Using Tag based Semantic Annotation to Empower Client and REST Service Interaction

Cong Peng and Guohua Bai
Department of Creative Technologies, Blekinge Institute of Technology, Karlskrona, Sweden

Keywords: Web Service Description, Semantic Annotation, REST, OpenAPI, Service Discovery, Semantic Web Services.

Abstract: The utilization of Web services is becoming a human labor consuming work as the rapid growth of Web. The semantic annotated service description can support more automatic ways on tasks such as service discovery, invocation and composition. But the adoption of existed Semantic Web Services solutions is hindering by their overly complexity and high expertise demand. In this paper we propose a highly lightweight and non-intrusive method to enrich the REST Web service resources with semantic annotations to support a more autonomous Web service utilization and generic client service interaction. It is achieved by turning the service description into a semantic resource graph represented in RDF, with the added tag based semantic annotation and a small vocabulary. The method is implemented with the popular OpenAPI service description format, and illustrated by a simple use case example.

1 INTRODUCTION

The Web is growing exceedingly, either for human or for machines (Web APIs). It is, on the one hand, bringing an imaginative utilization of the services on the Web. On the other hand, it also brings about a problem that the utilization of Web services is becoming a tediously repetitive work, such as search for services required, read documents to know how to invoke, and program similar clients to utilize services. Those works currently are more of human labor-centered works other than intelligence-centered works, which they are supposed to be.

For enabling a more autonomous usage of Web services, it requires the client to have a certain level of capabilities for interpreting the resources, such as the understanding of what the resources are and the possible actions to the resources. And it should also reduce coupling between client and service. Therefore, a machine understandable service description is important to facilitate a more autonomous Web service utilization from discovery, invocation to composition.

However, there are very few semantic service descriptions available for the current Web services. The reasons are multi-fold. The existed semantic service description standards are lacking links to the actual Web service development technology stack. It requires extensive manual work to annotate or even recreate the descriptions of existed Web services. Besides, the attempt to achieve fully semantic understandable at once made those solutions (Kopecký et al., 2007; Vitvar et al., 2008) complicated and highly expertise requisite, which caused the reluctant of application.

In this paper, we propose a method called SemREST (Semantic RESource Tagging) to add semantic annotations to REST service description in a simple manner to make the client service interaction more generic and autonomous. To mitigate the *semaphobia* (Lanthaler and Gütl, 2011), our approach aims to avoid much knowledge requirement on Semantic Web by adding tag based semantic annotations to the service description, which will be presented with the widely used OpenAPI specification[1] in a non-intrusive way. An example case is presented to demonstrate how this approach could work.

2 RELATED WORK AND PROBLEM ANALYSIS

There have been extensive works done on both syntactic description and semantic description to Web services. Here we only discuss those works that have strong influence and are considered most relevant to the work in this paper.

[1] https://swagger.io/specification/

Peng, C. and Bai, G.
Using Tag based Semantic Annotation to Empower Client and REST Service Interaction.
DOI: 10.5220/0006682500640071
In *Proceedings of the 3rd International Conference on Complexity, Future Information Systems and Risk (COMPLEXIS 2018)*, pages 64-71
ISBN: 978-989-758-297-4

The Semantic Annotations for WSDL and XML Schema (SAWSDL) aims to add semantic annotations to a WSDL (Web Services Description Language) document's input/output message, interfaces and operations (Kopecký et al., 2007). An annotation mechanism is also defined in SAWSDL to specify the data mappting between XML Schema types and an ontology. SAWSDL is regarded as a lightweight approach to annotate Web services described in WSDL, and recommended by the World Wide Web Consortium (W3C) in 2007.

As SAWSDL only provides a way to attach semantic annotation in WSDL documents but without the specification and standardization of the concrete service semantics, WSMO-Lite was then proposed to model service semantics within the SAWSDL framework (Roman et al., 2015). WSMO-Lite is a service ontology inspired by the WSMO (Web Service Modeling Ontology) ontology that uses SAWSDL as annotation mechanism. It defines a a lightweight set of semantic service descriptions in RDFS (Resource Description Framework Schema) to annotate WSDL elements (Vitvar et al., 2008). WSMO is a top-down conceptual model for semantic Web services, whereas the WSMO-Lite is a bottom-up model to provide a lightweight ontology (Dieter Fensel et al., 2010).

In addition, in order to combine the common RESTful Web services and WSDL-based services into one semantic Web service framework, Roman et al. added hRESTS (HTML format for describing RESTful Services) and MicroWSMO, the two HTML microformats mirror WSDL and SAWSDL, to work together with WSMO-Lite (Roman et al., 2015). In which the WSDL and hRESTS work as the service description layer, SAWSDL and MicroWSMO work as the semantic annotation layer points to concepts defined in WSMO-Lite.

RESTdesc is RESTful Web services semantic description method with an emphasis on hypermedia APIs. It describes service's request preconditions, postconditions and quantifiers in RDF/Notation3 to enable a more automatic service composition (Verborgh et al., 2011).

The Hydra framework is a set of technologies tries to make semantic enriched Hypermedia API for Web services to enable generic clients to consume services [2]. It is based on the Hydra Core Vocabulary, which is a lightweight vocabulary that defines some fundamental concepts commonly used by Web APIs (Lanthaler and Guetl, 2013).

These solutions enabled the semantic annotations to service descriptions, but they suffer a problem that denoted by Lanthaler et al. as *Semaphobia* (Lanthaler and Gütl, 2011). It requires too much knowledge on Semantic Web to annotate with these solutions, which causes the reluctant of application.

Another problem with these solutions is user unfriendly. Though the semantic service description is meant to be understood by machines, it requires human to create, with handy support tools and human readable formats. This is why the description standards like OpenAPI Specification, API Blueprint and RESTful API Modeling Language (RAML) are becoming popular these years. OpenAPI Specification (widely known as Swagger) is a definition standard proposed by Open API Initiative to describe RESTful Web APIs [3]. OpenAPI is the mostly used syntactic description format currently, thanks to its easy to use editor and multiple languages supported code generation tools.

However, due to lack of machine understandable semantics, these syntactic API descriptions are still inefficient for service consumers to discover the specific resources within the service in a more autonomous manner with a generic client. The work done by Lucky et al. presented an approach to add semantic annotation to the response data schema in an OpenAPI description (Lucky et al., 2016). However, it did not mention how can the service client be improved by the semantic annotation to a more automatic way. And it is more an approach to empower static service composition.

In this paper we try to use a tag based approach to add semantic annotation to the popular and easy to use OpenAPI Specification, to achieve a more generic and autonomous client service interaction in a extremely simple manner, at the same time take advantage of human readable description format and supporting tools, and keep its simple to start, well-integrated to the existing web service work/development flow, easily achievable with minimal intrusion, to overcome the *Semaphobia*.

3 SEMANTIC RESOURCE TAGGING WITH OpenAPI

3.1 Semantic Tag Annotated OpenAPI

As the mostly used API description standard with a tooling ecosystem for both API developing and consuming, we firstly implement the SemREST with the OpenAPI Specification. Here in Listing 1 is an example OpenAPI description fraction of the activities resource in YAML format (which is a path object in

[2] http://www.hydra-cg.com

[3] https://swagger.io

```
paths:
  # Resource1-2
  /1/user/-/activities/calories/date/{base-date}/
  ↪ {end-date}.json:
    get:
      summary: Get Activity Time Series
      description: Returns time series data for activities
      ↪ calories resource.
      parameters:
        - name: base-date
          in: path
          description: The range start date.
        - name: end-date
          in: path
          description: The end date of the range.
      responses:
        200:
          description: A successful request.
          schema:
            $ref: '#/definitions/CaloriesTS'
```

Listing 1: Service Description Fraction of a Fitbit Resource.

the OpenAPI description) from Fitbit service API when the time this paper is writing, we modified a little for brevity. Fitbit is a health and fitness service that offers wearable devices to track activity data and hosts Web service to provide access to data[4]. And this Fitbit service description will be used as an example through out this paper.

As analyzed in the previous section, the current OpenAPI service description lacks machine understandable semantics that can provide the meaning of what the resource represents. So we propose to add tag based annotation to the resource. For the sake of not violating the standard OpenAPI specification, the semantic tag annotation will be added through specification's built-in optional field, tags. A list of tags to each operation object in OpenAPI is one kind of additional metadata, it is for the similar purpose as our semantic tag.

A semantic tag comprises 2 parts, which are a prefix and a suffix joint by a colon. The prefix is an abbreviation of the used open vocabulary's Internationalized Resource Identifier (IRI) (M. Duerst and M. Suignard. IETF, 2005), the suffix is the term in the used vocabulary to indicate the semantic meaning, which is case sensitive aligned to those vocabularies. The two parts as a whole is equivalent to the IRI of the term. This way of expressing semantic tag follows the abbreviation mechanism of OWL (Web Ontology Language) (W3C, 2012).

The Listing 2 shows the semantic tag annotated version of the resource shown in Listing 1. The only difference, which is highlighted, is the additional tags

[4]https://www.fitbit.com

```
paths:
  # Resource1-2
  /1/user/-/activities/calories/date/{base-date}/
  ↪ {end-date}.json:
    get:
      summary: Get Activity Time Series
      description: Returns time series data in the
      ↪ specified range for a given resource.
      tags:
        - schema:calories
        - schema:activities
        - schema:ConsumeAction
      parameters:
        ...
      responses:
        ...
```

Listing 2: Annotated Service Description Fraction of a Fitbit Resource.

```
tags:
  - name: schema
    description: the schema.org vocab
    externalDocs:
      url: http://schema.org
      description: uri of the schema.org vocab
```

Listing 3: Used Vocabulary Stated as OpenAPI Tags Object.

list for a resource's description. As in the OpenAPI specification, all the used tags could be declared in a `tags` object under the `OpenAPI` root object, so as these semantic tags. The second additional information will be added to the plain OpenAPI description is the IRI of each tag. The IRIs will be indicated by the `url` field of each tag object's `externalDocs` object under the root `tags` object. But for the simplicity, we can just state the vocabularies used in the root `tags` object (as shown in Listing 3), which is compatible.

For serving the annotated service description to the client, an `OPTIONS` method under the service's root path `/` will be added to each service to respond the service description (as shown in Listing 4). As a method for describing the communication options for a target resource in HTTP/1.1 specification (R. Fielding, J. Gettys, J. Mogul et al., 1999), it makes `OPTIONS` method a good way for discovering related resources for the target resource (Steiner and Algermissen, 2011). Since a service description contains all the resources, which can be regarded as all the related resources to the root resource, it is natural to use the `OPTIONS` method to retrieve the service description, as the same in (Lucky et al., 2016).

When the annotations are added, now we can generate the semantic description of the resources to RDF based on it and together with the OpenAPI

```
paths:
  # Resource1-1
  /:
    options:
      summary: Returns service description
      produces:
        - application/json
        - application/ld+json
      responses:
        ...
```

Listing 4: OPTIONS Method of Root Resource for Serving Annotated Service Description.

model and a small SemREST mapping vocabulary, which simply contains some mapping entities to be expressed in RDF. Table 1 only lists the entities those are necessary to generate the resource graph, a more complete vocabulary can refer to the work done by Musyaffa et al. (Musyaffa et al., 2016).

The generated RDF depicts the resource in the service is presented Figure 1 as a graph. To be noted that in Figure 1, the resource IRI is manually replaced by a human readable name for readability. The resource IRI in actual fact is identified by calculating MD5 hash of the resource's OpenAPI description path string. With the semantic tags indicating what the resource represents, now we can use the semantic tags to enable service client for the discovery of required resources in the service by semantic inference. For example in the case of different vocabularies are used among different services and client but indicating the same or similar entity, the client can infer all the resources annotated in different different vocabularies with the help of a simple ontology.

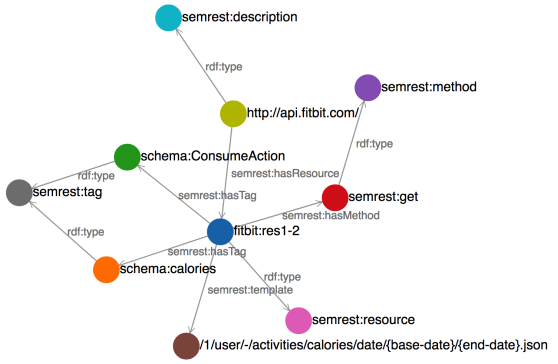

Figure 1: Resource Graph of a Fitbit Resource.

3.2 SemREST Framework

Since the server cannot predict what the client will require for the next step, so many computations of resource discovery will be done on the client side (resource URL query and inferring). Although it can be processed also on the service or third party service registry for different purposes, we only demonstrate the client side resource discovery in this paper.

The architecture of framework is shown in Figure 2, include the extended components to the official OpenAPI framework, the server side of service, and the client side with a semantic module. As we propose to add tag based semantic annotations to the OpenAPI service description, here we explain how the process is for generating the semantic annotated service description.

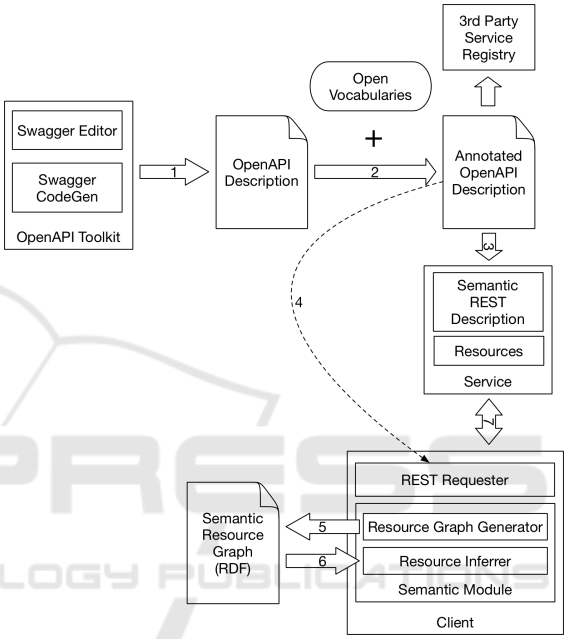

Figure 2: Architecture of SemREST.

When the service developer completed the normal OpenAPI description, the semantic tag annotations could be added. The annotated OpenAPI service description could be registered to third party service registries for enhanced semantic service discovery. Then the annotated OpenAPI description will be served at the service's root path via `OPTIONS` method in certain data exchange format, either JSON or YAML.

The client contains an additional semantic module, which is responsible for parsing service description into resource graph, and inferring for required resources. Firstly the the client need to register what services it connects to by setting the service root URL. The registration of services is a one time job, and in the work of this stage, we leave aside the security and authorization, but it is achievable with certain OpenAPI objects. The client can retrieve the annotated service description under the services' root path via `OPTIONS` method after registration meanwhile set-

Table 1: Named Entities Used in SemREST.

Name/Abbreviation	Type	Description
semrest:tag	Class	A semantic tag that annotates a resource
semrest:method	Class	A HTTP method that a resource supports
semrest:options	Class	A HTTP OPTIONS method, a subclass of semrest:method
semrest:get	Class	A HTTP GET method, a subclass of semrest:method
semrest:paths	Class	The paths of all resources in a OpenAPI description file
semrest:template	Class	An URL of a path, string value of a path object
semrest:hasResource	Property	To state a description of service contains which resources
semrest:hasTag	Property	To state the annotated semantic tags of a resource
semrest:hasMethod	Property	To state the supported HTTP method of a resource

ting required resources by semantic tags.

For generating a semantic resource graph, the resource graph generator component in the semantic module will parse the annotated description together with the small SemREST vocabulary into a RDF document, which could be in any RDF compatible format such as RDF/XML, JSON-LD or Turtle. The generated resource graph, it can be used multiple times and can be renewed when the service description changed by retrieving a new version. When the semantic resource graph is generated, the semantic inferrer component can then query in the graph which resources to request by the set semantic tags. The semantic module of client can suggest which tags to use under the same vocabulary as the services used, which can reduce some inference work. When the required resources are found, the metadata (URLs, parameters, etc.) will be passed to the request module to send the requests to the services.

4 FEASIBILITY OF SemREST

4.1 Tooling and Implementation

Our current proof-of-concept prototype is a very simple implementation supported by openly available tools. The creation of the normal OpenAPI service description is completed by the official Swagger online editor[5] in YAML format. For the semantic tag annotation, the vocabularies are searched via the Linked Open Vocabularies[6], and the adding of semantic tags is completed by Swagger editor as well. As the process of creation and editing of service description is only a text editing work, it could be achieved by any text editor.

We implemented both the imitation services and the client in Python. The imitation services, which implemented with Flask framework[7], imitate how the actual service receive requests and return responses with fake data, and only include the resources that appear in the example case in the next section. The client is comprised by 2 modules, a REST service requester and a semantic module. In our prototype, we simply integratively use the Python client generated by the Swagger CodeGen framework to handle the service requests. The semantic module is built upon the RDFLib[8] for a quicker implementation. Otherwise, the Apache Jena[9] is a more powerful Semantic Web framework that enables more functionalities like inference API. Currently, we use the JSON format of the annotated description file in the prototype, since it is easier to transfer the description into a JSON-LD document (Manu Sporny et al., 2014), which is a concrete RDF syntax, by add the vocabularies as a context object.

4.2 Example Case

Here we will use an example case to demonstrate how the proposed SemREST can achieve a more autonomous and generic client service interaction in a simple and non-intrusive way. The example case involves the forementioned Fitbit service and Human API service. Human API is a health data platform that aggregates clinical data from comprehensive sources such as Electronic Health Record, health devices and wearable devices, and makes data accessible as RESTful API[10].

The scenario is set as, a service consumer Alice uses Fitbit service to record her activity data include calories consumption, and uses a blood pressure monitor that the data of which can be consumed from Human API. Alice wants her health agent client with a semantic module can gather her data from the 2 ser-

[5]https://swagger.io/swagger-editor/
[6]http://lov.okfn.org/dataset/lov/
[7]http://flask.pocoo.org/
[8]http://rdflib.readthedocs.io
[9]https://jena.apache.org/
[10]https://www.humanapi.co/

vices to have a comprehensive view of the health status.

Although both Fitbit and Human API provide Web APIs with documentations, the data are stored and need to be retrieved separately, Alice still need to read the two documents and develop two different client programs in order to gather her data from the two services. If both Fitbit and Human API can provide their service description with semantic tag annotations as the approach proposed in this paper, then it will reduce much repeated works. We will present how the process will be for the two service providers and Alice's client.

Since it involves the service providers to serve their services with description, and clients to interact with the services, so this example case consists of 2 stage processes that are Semantic Service Description Generation and Serving, and Client Service Interaction.

4.2.1 Description Generation and Serving

Firstly we will create the Fitbit SemREST service description. Besides the activities' calories consumption resource shown in Section 3.1, the resource of the foods' calories intake data is also added to this example. Since the original OpenAPI service description of Fitbit is already provided for API consumer (as many other web service providers do), here we can directly add semantic tag annotations to the resources.

As we described in Section 3.1, the resource1-2 is annotated from Listing 1 into Listing 2. The only place we changed is the *tags* field. We add tag `schema:calories`[11] to resource1-2. Consider the calories data for activities can be regarded as consuming, so we can add `schema:ConsumeAction`[12] to resource 1-2. For the resource of the foods' calories intake data, which we call it resource1-3, the annotated service description is shown in Listing 5. We add tag `schema:calories` to it as well, and obviously tag `schema:FoodEvent`[13] can be added to resource 1-3. Combine these 2 annotated resources with Listing 3, in which described the used `schema` vocabulary stated in tags object, and Listing 4, in which described the root resource path for serving service description, we get the annotated SemREST description fraction to be used in this example case.

Now we will create the Human API SemREST service description. Since Human API has not provided API description in OpenAPI specification yet, so we need to compose its OpenAPI description imita-

[11]http://schema.org/calories

[12]http://schema.org/ConsumeAction

[13]http://schema.org/FoodEvent

```
paths:
  # Resource 1-3
  /1/user/-/foods/log/caloriesIn/date/{baseDate}/{endDate}:
    get:
      tags:
        - schema:calories
        - schema:FoodEvent
      responses:
        ...
```

Listing 5: Annotated Service Description Fraction of a Fitbit Resource.

```
tags:
  - name: m3lite
    description: The M3-lite Taxonomy
    externalDocs:
      url: http://purl.org/iot/vocab/m3-lite#
      description: uri of the M3-lite Taxonomy
paths:
  # Resource2-1
  /:
    options:
      summary: Returns service description
      produces:
        - application/json
        - application/ld+json
      responses:
        ...
  # Resource 2-2
  /v1/human/blood_pressure:
    get:
      summary: Returns the latest blood pressure reading
      description: The latest blood pressure reading
      tags:
        - m3lite:BloodPressure
      ...
```

Listing 6: Annotated Service Description of Human API's Root and BloodPressure Resource.

ted from their API documentation first. The imitated OpenAPI description is shown in Listing 6, in which the `blood_pressure` resource 2-2 is given the semantic tag `m3lite:BloodPressure`[14] under The M3-lite Taxonomy (Gyrard et al., 2016). As the same in Fitbit description, the service description is served under the path `/` via `OPTIONS` method.

4.2.2 Resources Look up and Request

Now we have the two services serving their resources and service descriptions with semantic tag annotation, Alice's health agent client can start to work. Alice will firstly tell her client where the services are serving by setting the root URLs of the 2 services, and then set what resour-

[14]http://purl.org/iot/vocab/m3-lite#BloodPressure

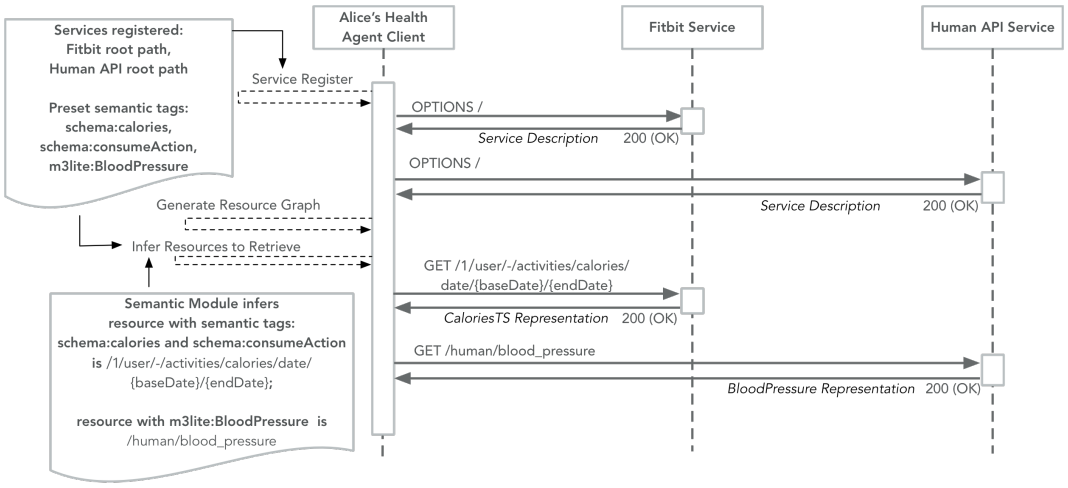

Figure 3: Interaction Sequence of the Example Case.

ces she want to retrieve by setting semantic tags `schema:calories` and `schema:ConsumeAction` to the calories data she wants for her activities, and tag `m3lite:BloodPressure` for her blood pressure data she wants. Then the client starts to interact with the services. It will firstly send `OPTIONS` request to the two services' root path to get their service descriptions. Figure 3 depicts the client service interaction process of the example case.

Once the client get the service descriptions, the semantic module parses all the resources into a combined resource graph in RDF, which is shown in Figure 4 (the graph is simplified for brevity). The semantic module can then reason about required resources by the semantic tags set by Alice. So here the resource templates of resource 1-2 and resource 2-2 are found by `schema:calories` + `schema:ConsumeAction` and `m3lite:BloodPressure`, respectively. Each resource's path, parameters and other metadata in its OpenAPI description will then be determined by resource templates and passed to the request module. For the resource 1-1, the client requires `baseDate` and `endDate` as parameters to retrieve the representation form Fitbit service. With all the metadata and parameters of a service request being available, the actual HTTP request could be achieved in various ways. In this example, we use the automatically generated service client by the Swagger CodeGen framework to send the requests.

After Alice's health agent client get the calories of activities data from Fitbit and blood pressure data from Human API, it can then utilize them more comprehensively, simply like a graph to present the trend and correlation between calories consumption and blood pressure.

The interactions of client between Fitbit service and HumanAPI are achieved by the same client that supported by the tag based semantic annotation in a very simple way, without breaking the normal web service interaction process.

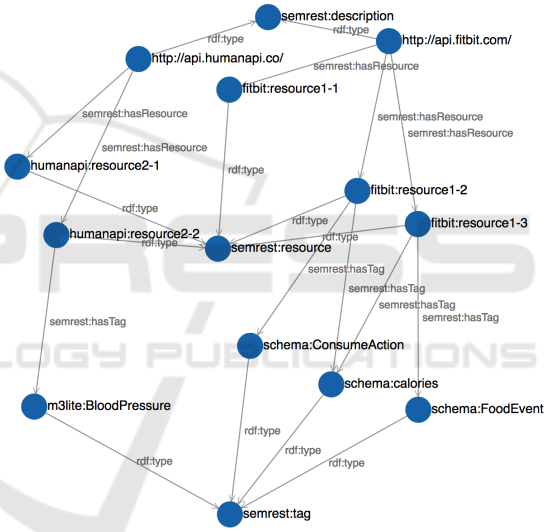

Figure 4: Combined Resource Graph of 2 Services.

5 CONCLUSION

In this paper, we proposed SemREST, an extremely lightweight and non-intrusive method, to enable semantics to resources of REST Web services. With this initial work, we identify a simple and fully compatible way of adding tag based semantic annotations to the widely used OpenAPI service description format. The small SemREST vocabulary we specified

together with the tag annotation mechanism provide a bridge from the syntactic OpenAPI description to the semantic resource graph, which enables a more generic and automatic client service interaction that can reduce many repeated human labor consuming work. The method we proposed also provide the potential for service discovery and composition, as the resources of the annotated services are semantically linked. The semantic resource graph could be generated and host on service side or third party service registry for the clients to query required resources.

However, in this initial work, we only realized the GET method for resources to make it as a data aggregation agent for different sources. And we only shown some naive cases to demonstrate this method, it leaves the authorization and security issues out for the sake of briefness.

For the future work, we plan to validate this method with more different services and realize other HTTP methods like POST and PUT, and evaluate the usability by performing survey with Web service providers and consumers. For the invoking and composition of services, it could be achieved in 2 difference approaches, one is the code automatic generation like OpenAPI framework provided, the other is the semantic way like what Hydra (Lanthaler and Guetl, 2013) is trying to do. We will try to explore how our method can integrate with approach like Hydra.

REFERENCES

Dieter Fensel, Florian Fischer, Jacek Kopecký, Reto Krummenacher, Dave Lambert, and Tomas Vitvar (2010). WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. https://www.w3.org/Submission/WSMO-Lite/. Accessed: 2017-09-19.

Gyrard, A., Gomez, D., Bajaj, G., Lanza, J., Sanchez, L., Agarwal, R., and Elsaleh, T. (2016). The M3-lite Taxonomy. http://ontology.fiesta-iot.eu/ontologyDocs/fiesta-iot/doc. Accessed: 2017-09-19.

Kopecký, J., Vitvar, T., Bournez, C., and Farrell, J. (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11(6):60–67.

Lanthaler, M. and Guetl, C. (2013). Hydra: A Vocabulary for Hypermedia-driven Web APIs. In *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW2013) at the 22nd International World Wide Web Conference*. CEUR.

Lanthaler, M. and Gütl, C. (2011). A semantic description language for RESTful Data Services to combat Semaphobia. In *5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011)*, pages 47–53.

Lucky, M. N., Cremaschi, M., Lodigiani, B., Menolascina, A., and De Paoli, F. (2016). Enriching API Descriptions by Adding API Profiles Through Semantic Annotation. In *International Conference on Service-Oriented Computing*, pages 780–794. Springer, Cham.

M. Duerst and M. Suignard. IETF (2005). RFC 3987: Internationalized Resource Identifiers (IRIs). http://www.ietf.org/rfc/rfc3987.txt. Accessed: 2017-08-15.

Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström (2014). JSON-LD 1.0. https://www.w3.org/TR/json-ld/. Accessed: 2017-08-08.

Musyaffa, F. A., Halilaj, L., Siebes, R., Orlandi, F., and Auer, S. (2016). Minimally Invasive Semantification of Light Weight Service Descriptions. In *2016 IEEE International Conference on Web Services (ICWS)*, pages 672–677. IEEE.

R. Fielding, J. Gettys, J. Mogul, H. F., L. Masinter, P. Leach, and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. http://tools.ietf.org/html/rfc2616.

Roman, D., Kopecký, J., Vitvar, T., Domingue, J., and Fensel, D. (2015). WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs. *Web Semantics: Science, Services and Agents on the World Wide Web*, 31:39–58.

Steiner, T. and Algermissen, J. (2011). Fulfilling the hypermedia constraint via HTTP OPTIONS, the HTTP vocabulary in RDF, and link headers. In *Proceedings of the Second International Workshop on RESTful Design - WS-REST '11*, page 11, New York, New York, USA. ACM Press.

Verborgh, R., Steiner, T., Van Deursen, D., De Roo, J., de Walle, R., and Gabarró Vallés, J. (2011). Description and Interaction of RESTful Services for Automatic Discovery and Execution. *Proceedings of the FTRA 2011 International Workshop on Advanced Future Multimedia Services*.

Vitvar, T., Kopecký, J., Viskova, J., and Fensel, D. (2008). WSMO-lite annotations for web services. In *ESWC 2008: The Semantic Web: Research and Applications*, pages 674–689, Berlin, Heidelberg. Springer Berlin Heidelberg.

W3C (2012). OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#IRIs. Accessed: 2017-09-08.