SOAS 3.0: Semantically Enriched OpenAPI 3.0 Descriptions and Ontology for REST Services

Nikolaos Mainas  
School of Electrical and Computer Engineering  
Technical University of Crete (TUC)  
Chania, Crete, Greece  
nmainas@intelligence.tuc.gr

Euripides G.M. Petrakis  
School of Electrical and Computer Engineering  
Technical University of Crete (TUC)  
Chania, Crete, Greece  
petrakis@intelligence.tuc.gr

Abstract—OpenAPI Specification (OAS) is a description format for REST APIs. Building-upon the latest version 3.0 of the specification, this work analyses the reasons that cause ambiguities in service descriptions. Taking advantage of the extension features foreseen in OAS, our approach suggests that OAS properties must be semantically annotated and shows how service descriptions can be instantiated to an ontology.

Index Terms—OpenAPI, REST, Hydra, SHACL, Web service, service description, ontology

I. INTRODUCTION

Web services need to be described in a way that eliminates ambiguities and provides descriptions which are both uniquely defined and machine readable (e.g. by other services). OAS\(^1\) is a simple yet powerful framework for the description of REST APIs. OAS 3.0 is the first major update of the specification since 2015. OAS 3.0 features a more elaborate (yet simple) structure and format than its predecessor OAS 2.0. However, the meaning of OAS entities is sometimes vague.

In previous work [1] we showed how OAS 2.0 descriptions are disambiguated and mapped to an ontology. Following a similar approach, Semantic OAS 3.0 (SOAS 3.0) suggests that OpenAPI 3.0 service descriptions are semantically annotated by associating OpenAPI entities to entities of a domain ontology. At the same time, it is plausible to transform SOAS 3.0 descriptions to ontologies. This would enable application of query languages (e.g. SPARQL) for service discovery and of reasoning tools for detecting inconsistencies in service descriptions. SOAS 3.0 ontology incorporates features of Hydra [2] model for modeling service operations along with models not foreseen in Hydra (e.g. security features, header, constraints). Classes together with constraints on class properties are described using SHACL\(^2\) allowing service descriptions to be validated against conditions defined in SOAS 3.0 model.

The SOAS 3.0 approach is discussed in Sec. II. The instantiation of service descriptions to the SOAS 3.0 ontology is discussed in Sec. III, followed by issues for future work in Sec. IV.

II. SEMANTIC OPENAPI 3.0

Fig. 1 illustrates the structure of an OAS 3.0 service description\(^3\). The Info object provides non-functional information such as the name of the service, service provider and terms of the service. The Servers object provides information on where the API’s servers are located (i.e. multiple servers can be defined). The Security object describes the security schemes that the service uses for authentication. The specification offers support for basic HTTP authentication, API keys, OAuth2\(^4\) common flows and OpenID Connect. The Paths object contains the relative paths for the service endpoints. Each Path item describes the available operations based on HTTP methods.

The Components object holds a set of reusable objects which can be responses, parameters, schemas, request bodies and more. The Schemas object defines data structures that are used to describe the request and response messages. A Schema object can be a primitive (string, integer), an array or a model. The specification introduces also properties supporting model composition and polymorphism. The Responses object describes the expected responses of an operation, by mapping them to a specific HTTP status code. A response object defines the message content, as well as HTTP headers that a response may contain. The Parameters object describes parameters that

\(^{1}\)https://www.openapis.org  
\(^{2}\)https://www.w3.org/TR/shacl/  
\(^{3}\)https://blog.readme.io/an-example-filled-guide-to-swagger-3-2/  
\(^{4}\)https://oauth.net/2/
operations use. The specification, categorizes parameters into five types:

- **Path parameters** are used in cases where the parameter values are part of operation’s path.
- **Query parameters** are appended to the url when sending a request.
- **Header parameters** define additional custom headers that may be sent in a request.
- **Cookie parameters** are passed in the Cookie header.

### Table I

<table>
<thead>
<tr>
<th>Property</th>
<th>Applies to</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-refersTo</td>
<td>Schema Object</td>
<td>The concept in a semantic model that describes an OAS element.</td>
</tr>
<tr>
<td>x-kindOf</td>
<td>Schema Object</td>
<td>A specialization between an OAS element and a concept in a semantic model.</td>
</tr>
<tr>
<td>x-mapsTo</td>
<td>Schema Object</td>
<td>An OAS element which is semantically similar with another OAS element.</td>
</tr>
<tr>
<td>x-collectionOn</td>
<td>Schema Object</td>
<td>A model describes a collection over a specific property.</td>
</tr>
<tr>
<td>x-onResource</td>
<td>Tag Object</td>
<td>The Tag object refers to a resource in a semantic model.</td>
</tr>
<tr>
<td>x-operationType</td>
<td>Operation Object</td>
<td>Clarifies the type of operation.</td>
</tr>
</tbody>
</table>

SOAS 3.0 introduces extra properties to annotate existing OAS properties which are ambiguous. Table I summarizes the extension properties, their scope and their meaning. The `x-refersTo` extension property specifies the association between an OAS element and a concept in a semantic model. Listing 1 shows how `x-refersTo` is used to semantically annotate a `Person` model and its properties: it associates the model with a specific group of people (e.g. teenagers), `x-kindOf` extension property is used instead to denote that the model is a subclass of the referred semantic concept. The `x-mapsTo` is used to define `Schema` object elements that share the same semantics.

In Listing 1, `x-mapsTo` property is used to dictate that query parameter `surname` refers to “lastname” property of `Person` Schema object.

```json
parameters:
  surnameQuery:
    name: surname
    in: query
    description: Person’s last name for filtering
    required: true
    schema:
      type: string
      x-mapsTo: ‘#/components/schemas/Person.lastname’

schemas:
  Person: # A Person model
    type: object
    x-refersTo: http://schema.org/Person
    properties:
      firstName:
        type: string
        x-refersTo: http://schema.org/givenName
      lastName:
        type: string
        x-refersTo: http://schema.org/familyName
```

The `x-collectionOn` extension property is used to indicate that a model in `Schemas` object is actually a collection. Typically, a collection (or a list) of resources in OAS 3.0 is described using the `array` type. However, it is very common a collection’s definition to be encapsulated within an `object` type with additional properties. Then, `x-collectionOn` property is used to denote the data types of the objects of the collection. Listing 2 defines a model as a collection of `Pet` objects (`totalItems` property denotes population).

```json
List the 2. Model definition representing a collection

schemas:
  PetCollection: # A Pet Collection definition
    x-collectionOn: pets
    type: object
    properties:
      pets:
        type: array
        items:
          $ref: ‘#/components/schemas/Pet’
      totalItems:
        type: integer
```

The `x-onResource` extension property is used in `Tag Objects` to specify the resource that a tag refers. In OAS 3.0, tags are used to group operations either by resources or any other qualifier. If the tag is used to group operations by resources, a human may recognize that the referred resource is described by a `Schema` object in `Schemas` but, a machine cannot. The `x-onResource` property is used to associate the tag with a `Schema` object that describes a specific resource. In Listing 3, `x-onResource` property is assigned on a `pet` tag that provides information regarding the operations that are available for `Pet` model in `Schemas` object.

```json
List the 3. Excerpt from Swagger Petstore OAS service description
tags:
  name: pet
description: Everything about your Pets
externalDocs:
  description: Find out more
  url: ’http://swagger.io’
x-onResource: ‘#/components/schemas/Pet’
paths:
  /pet/findByStatus:
    get:
      x-operationType: ’http://schema.org/SearchAction’
tags:
  - pet

summary: Finds Pets by status
description: Multiple status values can be provided
parameters:
  - $ref: ‘#/components/parameters/statusQuery’
responses:
  ’200’:
    description: successful operation
content:
Finally, `x-operationType` extension property is used to specify the type of an `Operation` object. A request is characterized by the HTTP method it applies. However, the semantics of the HTTP methods are too generic and may have a more specific meaning. In Listing 3, this property is used to clarify that a GET request on path `/pet/findByStatus` is a search operation on pets based on their status. The property of the value is a URL pointing to the concept that semantically describes the operation type. The `Action` type of the Schema.org vocabulary provides a detailed hierarchy of `Action` sub-types that can be used by the property.

III. OpenAPI 3.0 Ontology

OpenAPI ontology in Fig. 2 captures all information specified by SOAS 3.0 description. Properties of classes are mapped to classes as well. At the heart of the ontology is Hydra Core Vocabulary, enhanced with additional models in order to capture all information provided by SOAS (i.e. security, headers, constraints).

![Fig. 2. SOAS 3.0 ontology](image)

In accordance to OAS 3.0 structure, `Document` provides general information (`Info` class) regarding the service; it specifies service paths, the entities and the security schemes that it supports. `Path` class represents (relative) service paths (pathName property). `Operation` class provides information for sending HTTP requests to the service as well as the HTTP responses. Responses are further described by `Response` class, specifying the status code and the data returned. The entire range of HTTP response values is represented. Class `Operation` refers to a security scheme in `SecurityRequirement` class.

![Fig. 3. SOAS 3.0 security class](image)

Listing 4 illustrates how an OAS `Path` item and `Operation` are defined in the ontology using the example of Listing 3. Class `Security` defines the security schemes that the specification supports. Class `Operation` refers to a security scheme using class `SecurityRequirement`, which in the case of the OAuth2 security scheme represents the scopes of the operation. Operation `path2_op1` refers to a `SecurityRequirement` individual, specifying an OAuth2 security scheme (i.e. `petstore_oauth` individual) and the corresponding scope (i.e. `read_pets` and `write_pets` individuals). Individual `path2_op1` is also considered to be an individual of `SearchAction` type defined in Schema.org vocabulary (i.e. as defined by the `x-operationType` extension property).

```json
application/json:
  schema:
    $ref: "#/components/schemas/Pets"
  '400':
    description: Invalid status value
  security:
    - petstore_auth:
      - 'write:pets'
      - 'read:pets'

Finally, `x-operationType` extension property is used to specify the type of an `Operation` object. A request is characterized by the HTTP method it applies. However, the semantics of the HTTP methods are too generic and may have a more specific meaning. In Listing 3, this property is used to clarify that a GET request on path `/pet/findByStatus` is a search operation on pets based on their status. The value of the property is a URL pointing to the concept that semantically describes the operation type. The `Action` type of the Schema.org vocabulary provides a detailed hierarchy of `Action` sub-types that can be used by the property.

III. OpenAPI 3.0 Ontology

OpenAPI ontology in Fig. 2 captures all information specified by SOAS 3.0 description. Properties of classes are mapped to classes as well. At the heart of the ontology is Hydra Core Vocabulary, enhanced with additional models in order to capture all information provided by SOAS (i.e. security, headers, constraints).

![Fig. 2. SOAS 3.0 ontology](image)

In accordance to OAS 3.0 structure, `Document` provides general information (`Info` class) regarding the service; it specifies service paths, the entities and the security schemes that it supports. `Path` class represents (relative) service paths (pathName property). `Operation` class provides information for sending HTTP requests to the service as well as the HTTP responses. Responses are further described by `Response` class, specifying the status code and the data returned. The entire range of HTTP response values is represented. Class `Operation` refers to a security scheme in `SecurityRequirement` class.

Fig. 3 displays the security schemes supported by OAS 3.0. Listing 4 illustrates how an OAS `Path` item and `Operation` are defined in the ontology using the example of Listing 3. Class `Security` defines the security schemes that the specification supports. Class `Operation` refers to a security scheme using class `SecurityRequirement`, which in the case of the OAuth2 security scheme represents the scopes of the operation. Operation `path2_op1` refers to a `SecurityRequirement` individual, specifying an OAuth2 security scheme (i.e. `petstore_oauth` individual) and the corresponding scope (i.e. `read_pets` and `write_pets` individuals). Individual `path2_op1` is also considered to be an individual of `SearchAction` type defined in Schema.org vocabulary (i.e. as defined by the `x-operationType` extension property).

Listing 4. Representation of `Path` and `Operation` in the ontology

```json
ex:path2
  a openapi:Path ;
  openapi:pathName "/pets/findByStatus" ;

ex:path2_op1
  a openapi:Operation ,
    schema:SearchAction ;
  openapi:onPath ex:path2
  openapi:method openapi:GET ;
  openapi:tag ex:tag_pet
  openapi:parameter ex:query_status
  openapi:response [ openapi:status Code 400 ; openapi:description "Invalid status value" . ] ;
  openapi:security [ openapi:SecurityRequirement [ openapi:oauth2Scheme openapi:PetStore ,
    openapi:scopes [ openapi:scopeReadPets ,
      openapi:scopeWritePets ] ]
```
openapi:securityType ex:petstore_oauth;
openapi:scope ex:read_pets, ex:write_pets
]
openapi:name "findPetsByStatus";
openapi:summary "Finds Pets by Status";
openapi:description "Multiple status values with comma seperated strings".

### Table II

<table>
<thead>
<tr>
<th>Schema Object property</th>
<th>SHACL property</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum</td>
<td>sh:exclusiveMaximum if openAPI exclusiveMaximum is true</td>
</tr>
<tr>
<td></td>
<td>sh:inclusiveMaximum if openAPI exclusiveMaximum is false</td>
</tr>
<tr>
<td>exclusiveMaximum</td>
<td>sh:exclusiveMinimum if openAPI exclusiveMinimum is true</td>
</tr>
<tr>
<td></td>
<td>sh:inclusiveMinimum if openAPI exclusiveMinimum is false</td>
</tr>
<tr>
<td>minLength</td>
<td>sh:minLength</td>
</tr>
<tr>
<td>maxLength</td>
<td>sh:maxLength</td>
</tr>
<tr>
<td>pattern</td>
<td>sh:pattern</td>
</tr>
<tr>
<td>maxItems</td>
<td>sh:maxCount</td>
</tr>
<tr>
<td>minItems</td>
<td>sh:minCount</td>
</tr>
<tr>
<td>enum</td>
<td>sh:in</td>
</tr>
<tr>
<td>allOf</td>
<td>sh:and</td>
</tr>
<tr>
<td>oneOf</td>
<td>sh:xone</td>
</tr>
<tr>
<td>anyOf</td>
<td>sh:or</td>
</tr>
<tr>
<td>not</td>
<td>sh:not</td>
</tr>
<tr>
<td>default</td>
<td>sh:defaultValue</td>
</tr>
</tbody>
</table>

**Schema Objects** become classes, object and data properties using SHACL vocabulary. SHACL is an RDF vocabulary that can be used to describe and validate the structure of RDF data, similarly to XML-Schema or JSON-Schema. SHACL is used to define classes together with constraints on their properties. It provides built-in types of constraints (e.g. cardinality: minCount/maxCount) and allows expression of constraints on the type of properties and on the values they can take. Table II shows the mapping between Schema Object and SHACL.

Classes **NodeShape** and **PropertyShape** are sub-classes of **class Shape**. They are classes that describe models of an OAS 3.0 description and their properties, respectively. Class **NodeShape** is used for describing a **Schema object** in the ontology. It defines the properties of a class and may also refer to the operations related to this class (i.e. supportedOperation), which comes from **x-onResource** extension property. Class **PropertyShape** represents the properties of a class, their datatype and restrictions (e.g. a maximum value for a numeric property) and indicates whether the supported property is required or read-only.

Listing 5 shows how the **Person** model of Listing 1 is represented in the ontology. The model contains references to **Schema.org** vocabulary using x-refersTo. The SHACL class **PersonShape** is now defined according to **Schema object definition of Person** with the addition of new data properties and constraints (e.g. each person has exactly one first name, last name and gender). A Schema object defined using the combination of allOf property is represented as a subclass of the semantic model that is extended. A Schema object annotated with the x-kindOf extension property is also defined as a subclass of the referenced semantic concept.

Listing 6 shows that **PetCollection** object of Listing 2 becomes a subclass of class **Collection**. Without x-collectionOn property, **PetCollection** would be a simple class without any reference of being a collection. OAS 3.0 parameters are represented as separate classes for each parameter type. **Header** and **Cookie** parameters in HTTP requests and responses, become individuals of classes **Header** and **Cookie**, respectively. Class **Parameter** represents **Path** and **Query** parameters.

SOAS 3.0 is currently being applied on GURU® services catalogue. Both, a SOAS 3.0 catalogue and an ontology with services as instances will be available on the Web.

### References

