Figure 31 : Illustrates high level of classes, instances in the created OWL-S ontologies.....	40
Figure 32: Search Engine UI that implemented for the prototype of this research	41
Figure 33: Search engine results for keyword “Airport”.	41

Figure 34: SPAQL Query To filter by Output 42
 Figure 35: SPAQL Query To filter by Input..... 43
 Figure 36: F measure on API 48

LIST OF EQUATION

Equation 1: QALD-2 Measurements46

LIST OF TABLES

Table 1: Results of the web service testing using QALD-2 Measurements.....48

ABBREVIATION

OWL	Web Ontology Language
OWL-S	Semantic Markup for Web Services
API	Application Programming Interface
REST Services	Representational State Transfer services
SPARQL	SPARQL Protocol and RDF Query Language
REST Services	Representational State Transfer services
HTTPS	Hypertext Transfer Protocol Secure
REST Services	Representational State Transfer services
XML	Extensible Markup Language
JSON	JavaScript Object Notation
WSDL	Web Services Description Language
IOPE	Input, Output, Preconditions and Effects
SOA	Service Oriented Architecture
CRUD	Create, Read, Update and Delete
YAML	YAML Ain't Markup Language
HATEOS	Hypermedia as the Engine of Application State

CHAPTER 1: INTRODUCTION

1.1 OVERVIEW

In the present programming world, most developers produce microservice and service-oriented architectures-based solutions. The availability of thousands of web services on the web makes the consumers difficult in finding the REST service and understanding such services that suit their requirements. This Research Proposal intends to provide a search engine that queries the semantics of web services and retrieve the best suitable service. Furthermore, this solution will provide semantic specifications for web services.

1.2 PROBLEM STATEMENT

Many developers lately tend to integrate the available third-party APIs to support more features and build better solutions. However, for the developer to interact with the third-party, the service should be known to them. Third-party web services are merely a black box to the public and the source code is not open source or in a public repository. Consumer Developers in many instances face problems locating the correct web service for their requirement as the API developers publish their API with limited documentation attached.

Most of the REST API Producer Developers are reluctant to document their work due to reasons such as the mindset that endpoint names would be sufficient and time constraints of the developer. API published thus will not be used by consumer developers, due to a lack of awareness of API existence. Third-party APIs must be specified in proper structure to have access to consumer developers for better usage.

The API codebase through the syntactical specifications at present included, it is not used for any purpose. Almost all the developers intend to use existing services rather than building service from initiation. The consumer developer to integrate third-party service should be aware of the context of the API.

For a developer, for an instance require to create a new user in an application, the following flow needs to be used. The first developer needs to invoke the Authentication API and receive a valid auth token. Using the Auth token developer needs to invoke the create User API that will create a user.

These APIs must be invoked sequentially order as the User API needs the Auth token to be invoked if not it will return an unauthorized response. Consumer developers must know the flow of execution to invoke API to fulfill the end requirement. The proposed solution will generate OWL-S ontologies in which the consumer developer can examine the ontology and understand the flow.

1.2 PROBLEM DEFINITION

Tech giants like Facebook, Twitter, and Google extend their functionality via REST API services. The Freelances, however, tend to provide “Ready to Market” solutions, for specific domains like banking, pharmaceutical, etc. Most of the tech solutions expose their API to extend their functionality without publishing their code base keeping it as a black box.

Project-based companies from a different perspective, produce a massive number of domain-specific solutions that have in-house APIs developed in and around the globe for years. One of the major issues that face by the internal teams is the lack of knowledge on the current services available in the project. Further, a void is created in the implementation due to not knowing if the service for functionality is already implemented or not. For instance, when a new developer joins a project team, it will take a longer time to understand the domain-specific knowledge and the already existing APIs.

The new joiner is required to implement new functionality, and therefore needs to go through code to understand the availability of an existing implementation that can be reused without reinventing the wheel. The newcomer must search for APIs so that they can utilize the application. It is a difficult task as the internal APIs cannot be googled (not exposed to the public) and naming conversion is based on domain-specific.

Legacy codebases are too slowly transformed into REST Services. Most applications freely expose services using RESTful APIs offering for Consumer Developers. These APIs are consumer-ready and well-tested. Consumer Developers can improve their productivity with innovative ideas to the market in the least time.

API can be a composite of many functionalities. API's when internally executes complex logic it will not be exposed unless it is provided in the documentation. Every developer in general will face the confusion of not knowing what the API does.

1.4 MOTIVATION

As developers, one faces the problem regularly when developing ad-hoc solutions. Lack of awareness of the existing REST services or no knowledge of such services can be time-consuming and have adverse impacts the productivity. In the case of working on large projects that have many modules, facing difficulties is apparent in understanding the bigger picture, hence, tend to reinvent the wheel.

It would be extremely helpful to know the current implemented solution that is well tested than concerning building it from the initial stage, which will save time and cost. Developers in general face difficulties in discovering the most appropriate service to suit their needs. However, having a service discovery is personalized to the user's query and its feedback.

Solutions proposed in this study expect to motivate the API developer to add the swagger plugin to their API. The solutions will direct the consumer who searches an API using the domain ontology model (concept representation) that will return the best result while reducing the gap by bridging the Consumer and Provider developers.

Presently many researchers are focusing on alleviating different aspects of REST-based SOA Applications as REST has become a phenomenon that most developers adopt to nevertheless the programming language used.

1.5 COMPUTER SCIENCE PROBLEM

SOA architecture contains 3 main components. A **Provider** developer develops an API then publishes the API and registers the API in a **Registry**. **Consumers** use a lookup on the registry to find suitable API to fulfill the need. The registry includes meta (basic) information of the API such as the name, endpoint, URL, and reference link. The information saved in the registry is not adequate to do service discovery to get the most optimized API, thus search results are inaccurate in registries.

The process of searching for an appropriate service to a given requirement is called Service Discovery. Automating the process of service discovery is called a service matchmaker. For the process to work accurately the matchmaker needs to contain sufficient information such as the input parameters, response fields, and descriptions (Syntactic). Presently most registries do not capture this information as it is not mandatory to provide when publishing an API.

Unavailability of syntactic information for accurate service discovery problem is addressed by this project by proposing the most optimized service matchmaker approach to extract syntactic from REST endpoints, converting the syntactic to semantics and mapping the semantics to a pre-defined domain ontology.

1.6 RESEARCH CONTRIBUTION

1.6.1 GOAL

To develop a solution that could extract syntactic specification from an HTML, converting the syntactic to semantics and representing the service using OWL-for Web service ontologies (OWL-S). OWL-S properties that will be mapped to the domain ontology model, based on the OWL-S ontologies that provide a search engine. The search engine will give the facility to search by keyword and get the most relevant API.

1.6.2 OBJECTIVES OF THE STUDY

01). Provide suitable ontology model representation for REST services descriptions.

Every service that is discovered by the system creates and represents in form of OWL-S ontologies. OWL-S resources are mapped to concepts represented in the domain ontology. By understanding the domain ontology model. Ontology is modeled using the Protégé tool and access through Protégé API for the search engine.

02). Deep analysis of the SAHTML and other semantic descriptions.

Do prototype implementations of other semantic description approaches for evaluation purposes and better understanding. Measure the convenience and accuracy of this solution and other approaches.

03). Develop implementation that can be used to extract syntactic in an Open API Specification and creating OWL-S ontologies based on extracted semantics.

The implementation would be able to extract syntactic from Open API Specification. Using the semantics create Profile, Process, Grounding, and service ontologies. Services will be register inform of ontologies in the service ontologies.

04) Develop a search engine that filter out the API based on the searched Keyword

The search engine will be developed to be used as Service Discovery, where the end-user can type a keyword and Input/output type. The relevant search will return the most suitable API for the developer based on the domain ontology and OWL-S.

05) Deep analysis of the OWL-S service discovery on REST and matchmaker.

OWL-S specification developed a few decades ago will be analyzed in this specification on SOAP representations WSDL research.

1.7 SCOPE

- The implementation will be in the Java language.
- The REST API source code can be developed by any language but using the Swagger plugin. Thereby its source code should have Swagger documentation.
- The Search engine will web application that can be accessed by the developers. For the project purpose, the scope will limit the universal domain and its concepts.
- Search Engine will return results based on the REST API endpoint fed to the system. This solution won't provide any form of documentation for REST APIs.
- REST API execution nor API mashup won't be included in the scope of this project.
- Ontology will be used to extend as a reference but building new domain ontology is not in the scope.
- Synonyms keywords are not considered.
- Assume the naming conversions used by API matches domain concepts in the domain ontology.
- The project will focus on a small section of an ontology and demonstrate the end to end solution.
- Domain ontology is already existing and covers all the concepts.
- Search Engine will facilitate only a keyword search.

1.8 RESEARCH CONTRIBUTION

The research contribution of the project is mapper between Open API Specification to OWL-S and Search engine built on OWL-S.

The mapper includes,

1. Parsing a Swagger file for retrieving the syntactic.
2. Syntactic to semantics convert that will create OWL-S Profile, Grounding, and Process ontologies by mapping the semantics with resources.
3. The OWL-S resources are mapped to Domain-specific ontology.

Search Engine comprises an algorithm that will query (SPAQL) the OWL-S ontologies and retrieve the optimized Rest Service. Currently, there are outdated methods in discovering services in OWL-S and most of those approaches are not compatible with REST service architecture. The proposed solution has high adaptability to any programming language and a better precision rate in searching services.

1.9 CHAPTER STRUCTURE

The remainder of the research study in terms of chapter structure is unfolded here.

Second Chapter – Literature Review

This section provides an understanding of other implementations done by different researchers over the past couple of years. Study on technical proposed solutions and its limitations to understand the research gap.

The literature review is done following areas.

- REST Services
- Representing Semantics and Deriving Syntactic
- Lightweight Semantic Descriptions
- Microformats Semantic Descriptions
- Mapping semantics to an OWL-S ontology
- Searching REST services based on OWL-S ontology(matchmaker).

Third chapter – Methodology

Describes the methodology followed in conducting the study. Selected approaches from the Literature review is incorporated along with the new solution proposed. The chapter outlines in detail the proposed model, which explains on the technical and specifications used to solve the problem.

Fourth Chapter – Evaluation

This chapter unfolds the different evaluation strategies that were used in terms of evaluating the mapping between the REST service and the generated OWL-S. Also, evaluate the search engine based on the results and searched keyword.

Final chapter - Conclusion

This last chapter presents the overall conclusion of the research study and the key findings reached. Also includes prospective areas of future research related to the study.

CHAPTER 2: LITERATURE REVIEW

2.1 REST

The REST architectural style was proposed by Dr. Roy Fielding [1] is based on the client-server architectural style. The client sends a stateless HTTP request with information requesting from the server. In the server end, the server will receive, process the request, and return a response to the client.

REST APIs request comprises the following,

- **URI** of the web service in the server end that is unique for each web service and carries additional information like pagination and filtering.
- The **data type** the client is accepting as a response from the server. Mostly used data types are YAML, XML, and JSON.
- The **operation** of the REST service request intends the server to execute. For every CRUD operation, there is an HTTP method such as SELECT the HTTP method is GET likewise POST, PUT, and DELETE.

Example Curl of REST API:

```
curl -X GET "https://api.lufthansa.com/v1/references/countries/DK?limit=20&offset=0"  
-H "accept: application/json"
```

REST is known for stateless communication consequently the server or client does not maintain any state in between communications. To have stateless communication there few interface constraints [8] request and response need to adhere to.

1. Self- Descriptive message
2. HATEOS links – additional resources need to be specified as addressable links.

This research can be broken down into two main sections. The first section is deriving semantic information from REST services. The second section is mapping the semantics to an ontology model and searching using keywords.

2.2 ONTOLOGY

Ontology is knowledge base representation in a form of conceptual schema which represents concepts like entities, relationships, and rules. The ontology can be considered as a machine-understandable dictionary [15]. The standard language for defining web-based ontology is OWL according to the recommendation by “World Wide Web Consortium”.

An OWL ontology defines concepts by specifying sufficient conditions and populates a concept by describing its instances in terms of their relations with other individual instances in the domain. The most common open-source ontology editor and framework builder is Protégé. Each concept is describing the entity class of a domain and its relations to its properties. Each concept has a unique URI that can be a reference to any resource in the ontology which makes machines able to access the resource.

2.3 REPRESENTING SEMANTICS AND DERIVING SYNTACTIC

Researchers have proposed different approaches to include semantic descriptions in REST services. The main approach proposed is adding semantics to API documentation. Therefore, when the documentations contain details about the API also includes semantic for machines to understand and process further.

REST services are machine consumable and do not contain any semantics nor details about the REST services. In [1] it points out that there are two descriptions namely **machine interpreted** descriptions and **human-readable** descriptions for any API. Generally, the REST API docs are focused on human-readable documentation as it is a guide for consumer developers on invoking the API.

Traditionally Remote Procedure Call (RPC) model used the Interface Description Language (IDL) to specify documents for static contracts. In the SOAP aspect, the W3C standard documents are WSDL, and XML schema is used. Where else in REST services there is no standard for the document but commonly uses webpage HTML. HTML is human-readable documents that are presented in a such way that is convenient for the developer to refer to.

These HTML documents will contain the URL and in detail of endpoints along with the expected input and output.

E.g. Swagger uses API design code to auto-generate the API doc.

For machines to understand human-readable form documentation, it needs to be annotated semantically thereby when parsing the documentation machine can locate semantics and further process the semantic. REST services that incorporate semantics are called Semantic RESTful Service (SRS). Using semantics, it's possible to explore automation like discovery, negotiations, compositions, and invocations.

The underlying data model for most SWS approaches uses the ontology model thereby it is straightforward to integrate with the semantic web whilst minimizing the interoperability problem at the semantic level.

Challenges face when integrating syntactic descriptions in RESTful services,

1. Automated processes in registration and invocation of API.
2. Accuracy in interpreting service parameters using semantic annotations.

Semantic Descriptions can be classified into various implementations as depicted in figure 1.

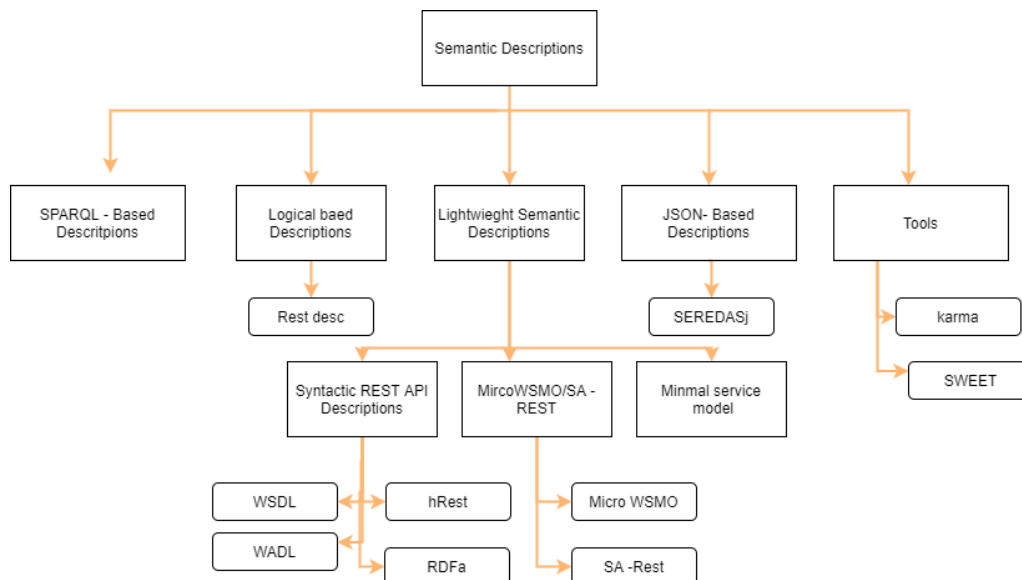


Figure 1: Abstract view currently existing semantic description implementations and approaches

Lightweight Semantic Descriptions

The most common lightweight descriptions are WSDL and WADL (Web Application Description Language) [1]. WSDL is an XML document that describes the web service using predefined tags. WSDL is not recommended for REST services as it's not designed for resource-oriented services and more suitable for operational -based service approaches.

WSDL has difficulty in specifying the message exchange format that limits to HTTP authentication methods and supports new resources that are identified in another document.

WADL [11] is a derivation from WSDL that is specifically designed for REST services. WADL also produces machine-readable XML descriptions that are platform/language independent. The main advantage of WADL is the capability to model the relationship between shared resources.

WADL [8] contains a description for each service resource with its HTTP method, HTTP status code, required input parameters, and how the output will represent. The drawback of using WADL is it's complex as it requires developers to be trained and have the tools to support WADL. Furthermore, the WADL urges to use of a specific resource hierarchy that can result in client-server coupling. The complexity of WADL contradicts the simplicity of RESTful services. Industry use of WADL has decreased with time.

Microformats Semantic Descriptions

Microformats description is the approach that annotates HTML pages with key information without modifying the overall view and these annotations are machine recognizable and processable.

An “adaptation of semantic XHTML that makes it easier to publish, index, and extract semi-structured information”, called microformats [9]. Microformat makes use of XHTML facility in HTML pages for example the *class* and *rel* attributes to annotate semantics to the HTML.

The main advantage of using Microformats is that the developer does not need to put an extra effort as annotation can be done whilst the creation of the documentation web page.

```

<html xmlns:sarest="http://lstdis.cs.uga.edu/SAREST#">
...
  <p about=" http://craigslist.org/search/">
    The logical input of this service is an
    <span property="sarest:input">
      http://lstdis.cs.uga.edu/ont.owl#Location_Query
    </span>
    object. The logical output of this service is a list
  of
  <span property="sarest:output">
    http://lstdis.cs.uga.edu/ont.owl#Location
  </span>
  objects. This service should be invoked using an
  <span property="sarest:action">
    HTTP GET
  </span>
  <meta property="sarest:lifting" content="
    "http://craigslist.org/api/lifting.xsl"/>

  <meta property="sarest:lowering" content="
    "http://craigslist.org/api/lowering.xsl"/>

  <meta property="sarest:operation" content="
    "http://lstdis.cs.uga.edu/
    ont.owl#Location_Search"/>
  </p>

```

Figure 2: An annotated API document HTML.

Main Microformats are **hRESTs** and **RDFa** implementations,

hRESTs (HTML for RESTFUL Services) approach enables developers to add custom HTML tags within the HTML, these custom tags will not be rendered in the User Interface. When the HTML page is viewed a web crawler will scan through the page and use XSL transformation [9] techniques to extract the customer tags and its relevant properties.

```

1  <div class="service" id="svc">
2  <p>Description of the
3    <span class="label">ACME Hotels</span> service:</p>
4  <div class="operation" id="op1"><p>
5    The operation <code class="label">getHotelDetails</code> is
6    invoked using the method <span class="method">GET</span>
7    at <code class="address">http://example.com/h/{id}</code>,
8    with <span class="input">the ID of the particular hotel replacing
9      the parameter <code>id</code></span>.
10   It returns <span class="output">the hotel details in an
11     <code>ex:hotelInformation</code> document.</span>
12  </p></div></div>

```

Figure 3: An example of hREST how the tags are showed in class level

Comparatively this approach is simpler and lightweight than XML-based approaches as it uses HTML tags with more marking options. The adoption of this approach is much easier as it needs only to create machine-readable RESTful service descriptions without writing web content.

Semantic frameworks that are targeting capturing web API characteristics are MicroWSMO and SA-REST (Services based on the Representational State Transfer).

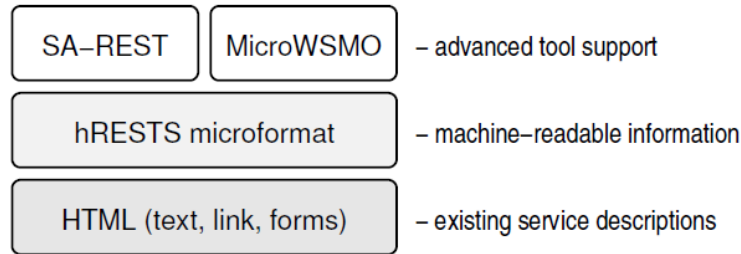


Figure 4: MircoWSMO layered overview.

WSMO (Web Service Modeling Ontology) is the formalism for semantic Descriptions of Web APIS. [2]MicroWSMO which uses hREST (HTML for RESTful services). As Figure 4 shows MircoWSMO is relying on hREST for constructing web service-related properties and uses microformats to add the properties to HTML pages for machine-processable.

MircoWSMO has 4 level concepts [8] to describe the characteristics of web services,

1. It is a domain-specific ontology.
2. is Goals which are needed by the end-user(discovery)
3. Descriptions of the services
4. Mediators.

The limitation of MircoWSMO is that it does not compile with W3C standards and the guidelines for creating a mediator are not mentioned properly in the documentation.

SA-REST - Similar to MicroWSMO, SA-REST also relies on RDFa for making web service-related properties. Properties such as I/O, operations, lifting, lowering or fault and linking semantics entities [5] SA-REST uses meta -models with model reference annotations. SA-REST differs from MicroWSMO from its implementation techniques.

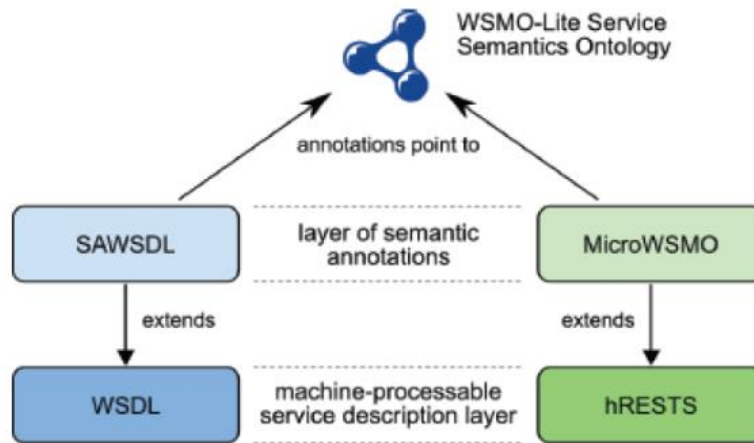


Figure 5: Unifying sawsdl and Microwsmo through wsmo-Lite service

Drawbacks of Microformats is that can encounter CORS issues and authentication issues. It uses a web crawler to detect values in the web pages wherein many browsers have multi-cors is disabled or have high authentication in place.

Other Semantic Retrieving Techniques proposed by researchers,

Research like [2] doesn't include semantics to web pages but uses automated rest service discovery methods and dynamically generate the descriptions. It generates a description by analyzing the input and output parameters of the REST endpoints. First, it will execute the API to gather the information on input and output. [2] the research uses Service Data Object4 (SDO) to perform invocation of API to check the availability. And then once it collects semantics it will query DBpedia ontology to find the corresponding concept or properties.

The drawback of this approach is there can be useless parameters like the page, count limit, etc. those parameters that are needed for frontend table paginations or to display customized warning messages. To query the ontology [2] uses SPARQL endpoint and different external resources like synonyms and suggestions services.

Another semantic REST Service discovery [3] is based on HATEOAS. This research is based on creating ontology based on resource interlink to each other and based on user feedback. For discovering the REST Services uses user profiling techniques with collaborative filtering.

[4] Research recommends agent-based service discovery and content extraction framework. In which the agent(BOT) will visits links based on its goal. It will retrieve data from a web page by analyzing its endpoint transferred through HTTP. After extracting the semantics from the rest service and populate the ontology then next to search from the knowledge base.

[6] Research is an ontology-based framework designed to use functional and non-functional parameters as semantics. Semantic discovery is based on search query terms (UDDI) instead of searching the keywords thus discovery can be performed effectively. The drawback of UDDI is that uses XML semantics and not all the REST service produces XML content type. Most of the research was done a few years back and cannot be applied for current developing techniques.

Mapping semantics to an OWL-S ontology

OWL-S

OWL-S is a subcategory of OWL that is mainly designed to provide web service descriptions in form of objects and relationships. OWL-S specifications were able to reduce the ambiguity between web service resources by conceptualizing class and properties to a domain ontology.

OWL-S ontology has three elements (Sub Ontologies).

- a. Service Profile
- b. Service Process
- c. Service Grounding

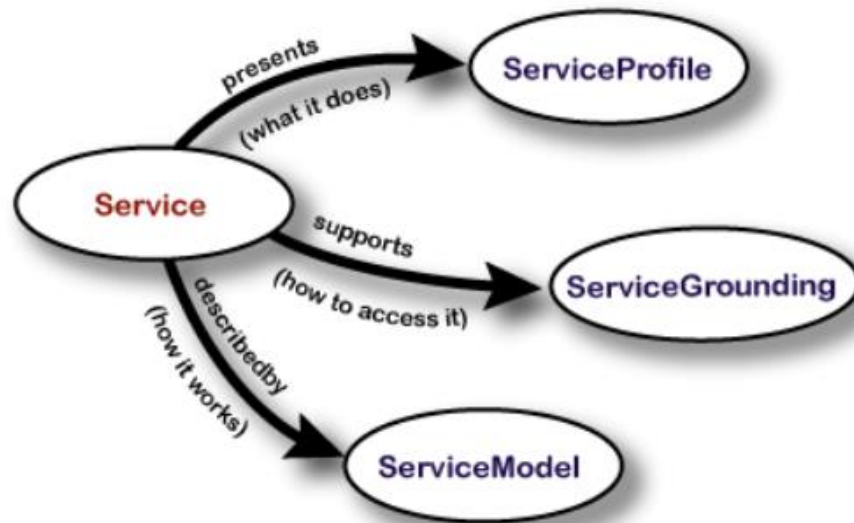


Figure 6: Top level of the service ontology

The Service Profile simply identifies “what the service does” the functional description of a service. This ontology representation comprises a description of the service functionality, service limitations, inputs, outputs, QoS, and preconditions that the service consumer developer must satisfy before invoking the service.

The structure of the OWL-S profile is shown in Figure 6. A profile represents the overview of the service which includes

- a. Information of the service owner/provider – E.g. Contact details of the API maintenance operator
- b. What functions the service will execute after triggering - Inputs that are mandatory by the service and the outputs that are generated as result. Also, it can include a set of condition that needs to be fulfilled before executing the service and expect effect after executing. (**Input, Output, Precondition, and Effect - IOPE**)

E.g. A Selling Service

Precondition – Valid credit card

Input – Credit card number and Expiry date

Output – Generate a receipt.

Effect – The amount deducted from the credit card.

c. Abstract level properties that define the service functionality

E.g. Category – the BIAN classification system

Quality rating of the service (QoS) – User rating and feedback.

Other service parameters - Estimate of the max response time and availability of the service.

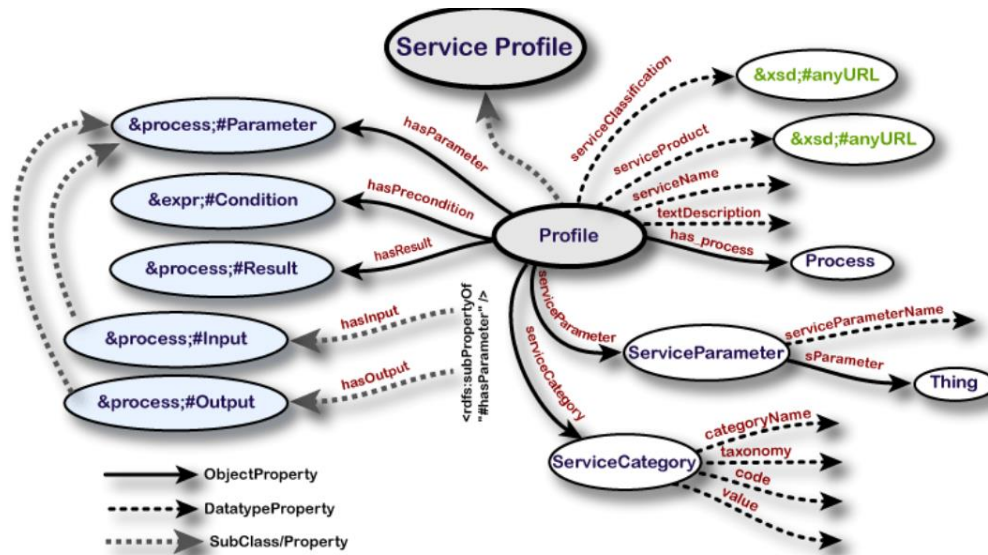


Figure 7: Select the class and properties of the profile

The Service Model identifies how the consumer should consume the service. This explains how to execute service and what occurs when a service is executed. This ontology deals with the semantic content of the request and the conditions or steps that leads to different outcomes.

Based on the way a client will interact OWL-S Specification has two types of process.

Atomic Process is a simple process that expects one request message and returns one response.

The composite Process is the collation of multiple atomic processors that maintain one state.

Expects a message that is passed through multiple atomic processes in which state will change.

The process will generate and respond to information based on the Input and Output given to the process. Also, another factor that can influence the processor is the Precondition and effect.

```

<owl:Class rdf:ID="Input">
  <rdfs:subClassOf rdf:resource="#Parameter"/>
</owl:Class>

<owl:Class rdf:ID="Output">
  <rdfs:subClassOf rdf:resource="#Parameter"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasPrecondition">
  <rdfs:domain rdf:resource="#Process"/>
  <rdfs:range rdf:resource="#&expr;#Condition"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="inCondition">
  <rdfs:label>inCondition</rdfs:label>
  <rdfs:domain rdf:resource="#Result"/>
  <rdfs:range rdf:resource="#&expr;#Condition"/>
</owl:ObjectProperty>

```

Figure 8: OWL syntax to represent IOPE

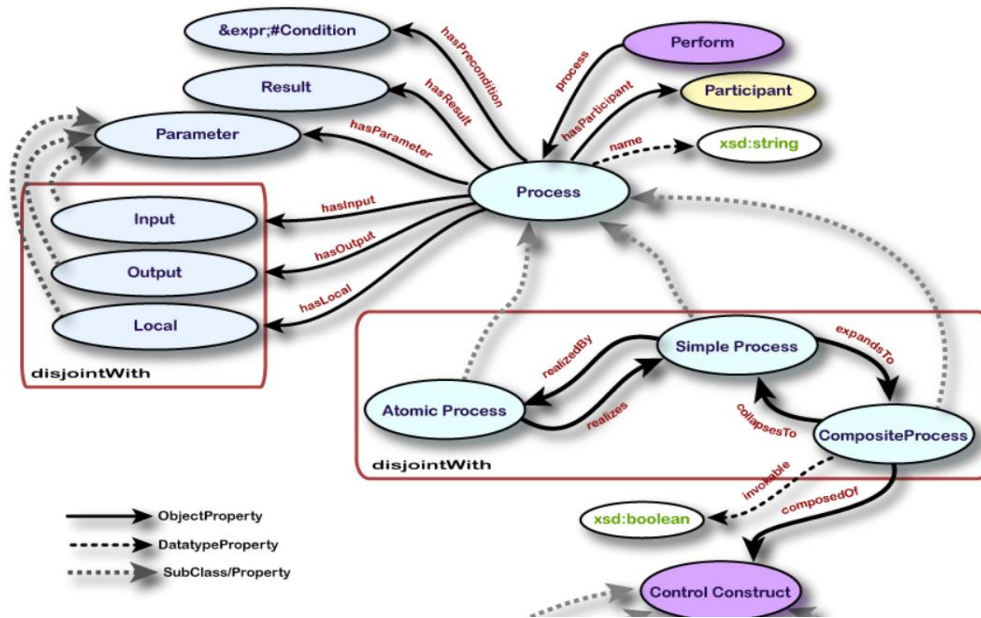


Figure 9: Select classes and properties of Service Process

The Service Grounding specifies how the consumer can access the service. Describes concrete information on communication protocols, message formats (JSON or XML), port numbers serialization, transport, and addressing.

The main function of an OWL-S grounding is to represent how the IOPE of an atomic process is to be sent in messages using a transmittable format. Most previous researchers have utilized WSDL in composing the OWL-S grounding ontology.

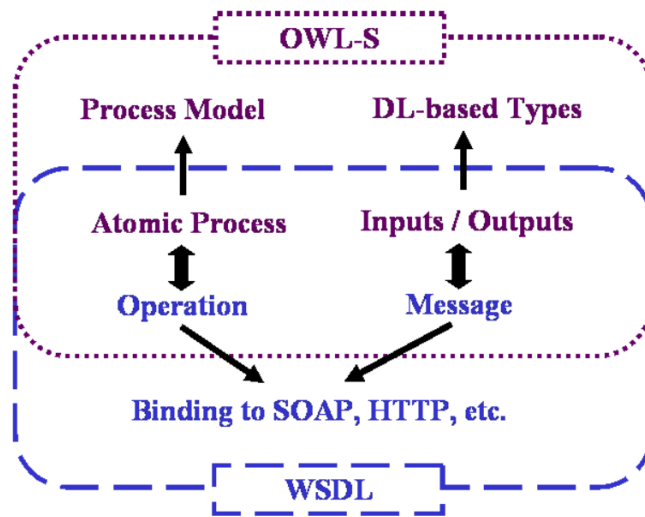


Figure 10 OWL-S and WSDL

This research approach will utilize Swagger documentation in replacement to WSDL where the developer has very little to contribute. Thereby this research will extract the semantics from the swagger and create OWL-S ontology. The disadvantage of using WSDL for REST service is shown in the above explanations.

Searching REST services based on OWL-S ontology(matchmaker).

After retrieving semantics from the REST service, it needs to be mapped to concept ontology. This ontology will be used by search-engine to search REST services. Among REST services there might be linked services or Services that serve the same purpose. [3] proposes to highlight those similarities and complementary relationships between the services and model it to a relationship to OWL-S ontology.

When searching for a service, **query/keyword** will be mapped to the relevant concept in the ontology and retrieve the particular service from OWL-S. The benefit of using matchmaker is requested service and services in the registry are uniformly described based on the same OWL-S ontologies regarding Doman Ontology therefore, it is useful to compare input and output.

Most common Service discovery approaches,

1. Based on category and Keyword-based approaches are too coarse-grained.
2. Syntactic specification approach that is known for retrieving false-negative and false-positive results

Universal Description Discovery and Integration (UDDI) discovery is a recognized standard repository for web service. UDDI technologies are an accurate matching process built on the main registration information description of API. It keeps a registry of the service when registering/onboarding service to the centralized portal. Using the information provided onboarding, it will search.

Support service discovery with a central registry (i.e. UDDI) that contains all the meta-data necessary to describe available services. [11] Web service discovery is about searching the web services repository (UDDI) for the desired web service. This technique is not suitable for its low matching precision. UDDI was designed for SOAP services; however, it is not even used for those anymore. UDDI was depreciated around 2006 as alternative presently DNS service discovery is used.

The semantic description proposed by [3] considers the Web as a graph where nodes are services and edges are the semantic links between them. Moreover, semantic annotation links between the services: "Is-Similar" and "Is-complementary" allows us to navigate in the graph.

[10] Proposes a service registry framework based on an ontology model called Concept Operation Ontology Model (COOM). It contains 3 ontology layers. Concept Ontology Layer, Operation Ontology Layer, and Association Ontology. Concept Ontology and Operation Ontology connected by Association Ontology.

The concept Ontology layer displays the relationship between the concepts of the cooperation relationship and the Operation layer includes the operations for each concept also represented as a class. The Associate class will inherit from concept and operations classes and do the relevant searching.

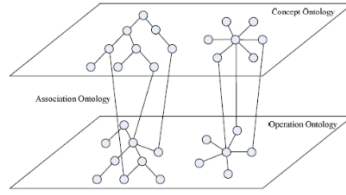


Figure 11: The Structure of Concept Operation Ontology Model (COOM)

The associate model can be used as a search engine. Where end-users can search for a keyword where it will find similarities in the existing services and return the highest service with the matching score. The service requested based on the matching score will get the available web service endpoints through the service pool in the services Database. [10] the formula to find the matching score.

Equation 1: Precise Matching

$$\text{Sim}_{sm}(S,Q) = \alpha_{\text{concept}}\text{Sim}_{cpm}(S,Q) + \beta_{\text{operation}}\text{Sim}_{opm}(S,Q).$$

[11] for precise matching, we should have domain-specific semantic knowledge where terms are connected based on their meaning in the context of the problem.[17] Firstly, the services will be filtered by UDDI and the resulting services will be ranked based on the above match degree Besides providing a discovery engine in UDDI, this framework builds a query engine to put OWL-QL statement over the integrated profiles. OWL -QL is a legacy approach that is not currently used.

Another popular IOPE approach for searching web services in ontology is Hierarchical capability-based matching [18].

The algorithm is written to find the minimal distance between concepts in the taxonomy tree. The 4 degrees of matching according to the rules are displayed in fig 12.

OutR => One output of requested service

OutA => One output of the service advertisement.

1. Exact – If $OutR = OutA$
2. Relaxed – If $OutA \text{ subsum } OutR$
3. Subsume – If $OutR \text{ subsum } OutA$
4. Fail – Mismatch between ontology concept

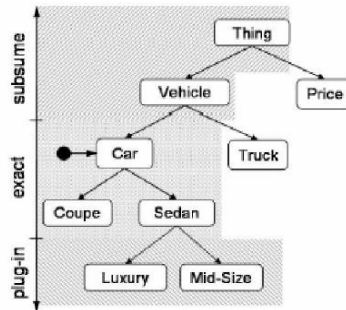


Figure 12: Hierarchical capability-based matching

A three-layer filtering matching algorithm for OWL-S was proposed by [19] research. Where service matching module looks up services based on a three-layer filter algorithm and returns to the end-user a set of services sorted by similarity.

Three-Layer Filtering Algorithm is,

1. Basic Description matching filtering. – Compares the description of service requests with that of all service profiles registered in the OWL-S.
If one matching score is higher than the threshold it will be filtered by 2nd filter if not terminated.
2. IOPE matching filtering - The second filter is in charge of matching the Inputs, Outputs, Precondition, and Effects respectively between service request and OWL-S Profile.
3. QoS matching filtering – Filter against a set of QoS attributes and offer the most accurate QoS concept.

In this research, it will utilize the three-layer similar filter to find the best suitable service for a given search keyword.

[15] proposed three methods of service discovery in OWL-S

1. Match Making of Web Service Profiles
2. Input /Output Types Matching
3. Precondition Effect/ Analysis

These methods are like IOPE methods mentioned above which this research project utilizes for service discovery.

2.4 RESEARCH GAP AND LIMITATIONS IN EXISTING APPROACHES

Many OWL-S related types of research were done based on WSDL decades ago. There is no holistic approach researched previously deriving semantic specification from the syntactic specification. Then using the semantics in linking to OWL-S ontologies using Domain concepts.

CHAPTER 3: METHODOLOGY

3.1. PROBLEM ANALYSIS

A. What is Semantics in REST?

Features of the HTTP protocol such as HTTP Verbs, HTTP status code, and HTTP Authentication that is included in the input and output of REST calls.

1. URL Structure

The endpoint URL includes the path to an endpoint.

e.g Store locator API has few endpoints like,

/v1/banks/1264 - Return the bank that has id 1234

/v1/banks - Return all banks

Each of the endpoints has its HTTP Verb, figure 13 shows the all possible CRUD operations REST could have.

HTTP Verb	Description
GET	This is the most common verb, and is used to retrieve data, e.g. GET /v1/stores/1234
PUT	PUT requests are commonly used to update/replace an object, e.g. PUT /v1/stores/1234
POST	This is used to create a new instance of an object, e.g. POST /v1/stores
DELETE	As the name suggests, this will delete the object, e.g. DELETE /v1/stores/1234

Figure 13: List of verbs supported by HTTP protocol for REST

2. Input and Output parameters of the API endpoint

Input parameters that need to be passed to execute endpoint can have semantic meaning included. The parameters need to be filled for the endpoint to execute thereby it will contain information related to the endpoint.

The output parameter more often is an object that includes a lot of information about a specific entity.

3. Descriptions

There is the API level description that gives an overview of the API and there is an endpoint level description that gives information on the endpoint.

4. Authorize and versioning

The security aspect of an endpoint is taken care of by Authorization and that how paid APIs limit their access to their API.

Most of the API has versioning as time passed by most companies release versions of APIs. The main reason to release an API version. Once you release the version there will be consumers who will use the API in their codebase. Then it's not possible to do any change to the API from service providers as it will be breaking the consumers' codebase. Usually, the version is embedded in the base URL. E.g. [Base URL: api.lufthansa.com/v1]

B. Where to include semantics in REST?

As describe in the literature chapter there are possible approaches suggested by researches.

From annotating source code to annotating REST docs. SOAP services have standard documentation that is WSDL but in REST there is no standard as such, but good practices and specifications to follow. That is having a simple unstructured HTML page for REST consumers. The HTML Page will contain service descriptions and detailed endpoints.

Developers are reluctant to include semantics because from the developer's point of view to execute the endpoint semantics are not required. Likewise, it's time-consuming writing descriptions and detailing the service. Most of the developer prefer coding the endpoint than writing about the endpoint.

To solve the issue there is a new trend that most of the developers adapt to. That is using open source documentation plugins that are readily available to use (plug and play scenario). By that means the developer has very minimal effort in documenting their API and worrying about the styling of the documentation.

For examples,

1. Swagger UI –The OpenAPI Specification (OAS) provides a standard, language-independent interface for RESTful APIs. Provides discover and understanding capabilities of APIs to computer and human without accessing the API source code.

[20]. Also Swagger gives the capability to execute the endpoint in the swagger UI. If someone wants to test the endpoint, they can execute the API on the swagger or get an executable Curl command. Fig 14 shows a dummy REST service's documentation using Swagger UI.

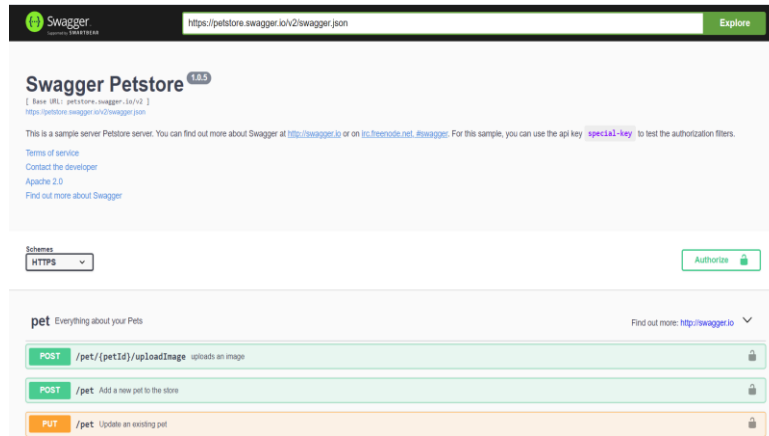


Figure 14: UI of Swagger

2. Spring REST docs.

Some programming languages provide an in-built documentation feature for automating the documenting the REST endpoint. Likewise, the Spring Application framework provides Spring REST docs. It combines hand-written documentation written with ASCII doctor and auto-generated snippets produced with Spring MVC Test.

The main factor on this is that when the MVC test run it will make sure that every parameter and response (by mocking the call) is documented if not it will through an error. Disclosing the part of the REST endpoint code not documented. Thereby it will be keeping the source code and the documentation up-to-date always. However like swagger this spring doc there is no possibility to execute the API in the UI and test the API.

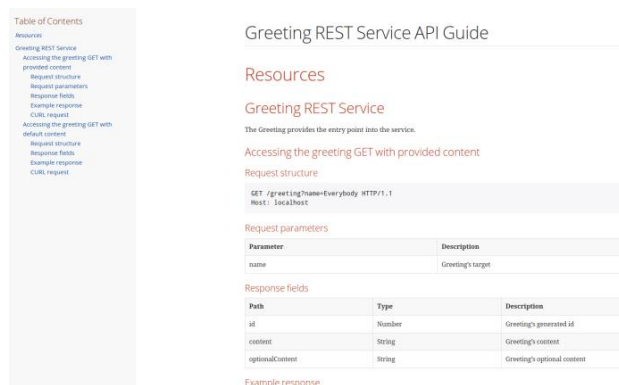


Figure 15: UI of the Spring doc

B. How to extract syntactic specifications from documentation or source code

As mentioned above semantics can be found in source code or HTML form. Which means it can extract multiple ways. The extracting approach can vary independently from the programming language the REST service is built.

Extracting requires to be done in a structured way as the semantics in the REST service are interlinked to each other. For example, inputs syntactic should belong to specific endpoints syntactically likewise the structure of the semantics is important.

As the literature review shows previous researches have used many techniques to extract semantic from REST service.

1. Using a registry – Having a central registry for API. Every API needs to be registered so when registering the API. The semantics can be extracted from the API.
2. Web crawler – Where web crawler is scan through the HTML to detect specific tags and identify the semantic included in the HTML.
3. Agents – Using agent-based extraction. Where a specialized agent will detect semantics of REST on a source code.
4. Using Parser – Using a third party parser to extract semantic from Swagger file or any API documentation. (In detail in next section)
5. Using annotations – Using custom annotations in the source code where the developer can annotate the rest method with semantics and that can be used to extract semantics.

In this research initial plan was to use annotations. An annotation approach was implemented in PoC to check the feasibility. Fig 16 shows the approach on the controller and add semantics to each of the endpoints.

```

18
19
20 @RestController
21 @RequestMapping("/availablebalance")
22 @Api(value = "AvailableBalance", description = "An API for Bank operations", tags = "Bank ")
23 public class AvailableBalanceController {
24
25     @GetMapping("/availablebalance")
26     @ApiOperation(value = "Get all the availablebalance", response = Bank.class)
27     public ResponseEntity<String> listAvailableBalances(@RequestParam(required = true) int page, @RequestParam(required = true) int size) {
28         return ResponseEntity.ok("availablebalance.getContent()");
29     }
30
31     @GetMapping("/availablebalance/{id}")
32     @ApiOperation(value = "Get the Bank info for given Bank Id", response = Bank.class)
33     public ResponseEntity<String> getAvailableBalancesById(@PathVariable("accountNumber") int BankId) {
34         return ResponseEntity.ok("availablebalanceeervice.getBankById(BankId)");
35     }
36

```

Figure 16: Code snippet of implemented annotation approach

This approach after analyzing the effort the service developer must go through was comparatively higher than other approaches. The developer must annotate the endpoint with semantic these custom annotations were provided by this research component. After annotating this custom component will extract values in the annotations and process. One of the main failure of this approach noticed was the source code needs to be exposed to extract the semantics.

One of REST's main advantages is hiding the source code but exposing its functionality as a black box. Also providing a new set of custom annotations to developers was extra troublesome for the developer as there is already a lot of annotations are used. Another failure was this annotation approach was restricted to JAVA language only.

C. How to generate OWL-S models.

As literature review pointed out that their existing conversion from WSDL to OWL-S. But there is no proper conversion from the REST document to OWL-S. Mainly this issue is that WSDL has more well-defined structural tags involved.

Therefore, when converting WSDL to OWL-S is easy as its matter of mapping the tags to relevant resources in OWL-S. Although when it comes to semantic retrieved by HTML file semantics will not exactly map to OWL-S. REST service semantics can be modeled as OWL-S (Further details in the next section)

OWL-S has references to domain Ontology. Those resource needs to find the matching resource in domain ontology amidst handling ambiguity. Here are few matching related issues that can be faced,

1. Exact word match.
2. If not, exact word match then using wordnet ontology to find the synonym word.
3. Might need to know the context to find the best suitable resource in Domain Ontology.
4. Domain Ontology might not be covering this concept.
5. If there are domain-specific words. Third-party lib for mapping is needed. (Eg Financial Data)

D. Search a service

As literature review pointed out that there many matchmaking algorithms proposed by previous researches. When a keyword is a search should return the most relevant services based on a ranking. From the most suitable service wise to the request. Search engine results should provide necessary information for the consumer to understand. Not only the service name should return.

Other related details as well should be returned. If not, a consumer should have extra effort in finding this service related information. Also, even this functionality is not scoped in this research there should be a way for a user to type their query in the search box.

E.g. I want all the flights traveling from Colombo to London.

A query should be converted from NLP (Natural Language Processing) to SPAQL (Query Language for Ontology) to return results.

3.2 PROPOSING MODEL/DESIGN

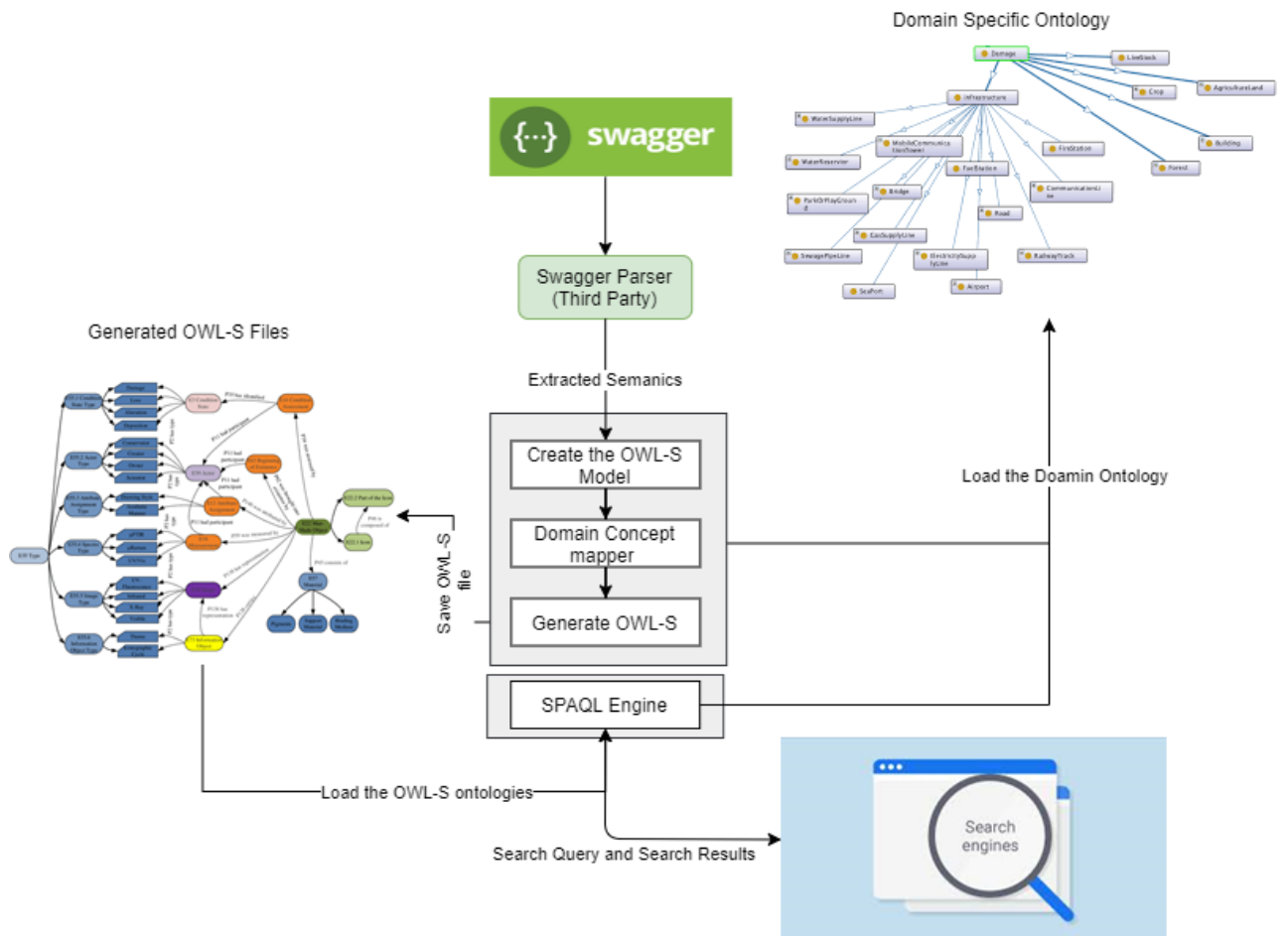


Figure 17: High-level Architecture Diagram

For semantic extraction, the proposed solution uses the Swagger- Open API Specification document (JSON file). The file will be parsed using the open-source swagger-parser library and retrieve the POJO class with mappings.

The proposed solution will create an instance of OWL-S ontologies for each service that is found in the Swagger file. Each left node in the OWL-S is referred to concept in the Domain Ontology. Domain Ontology is an ontology that is already existing that act as the knowledge base for the domain-specific concepts. Once the OWL-S ontology model was created with all the services and semantics mapped. The search engine will query the OWL-S and Domain Ontology to search for services.

Swagger File and Swagger Parser

Swagger File is a document that defines or describes an API and its endpoints. Helps to generate, visualizing, and API docs.

Important elements in syntactic specifications are,

1. Operations
2. Input of operations - raw data the service consumes.
3. Output of operations - data produced by the service.
4. Type of Input and Output – Data schema that prescribes the valid input and outputs.

Reasons to choose to Swagger File from other semantic approaches,

1. Swagger is used as good practice for API development thereby with this solution developer doesn't need to spend any extra
2. effort as it's reusing already existing documents.
3. Recognition - Microsoft, NorthernWestern University, Aurora, Facebook and google use Swagger as their API documentation for in-house built APIs.
4. Open Source – Apache License. Less cost and better credibility
5. Compatible with many Programming languages - Possibility to integrate the swagger plugin to JAVA, Clojure, ColdFusion, C++, Go, Haskell, JavaScript.

```
@CrossOrigin("**")
@RestController
@RequestMapping("/accounts")
@Api(value = "account", description = "An API for Account operations", tags = "Account API")
public class AccountController {

    @Autowired
    private AccountService accountService;

    @GetMapping("/account-balance/{accountNum}")
    @ApiOperation(value="Get the balance for the given account number", response=BigDecimal.class)
    public ResponseEntity<AccountBalanceResponse> getAccountBalance(@PathVariable("accountNum") String accountNum){
        BigDecimal accountBalance = accountService.getBalance(accountNum);
        if(accountBalance == null){
            throw new SmartBankException("Account number not found");
        }
        AccountBalanceResponse accountBalanceResponse = new AccountBalanceResponse(accountBalance);
        return ResponseEntity.ok(accountBalanceResponse);
    }
}
```

Figure 18: Example of Java code that is annotated with Swagger annotations

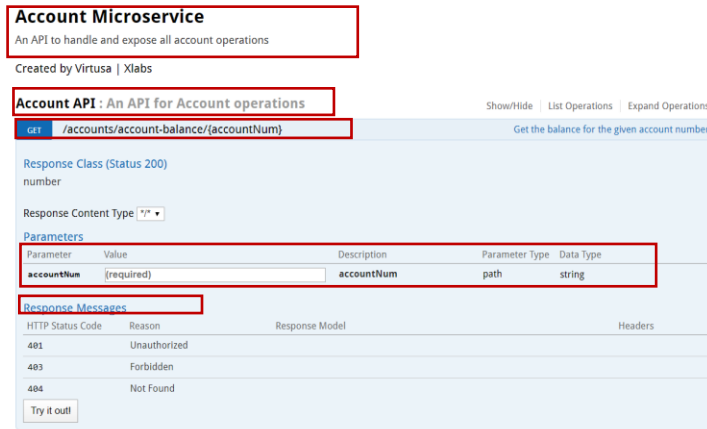


Figure 19: Example of the above service shown in Swagger UI.

The swagger file can be imported to .yaml or JSON file that is usually used for publishing web services in API portals. Boxed values are the semantics the swagger parser will extract and give.

Domain Ontology

In the software engineering industry, Domain knowledge is a key factor as it's very essential when developing a client-specific product. Domain knowledge transferring from an experienced developer to a new developer is exceedingly difficult in software engineering. Most domains have developed their ontologies for example banking, airline, and production domain.

Ontology is used to visualize the concepts involved in a particular domain and gives a bigger picture of the domain. Therefore any person who looks at the domain ontology should be able to get an overall overview of the knowledge of concepts used.

Other universal ontologies are knowledge bases for the semantic web. For example,

1. DBpedia – Contains Structure information that is extracted from Wikipedia pages and currently the largest ontology in the world. DBpedia contains 6 million entities that can be accessed via querying SPARQL. DBpedia has an API facility where querying can be done by executing an API. Over the dataset was too large for query the throughput time was high therefore DBpedia was not considered.

2. Schema.org – In this research as domain ontology Schema ontology was considered. It is also containing structure metadata found in the internet web pages. And contain a large vocabulary that covers entities and their relationships It was founded by Google.

Google uses predefined types from schema.org to map metadata in web pages. So that when query google uses these metadata linked to Schema.org for better search. The below screenshot shows how google uses schema.org to get metadata from web pages.

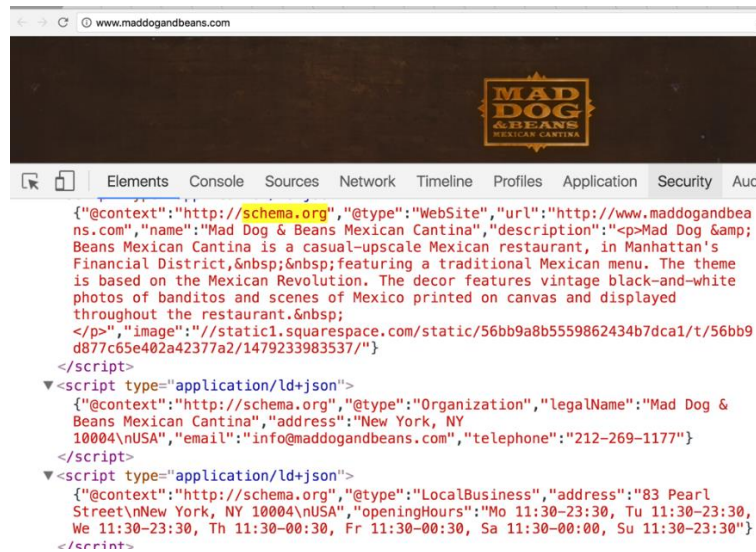


Figure 20: How Schema.org ontology gets populated using HTML meta tags in web pages

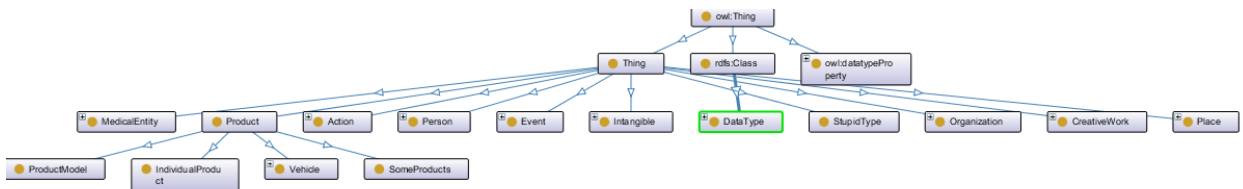


Figure 21: Abstract view of Schema.org ontology

OWL – S Generating Component

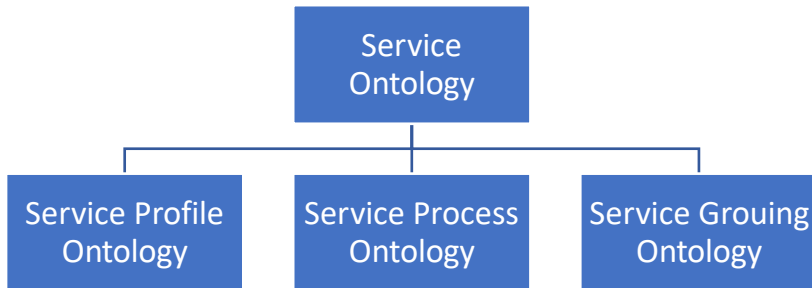
OWL-S is an OWL ontology specifically created for web services, that is used by web service providers to represent their service capabilities and properties in unambiguous, computer-interpretable form.

By converting REST service to OWL-S ontology that enables tasks that user can utilize,

1. Automatic Web service discovery
2. Automatic Web service innovation
3. Automatic web service composition and interoprations

This research proposes a model where REST service is converted to OWL-S model.

In OWL – S there are 4 ontologies. The main ontology is Service Ontology. Which is a directory of services with basic information and reference links to sub Ontologies.



OWL -S Specification has provided 4 Upper ontology for service. Those are namely,

1. Service.owl
2. Profile.owl
3. Process.owl
4. Grounding.owl

These upper ontologies mainly provide the object property relationships such as hasInput, hasOutput, describe. Also, it provides OWL-S specific tags that can be used when developing OWL-S. The prototype created for this research imported base ontologies and created ontologies for each service. OWL API was used to create ontology through code.

Generation of OWL-S Service Ontology

Generating OWL-S ontologies starts from the main Ontology that is the service Ontology. When a new service and its semantics are processed firstly it will create an instance under the service class. As shown in Fig 16 all the APIs are an instance of the class *Service*. Fig 22 shows the screenshot of the prototype created for this research.

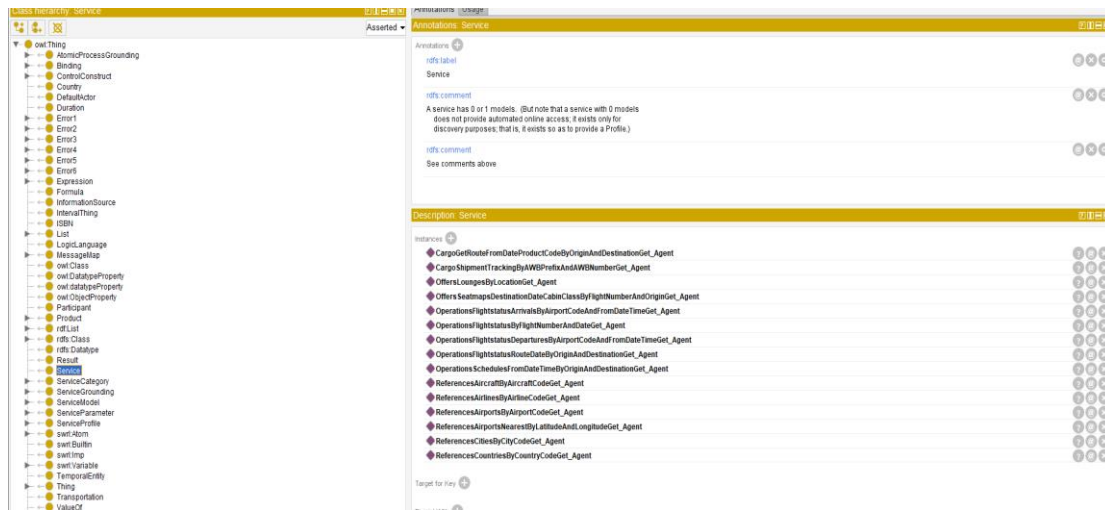


Figure 22: Protégé view of instances for a service

Then each service instance will have 3 object properties linked to it. Those are references to sub ontologies instances created for a service using semantics.

describedBy -> Service Process instance for that particular Service.

presents -> Service Profile instance for that particular Service.

Supports -> Service Grounding instance for that particular Service.

Also, will have one data property populates to store the base URL.

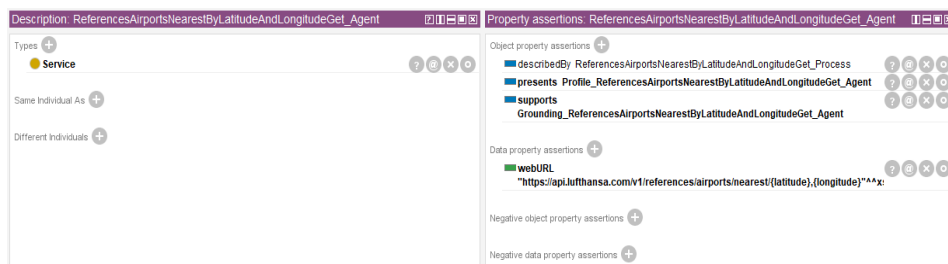


Figure 23: Object Properties of Service instance

Generation of OWL-S Service Profile

For every service, there will be a service profile. Service Profile will include information like service description, service name, and related input/ output process. According to OWL-S specification, a subclass should be created under *serviceProfile* for each service and each class will have an instance called an agent.

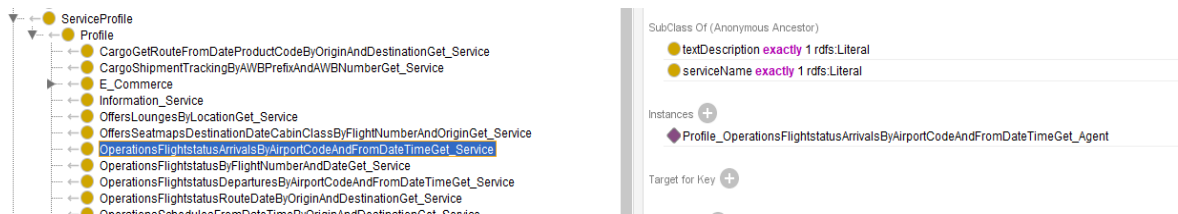


Figure 24: Protégé view of Service Profile subclass for a new service call Operation Flight

These agent names are taken from the operation Id that is passed by the swagger.

Agent instances will carry all the input and out processes (Processes that are created by Service Process Ontology) as object properties. Descriptions and service name will be saved as data properties for reference.

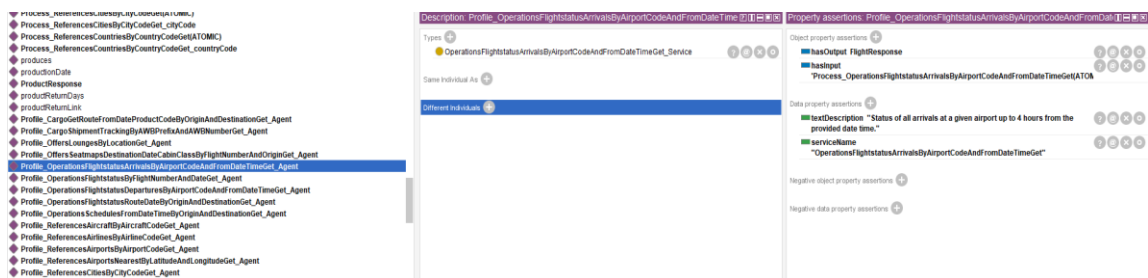


Figure 25: Protégé view of the agent instance.

For service discovery Service Profile is used. As the literature review mentioned most searching algorithms are based on the profile ontology.

For example, when the user searching for a particular service. The user is given a Form to fill the input and output user is expecting in the service that is searched for. When the user fills in those details, it will be converted to a profile.

Those profiles are called advertisements. There are already implemented matchmakers that compare an advertisement and a set of service profiles. The matchmaker will return the most compatible service profile to the advertisement.

Generation of OWL-S Service Profile

The parameter that was extracted is now used to create process and fill object properties. One service in Swagger is matched to Service in OWL-S. That Service will have an Atomic process that is populated by Operation (In Swagger). The operation always has a Response and Parameter Request. Those will be mapped to Input and Output in OWL-S.

Those parameters or response values are mapped to a Domain Ontology Concept. if there is a parameter called latitude it will be matched to a resource in domain Ontology called <http://schema.org/latitude>. For this mapping, this research suggests loading the domain ontology to the system and traverse the ontology.

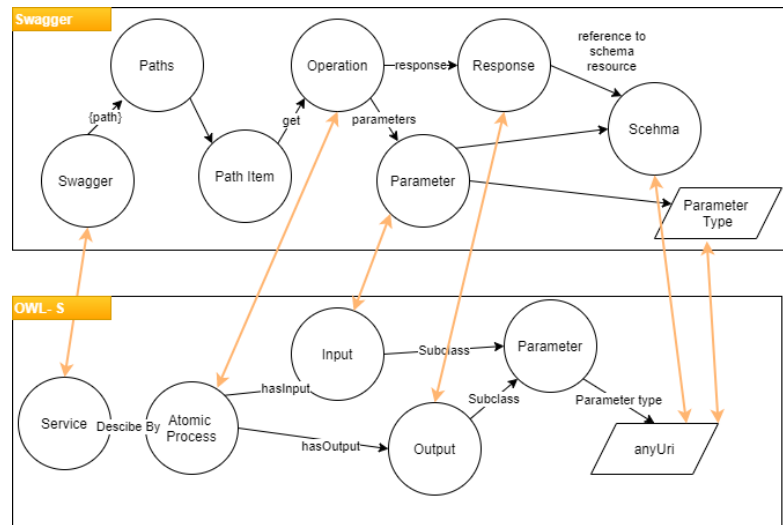


Figure 26: illustrates how Swagger objects are mapped to OWL-S Process concepts.

When implementing this scenario in OWL-S. The atomic process is created and it will have a subprocess for each input parameter. The atomic process will have “hasInput” relationship to the subprocess.

The atomic process will carry few annotations as well.

1. `rdfs:label` – that is the name of the atomic process that is taken from swagger semantics.
2. `Rdfs:Comment`- Will include a description of the input parameter that was extracted from the swagger.

These annotations are populated for future references.

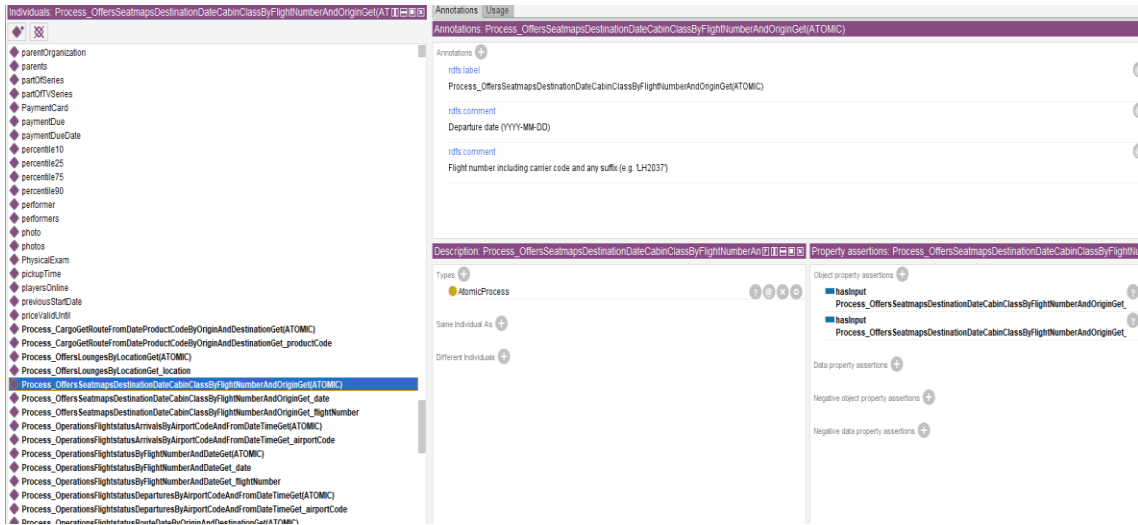


Figure 27: Protégé view of Atomic process instance with multiple input parameters.

Sub Process will have the details of a parameter like a parameter name and the reference concept to the Domain Ontology.

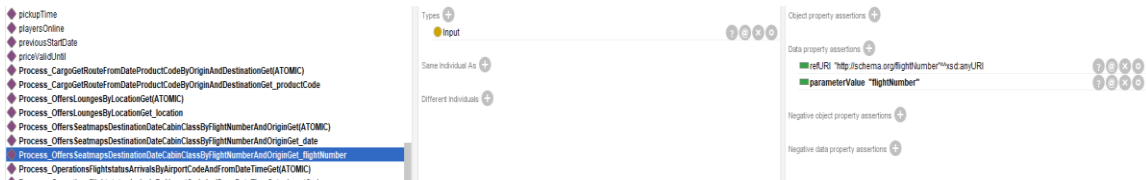


Figure 28: Protégé view of an Input Process instance

The same implies to the output parameter. It will have an atomic process with multiple subprocesses. Each subprocess will be *Output* type and will have data properties

1. `refUrl` - the URI of the matching concept in Domain Ontology
2. `parameterValue` – parameter value that was passed from Swagger.

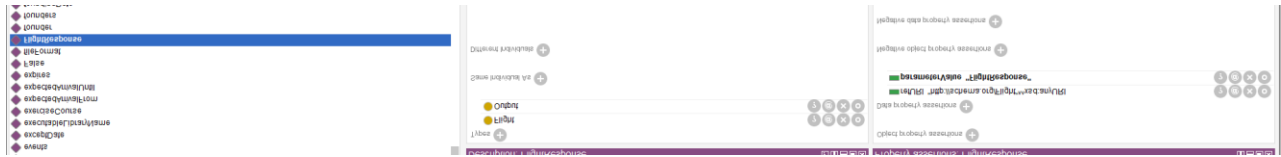


Figure 29: Protégé view of Output Process instance

Generation of OWL-S Service Grounding

For service grounding information like base URL and authentication is included. The main service instance will have supportBy object property to grounding instance. The grounding instance will have a mapping to the grounding atomic process.

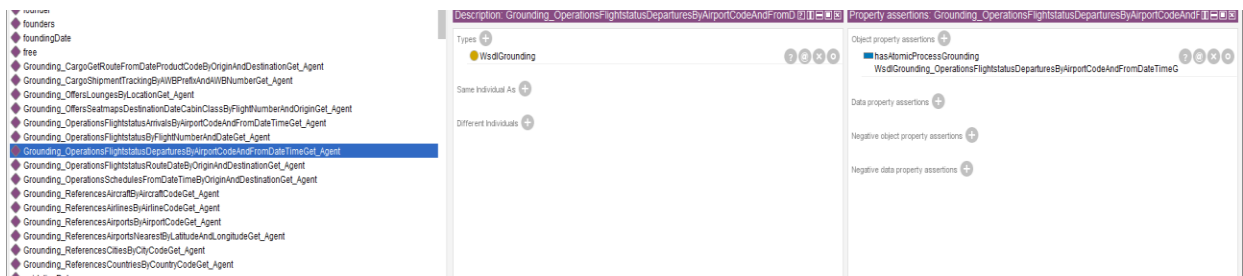


Figure 30: Protégé view of Grounding instance

At the end of the generation, this solution creates 4 OWL files interlinked to each other.

As mentioned before the OWL-S files can be used for various tasks like service discovery and service compositions. Mainly when a consumer developer visualizes this OWL-S ontology, the developer will gain a better understanding of how the services are functioning.

Also, if we check the domain ontology point of view. If a person goes through the domain ontology and finds a concept. That person can easily find the relevant service attached to the concept and in detail information about the service.

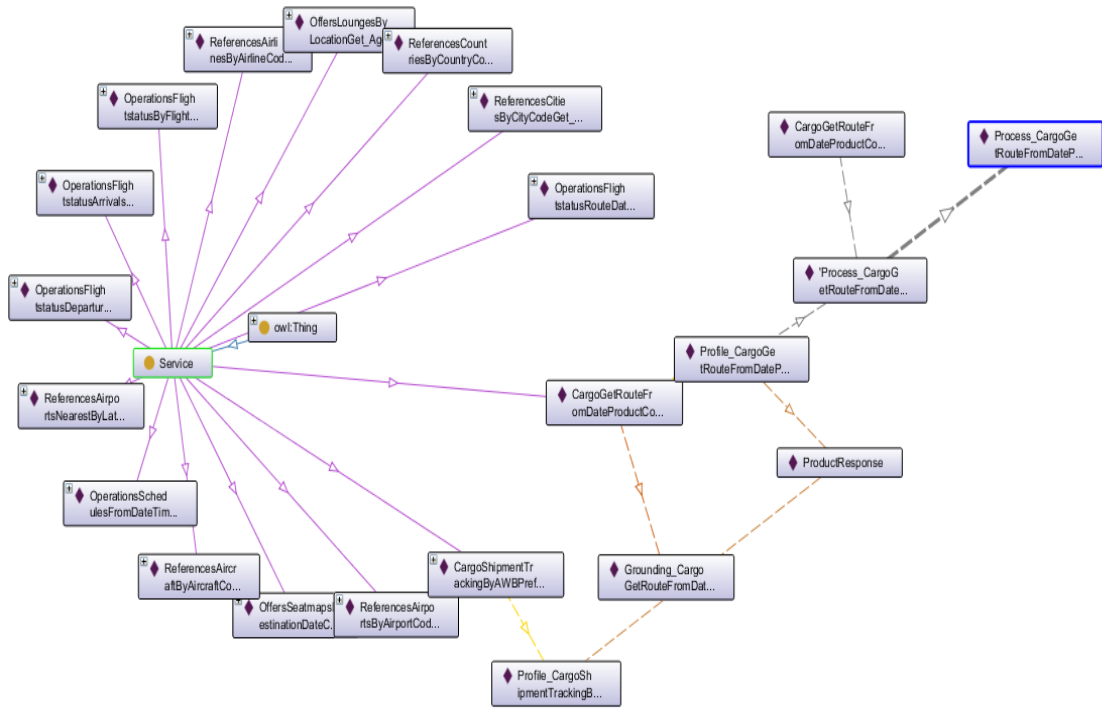


Figure 31 : Illustrates high level of classes, instances in the created OWL-S ontologies.

Search Engine based on generated OWL-S

One of the main benefits of having service OWL-S ontologies is that it can be used for service lookup. The consumer Developer will be the end-user for this search engine.

Developers can visualize the domain Ontology and get the required concept needed for his need and that using that concept he can search for relevant service using the search engine.

Users can query by keywords or using expected inputs and output. Once the user has entered the value. It will be sent to the backend and it will process the keyword value. For searching for a suitable service, the implemented prototype used SPAQL query language. For running SPAQL on OWL ontologies there is third party lib called ONT-API that is built over the Apache Jena framework.

Services can be searched by input and output, In Fig 32 shows the result of a service that has location as the input and airport as the output (IOPE matching algorithm is used).

Search Engine

Ontology based search engine for webservices

Search By Keyword	<input type="text" value="Keyword"/>	<input type="button" value="Search"/>	
Search By Input/Output	<input type="text" value="location"/>	<input type="text" value="Airport"/>	<input type="button" value="Search"/>

Search results

Service Name	Service Description	Endpoint URL
OffersLoungesByLocationGet	Lounge information	https://api.lufthansa.com/v1/offers/lounges/{location}

Figure 32: Search Engine UI that implemented for the prototype of this research

For this research purpose, searching keywords are limited to the concepts found in the domain ontology. Search engine results will contain,

1. Service Name
2. Service Description – End-user can read the description and understand whether the service matches the needs.
3. Endpoint URL – If the end-user wants to test the URL before integrating the service. Users can use the URL provided to use POSTMAN to execute the service.

Search Engine

Ontology based search engine for webservices

Search By Keyword	<input type="text" value="airport"/>	<input type="button" value="Search"/>	
Search By Input/Output	<input type="text" value="Input Param"/>	<input type="text" value="Output Param"/>	<input type="button" value="Search"/>

Search results

Service Name	Service Description	Endpoint URL
OperationsFlightstatusArrivalsByAirportCodeAndFromDateTimeGet	Status of all arrivals at a given airport up to 4 hours from the provided date time.	https://api.lufthansa.com/v1/operations/flightstatus/arrivals/{airportCode}/{fromDateTime}
OperationsFlightstatusDeparturesByAirportCodeAndFromDateTimeGet	Status of all departures from a given airport up to 4 hours from the provided date time.	https://api.lufthansa.com/v1/operations/flightstatus/departures/{airportCode}/{fromDateTime}
ReferencesAirportsByAirportCodeGet	List all airports or one specific airport. All airports response is very large. It is possible to request the response in a specific language.	https://api.lufthansa.com/v1/references/airports/{airportCode}
OffersLoungesByLocationGet	Lounge information	https://api.lufthansa.com/v1/offers/lounges/{location}
ReferencesAirportsNearestByLatitudeAndLongitudeGet	List the 5 closest airports to the given latitude and longitude, irrespective of the radius of the reference point.	https://api.lufthansa.com/v1/references/airports/nearest/{latitude},{longitude}

Figure 33: Search engine results for keyword "Airport".

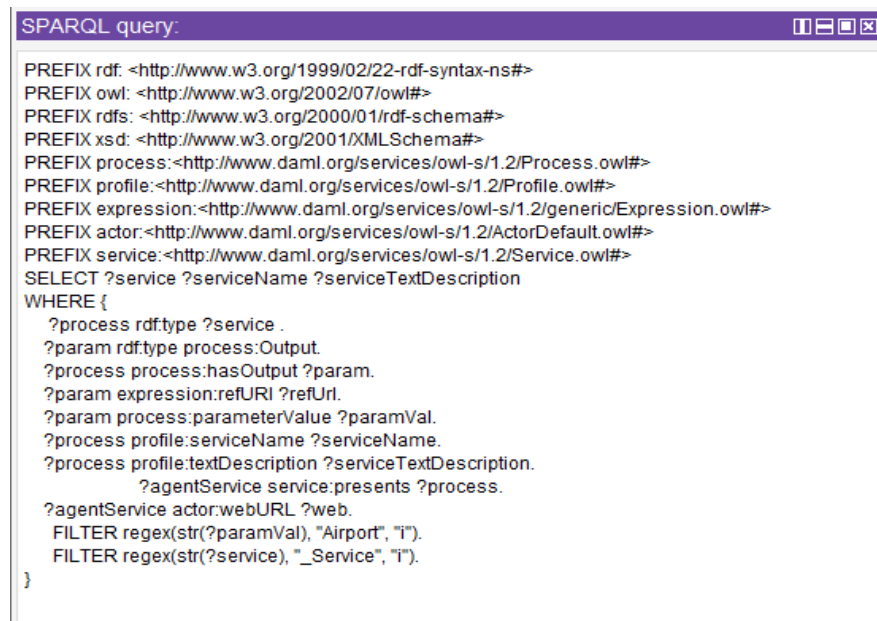
Services can also be searched by keyword. Fig 33 shows the results of services that have searched keyword in a text description, in Output, or Input.

For this research purpose, searching keywords are limited to the concepts found in the domain ontology.

Search engine results will return the below details,

1. Service Name
2. Service Description – End-user can read the description and understand whether the service matches the needs.
3. Endpoint URL – If the end-user wants to test the URL before integrating the service. Users can use the URL provided to use POSTMAN to execute the service.

SPAQL Queries used by a search engine is,



```
SPARQL query:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX process: <http://www.daml.org/services/owl-s/1.2/Process.owl#>
PREFIX profile: <http://www.daml.org/services/owl-s/1.2/Profile.owl#>
PREFIX expression: <http://www.daml.org/services/owl-s/1.2/generic/Expression.owl#>
PREFIX actor: <http://www.daml.org/services/owl-s/1.2/ActorDefault.owl#>
PREFIX service: <http://www.daml.org/services/owl-s/1.2/Service.owl#>
SELECT ?service ?serviceName ?serviceTextDescription
WHERE {
  ?process rdf:type ?service .
  ?param rdf:type process:Output.
  ?process process:hasOutput ?param.
  ?param expression:refURL ?refUrl.
  ?param process:parameterValue ?paramVal.
  ?process profile:serviceName ?serviceName.
  ?process profile:textDescription ?serviceTextDescription.
  ?agentService service:presents ?process.
  ?agentService actor:webURL ?web.
  FILTER regex(str(?paramVal), "Airport", "I").
  FILTER regex(str(?service), "_Service", "I").
}
```

Figure 34: SPAQL Query To filter by Output

SPARQL query to filter by Input based on generated OWL-S,

```
SPARQL query:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX process: <http://www.daml.org/services/owl-s/1.2/Process.owl#>
PREFIX profile: <http://www.daml.org/services/owl-s/1.2/Profile.owl#>
PREFIX expression: <http://www.daml.org/services/owl-s/1.2/generic/Expression.owl#>
PREFIX actor: <http://www.daml.org/services/owl-s/1.2/ActorDefault.owl#>
PREFIX service: <http://www.daml.org/services/owl-s/1.2/Service.owl#>
SELECT ?service ?serviceName ?serviceTextDescription
WHERE {
  ?agent rdf:type ?service .
  ?atomicprocess rdf:type process:AtomicProcess.
  ?process rdf:type process:Input.
  ?atomicprocess process:hasInput ?process.
  ?agent process:hasInput ?atomicprocess.
  ?process expression:refURI ?refUri.
  ?process process:parameterValue ?paramVal.
  ?agent profile:serviceName ?serviceName.
  ?agent profile:textDescription ?serviceTextDescription.
  ?agentService service:presents ?agent.
  ?agentService actor:webURL ?web.
  FILTER regex(str(?paramVal), "location", "i")
}
```

Figure 35: SPARQL Query To filter by Input

CHAPTER 4: EVALUATION PROCESS

4.1 EVALUATION SCOPE

The research scope of this project included two main phases. The first Phase is Retrieving semantics from Swagger and generating OWL-S ontology with mapping to domain Ontology resource. The second Phase is to Search for a service by querying the OWL-S Ontology.

The hypothesis of this project is using a dataset that contains several APIs and semantic descriptions fed to a system that will automatically map to predefined ontology resources. Then test the search engine whether it returns the most suitable service when a question is asked. Thereby identify the vulnerabilities present in the application.

The evaluation approach is Experiment based using publicly available APIs.

4.2 EVALUATION DATASET

For evaluating the semantics retrieval phase of this project 25 swagger files were used. Swagger files taken from a public repository called API guru and the swagger are publicly available for use. API guru has 1600+ swagger files of various domains and reliable.

Swagger files belong to multiple domains such as financial, Social media, Travel, and so on. Since the domain ontology, the prototype used was Schema.Org. Therefore, any domain is compatible with the solution. Each API includes CRUD endpoints.

The reason to choose multiple different domains to make sure that the proposed solution is compatible with any API that is developed in any programming language and developed by different organizations.

4.3 EVALUATION APPROACH

First Phase Evaluation

According to the Literature review done. Every semantic service search engine can be evaluated based on 4 criteria's:[15]

1. Compatibility of various service description formats and languages.
2. Total effort needed for configuration and it's usability.
3. User data privacy policy as services may contain highly confidential information.
4. Service performance is measured by accuracy and average query response time (AQRT) over given test questions.

The accuracy of a semantic retrieval is generally evaluated using retrieval measurements such as

1. Average precision (AP).
2. Macro-averaged precision at standard recall levels for binary relevance
3. Normalized discounted cumulative gain or Q-measure for graded relevance.

Second Phase Evaluation

To evaluate the accuracy of the search engine. We need to test the search engine with a set of questions and check whether it returns the correct service.

For better evaluation purposes most, existing implementations use the target ontology as Schema.org. For question evaluation benchmark uses questions provided by “the 2nd Open Challenge on Question Answering over Linked Data10” (QALD-2). Datasets consist of 100 natural language questions annotated with SPARQL queries and answers.

Gold Standard and Evaluation Metric

For each of the questions, a system-generated answer that is specified SPARQL query retrieves will be compared to the answers provided by the gold standard XML document. All answers are counted as correct answers as long as the answer list contains proper valid resources from ontology.

The measurements will compute precision, recall, and F-measure for every question (suggested by QALD-2):

Equation 2: QALD-2 Measurements

$$\begin{aligned}\text{Recall} &= \frac{\text{number of correct system answers}}{\text{number of gold standard answers}} \\ \text{Precision} &= \frac{\text{number of correct system answers}}{\text{number of system answers}} \\ \text{F-measure} &= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\end{aligned}$$

Computes the overall precision and recall taking the average mean of all single precision and recall values, as well as the overall F-measure. QALD-2 Challenge has the international recognition for Question Answering over Linked Data benchmark and is organized as part of the workshop Interacting with Linked Data (ILD) at the Extended Semantic Web Conference 2012.

“Freitas et al.” evaluate query expansion as an information retrieval task, where expansion candidates are ranked according to their relevance. The measure used for evaluation is Mean Reciprocal Rank (MRR) which measures the quality of the ranking by calculating the inverse rank of the best result.

In this project, this approach can implement in the Query engine. That is, apply MRR to the resulting ranked list of services given by the application. When a question is searched in the search engine it will return a set of REST services and sorted based on MRR. Which means the most suitable REST service returns on top of the list.

The Semantic Service Selection (S3) contest

“International contest series on semantic service selection(S3)”. The S3 contest is an annual forum for evaluating publicly available Semantic web service matchmakers using test collection. This evaluation framework has standard OWL-S and SWSDL service test collection to evaluate the retrieval performance of Semantic web service matchmakers. For OWL-S test collection is OWLS-TC.

Performance measurements are recall, precision, F1, and average response time. OWLS- Test collection has 32 domain-specific ontologies and 29 OWLS query files. So, when a matchmaker is evaluated it will first load the ontologies and then questions are raised. Matchmakers must match the question to most matching service. Better measurements matchmaker will win the contest.

4.4 EVALUATION RESULTS

The Evaluation for the search engine was measured by the accuracy of the search engine against a set of questions and the GOLD standard mentioned above. Those measurements were used to evaluate whether the resulting answer is most suitable for the question asked by the user.

When evaluating one service at a time was evaluated and taken down the measurements.

For example, Service called Lufthansa API. Firstly, went through the API documentation and created the Questions and expected Gold standard answers. Then ran the prototype tool against the swagger file. Checked whether any issues occur, or mismatch has happened. If there were mismatches correct and rerun the tool. Check the generate OWL-'s ontologies created. Make sure all the components are mapped in OWL-S.

Then run the search engine against OWL-S that was created. Searched the Question initially created and noted the search results. Results were then compared with expected answers and noted the measurements.

After all, 25 swagger files were tested. The number of correct answers and the number of system answers were noted and then calculated F-measurement. There are a set of criteria (mentioned above) to determine whether the semantic description approach is valid.

1. Usability and configurations- This proposed solution is easily configured as it needs only domain ontology and swagger file for processing. Usability wise the generated OWL-S ontologies can be visualized and self-explanatory.

2. Security and Privacy – This proposed solution can execute in a private and local environment as it doesn't need any cloud infrastructure or any public repository access. This tool protects the API provider's privacy by not letting any data leak.

Table 1: Results of the web service testing using QALD-2 Measurements.

API Name	Domain	Number of gold standard answers	Number of system answers	Number of correct system answers	Recall	Precision	F-measure
afterbanks	financial	4	5	3	0.75	0.60	0.67
Amazon CloudSearch	Devops	5	4	4	0.80	1.00	0.89
BBC	Radio & Music Services	3	4	3	1.00	0.75	0.86
ebay - Buy Marketing	ecommerce	4	3	2	0.50	0.67	0.57
ExchangeRate-API	financial	4	5	4	1.00	0.80	0.89
Flickr	media	4	3	3	0.75	1.00	0.86
github	developer_tools	5	4	4	0.80	1.00	0.89
Healthcare	open_data	6	8	4	0.67	0.50	0.57
HSBC_UK	financial	3	5	2	0.67	0.40	0.50
Instagram	social media	4	5	4	1.00	0.80	0.89
Just Eat UK	ecommerce	5	5	5	1.00	1.00	1.00
Lyft	location	2	2	2	1.00	1.00	1.00
lufthansa	transport	4	4	4	1.00	1.00	1.00
medium	blog	5	4	4	0.80	1.00	0.89
microsoft	cloud	3	4	2	0.67	0.50	0.57
Open Data	financial	2	4	2	1.00	0.50	0.67
Slack	collaboration	5	5	4	0.80	0.80	0.80
Spotify	media	4	4	4	1.00	1.00	1.00
Trello	collaboration	3	2	2	0.67	1.00	0.80
WhatsApp	messaging	4	4	4	1.00	1.00	1.00
Zoom	telecom	6	5	3	0.50	0.60	0.55
Walmart	ecommerce	3	3	3	1.00	1.00	1.00
Vimeo	entertainment	5	6	5	1.00	0.83	0.91
uebermaps	location	6	6	6	1.00	1.00	1.00
Sonar Trading	financial	4	3	4	1.00	1.33	1.14

Analysis of the measurement collected, API that matchmaker was able to answer all the questions has got higher F-Measure.

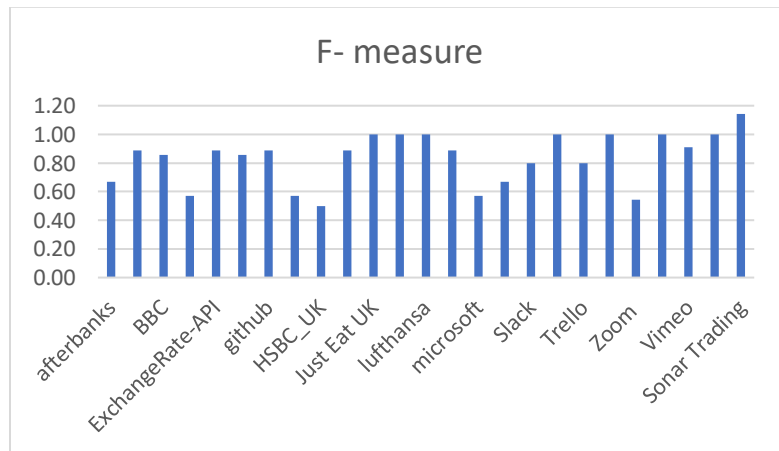


Figure 36: F measure on API

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

Web service discovery is especially important to a developer who wants to integrate third-party service to their solution. As we move with the world new opportunities open up for developers to build innovative solutions. Most developers face barriers when it comes to creating an API on complex business logic from the scratch. By having a service discovery tool, it can be minimized. Where the subject matter expert can provide an API that can be integrated. Tech giants extend their application features through API allowing developers to integrate and use for their solutions.

Open API is becoming immensely popular among API providers as it is a “good to have” aspect. From that simple document, it gives the visibility the API needs to the rest of the world developers.

Semantic web service needs to be introduced in the first place to make semantics readable/Accessible by computers. Data mining, Machine learning, and indexing algorithms can be executed on the semantics.

There is a saying where a picture speaks a thousand words. Likewise, API represented in OWL-S form helps developers to conceptually understand the functionality the API provides. Domain Ontology is a greater way for a developer to get an overall view of the domain. Having domain Ontology and OWL-S ontologies the developer can easily navigate through and understand.

Conclusion this research by the famous quote

“All our work, our whole life is a matter of semantics, because words are the tools with which we work, the material out of which laws are made, out of which the Constitution was written. Everything depends on our understanding of them. - Felix Frankfurter”

5.2 FUTURE WORK

As future work of this research, we can incorporate indexing to OWL-S service discovery as the number of services increases the ontology will proportionally increase. Therefore, querying the ontology will have longer throughput. If we can index the ontology services, it would increase the result response time.

Service engines can be improved to facilitate synonyms and Natural Language Questions. The search engine can have an algorithm that not only returns the same concept services but neighboring concepts along with a ranking.

There is new research where OWL-S is extended to add more features to the ontology for better representation of the service. A solution like this can be adapted to the new OWL-S extensions.

REFERENCES

1. Ruben Verborgh¹, Andreas Harth², Maria Maleshkova², Steffen Stadtmüller² Thomas Steiner³, Mohsen Taheriyani⁴, and Rik Van de Walle² “Semantic Description of rest apis”
2. Victor Saquicela, Luis. M. Vilches-Blázquez, and Óscar Corcho “Semantic Annotation of RESTful Services Using External Resources”
3. Sana Ben Abdallah Ben Laminea, Hajer Baazaoui Zghala, Michael Mrissab and Chirine Ghedira Guegan “An ontology-based approach for personalized RESTful Web service discovery”.
4. Jos´e Ignacio Fern´andez-Villamor, Carlos A. Iglesias, and Mercedes Garijo “A Framework for Goal-Oriented Discovery of Resources in the RESTful Architecture” *IEEE Transactions On Systems, Man, And Cybernetics: Systems*, VOL. 44, NO. 6, JUNE 2014
5. Amit P. Sheth, Jon Lathem and Karthik Gomadam” SA-REST: Semantically Interoperable and Easierto- Use Services and Mashups
6. T.Suganya and R.Rajmohan “Ontology Based Semantic Web Service Discovery and Composition Framework” *T.Suganya et al. / International Journal of Computer Science & Engineering Technology (IJCSET)ISSN*
7. Abdelhadi Belfadel, Emna Amdouni, Jannik Laval, Chantal Cherifi, Néjib Moalla. “Ontology-based Software Capability Container for RESTful APIs”. *9th IEEE International Conference on Intelligent Systems (IS 2018), Sep 2018, Madeira, Portugal.*
8. Markus Lanthaler, Michael Granitzer and Christian Gütl. “SEMANTIC WEB SERVICES: STATE OF THE ART” *IADIS International Conference on Internet Technologies & Society 2010*
9. Jacek Kopeck´y, Karthik Gomadam and Tomas Vitvar. “hRESTS: an HTML Microformat for Describing RESTfulWeb Services” *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*
10. Ji, Jiaxuan & Bu, Fenglin & Cai, Hongming & Wang, Junye. (2010). “Ontology Model for Semantic Web Service Matching”. *6377. 181-188. 10.1007/978-3-642-16167-4_24.*
11. Reihaneh Rabbany Khorasgani, Eleni Stroulia, Osmar R. Zaiane “Web Service Matching for RESTfulWeb Services”

12. Abdelhadi Belfadel, Emna Amdouni, Jannik Laval, Chantal Cherifi and Nejib Moalla
 “Ontology-based software capability container for RESTful APIs” *University of Lyon, University Lumiere Lyon*
13. H. Yoo, Y. Park and T. Lee, "Ontology based keyword dictionary server for semantic service discovery," *2013 IEEE Third International Conference on Consumer Electronics ; Berlin (ICCE-Berlin), Berlin, 2013, pp, 295-298. doi: 10.1109/ICCE-Berlin.2013.6698036*
14. M. Uschold and M. Gruninger.” *Ontologies: Principles, Methods and Applications, Knowledge Engineering Review*” *11(2): 93–136, 1996.*
15. Sukasom Chaiyakul, Kati Limapichat, Avani Dixit and Ekawit Nantajeewarawat “A Framework for Semantic Web Service Discovery and Planning” *Department of Information and Computer Technology*
16. Xie B in-hong, Zhang Ying-jun and Guo Yong-yi “A Web Service Matchmaker Based on Fuzzy Logic and OWL-S” *2010 International Conference on Computational Aspects of Social Networks*
17. Changjun Hu, Huayu L, Xiaoming Zhang “A Framework of Web Services Description and Discovery Based on OWL-S and Domain Ontology” *2008 IEEE Asia-Pacific Services Computing Conference*
18. Moonyoung Chung, Hyun Namgoong*, Kyung-Il Kim, Seungwoo Jung, HyeonSung Cho “Improved Matching Algorithm for Services described by OWL-S” *Electronics and Telecommunications Research Institute, University of Science & Technology*
19. Lin Zhang “OWL-S Based Web Service Discovery in Distributed System” *2014 IEEE Workshop on Electronics, Computer, and Application*