Mapping the Web Ontology Language to the OpenAPI Specification

Paola Espinoza-Arias[1][0000−0002−3938−2064], Daniel Garijo[2][0000−0003−0454−7145], and Oscar Corcho[1][0000−0002−9260−0753]

[1] Ontology Engineering Group, Universidad Politécnica de Madrid
{pespinoza,ocorcho}@fi.upm.es
[2] Information Sciences Institute, University of Southern California
dgarijo@isi.edu

Abstract. Many organizations maintain knowledge graphs that are organized according to ontologies. However, these ontologies are implemented in languages (e.g. OWL) that are difficult to understand by users who are not familiar with knowledge representation techniques. In particular, this affects web developers who want to develop ontology-based applications but may find challenging accessing ontology-based data in knowledge graphs through SPARQL queries. To address this problem, we propose an accessible layer for ontology-based knowledge graphs through REST APIs. We define a mapping specification between the Web Ontology Language (OWL) and the OpenAPI Specification (OAS) to provide ontology-based API definitions in a language well-known to web developers. Our mapping specification identifies key similarities between OWL and OAS and provides implementation guidelines and examples. We also developed a reference mapping implementation that automatically transforms OWL ontologies into OpenAPI specifications in a matter of seconds.

Keywords: OWL · OpenAPI · REST API · ontologies

1 Introduction

Many public and private organizations have adopted a knowledge-driven approach to make publicly available their knowledge graphs. Ontologies [10] play a crucial role in this approach, as they describe the knowledge about a domain in an agreed and unambiguous manner; and they allow organizing data, ease its reusability, and facilitate its interoperability. Ontologies are usually formalized in the Web Ontology Language (OWL) [6], a W3C recommendation to represent the semantics of a domain in a machine-readable way. However, OWL has a steep learning curve due its inherent complexity [12], and newcomers may get confused with the meaning of constraints, axioms or the Open World Assumption.

This problem has become evident in the case of developers who have an interest in taking advantage of the data available in existing knowledge graphs but are not used to Semantic Web technologies. Instead, developers are familiar with REST APIs as a resource-access way which hides details about the

implementation of operations for resource management or how such resources have been described according to the data models. To describe APIs, several Interface Description Languages have been defined to document their domain, functional and non-functional aspects. The OpenAPI Specification[3] (OAS) is a broadly adopted de facto standard for describing REST APIs in a programming language-agnostic interface. OAS allows both humans and computers to understand and interact with a remote service. Due to its wide adoption, OAS has a big community behind, wich has provided tools to allow developers to generate API documentation, server generation, mockup design, etc.

In this paper we describe additional work in the direction of making ontology-based data available though REST APIs. We define a mapping specification between OWL and OAS to facilitate the work of those who are interested in using data represented by semantic technologies and have to face the challenging task of developing ontology-based applications. Our mapping also aims to enhance adherence to the FAIR data principles [13] by facilitating: Findability, as it provides a template for finding types of available resources in knowledge graphs; Accessibility because it allows translating the ontology to an interface that developers are used to; Interoperability because the mapping matches two recommendations, the OWL W3C recommendation and OAS de facto standard; and Reusability because the mapping also translates explicit and understandable data definitions available in the ontology and generates an HTML document with the API details that may be published on the Web.

Our work has the following contributions: 1) A mapping specification between OWL and OAS to provide ontology-based API definitions (Section 2); and 2) A reference implementation to automatically transform OWL ontologies into an OAS document according to our mapping specification (Section 3). We also present work related to our approach in Section 4 and discuss conclusions and future work in Section 5.

2 Mapping OWL to OAS

In this section we provide the details about our mapping specification. First, we explain the methodology followed to generate the mapping. Then, we present a summarized description of the main characteristics of the specification. Finally, we give an example on how an OWL ontology is translated into OAS according to our mapping specification.

2.1 Method for mapping generation

The method followed to generate the mapping included the following steps:

1. Manual analysis of the OWL constructs. We analyzed the constructs from the OWL 2 Web Ontology Language [4]. We also analyzed the RDFS [1] constructs and the XSD datatypes [8] that are used in OWL 2.

[3] https://github.com/OAI/OpenAPI-Specification

2. Manual analysis of the OAS definitions. We analyzed the definitions provided by the OpenAPI specification v3.0.3.[4]
3. Manual generation of the mapping specification. Once the analysis of OWL and OAS constructs and definitions had been completed, we selected the OAS definitions which allow representing the OWL constructs. Then, we wrote a specification document to describe the equivalences found. To show how these equivalences could be implemented, we developed a sample OWL ontology and its corresponding OAS representation. The mapping specification and the examples are available online.[5]

2.2 Mapping definitions

We summarize below the main concepts of OAS that we use in this mapping:

1. A `Schema Object` allows defining input and output data types. For example, we may specify objects such as *Person*, or any concept that has its own attributes, primitives, or any expression to specify types of values.
2. A `Component Object` holds a set of reusable definitions of schemas, parameters, responses, etc. that may be referenced from somewhere in the API definition. For example, a component may hold a *Person* schema definition.
3. A `Reference Object` allows linking to other components in the specification instead of defining them inline. For example, to reuse the *Person* schema definition we reference it from its definition specified in the `Component Object`.
4. A `Path Object` holds the resources exposed in an API. For example, the path of the *Person* resource may be */persons*. It contains `Path Item Object`s.
5. A `Path Item Object` describes the available operations (HTTP methods to manage the resource) on a single path. For example, we may specify that the */persons* path allows the GET method.

Figure 1 illustrates these concepts with an example of a *Person* schema. Each number in the figure corresponds to the number shown in the enumeration list presented above. Table 1 describes the prefixes that we use throughout this paper. Regarding the mapping specification details, we present them in three main sets corresponding to classes and properties, restrictions, and boolean combinations. Table 2 shows the similarities between OWL classes and properties and OAS definitions.

In general, OWL classes are similar to a `Schema Object` that must be defined as an object type. OWL object and data properties are similar to the `Schema Object`'s properties and depending on the property type its value can be a data type or an object. Also, when a property is functional it must be defined as an array with 1 as the maximum number of array items. Additional details about similarities between OWL and OAS data types are provided in our online specification,[6] including a naming strategy for `Schema Objects` and `Paths`. Each

[4] http://spec.openapis.org/oas/v3.0.3
[5] https://w3id.org/OWLToOAS
[6] https://owl-to-oas.readthedocs.io/en/latest/mapping/#data-types_1

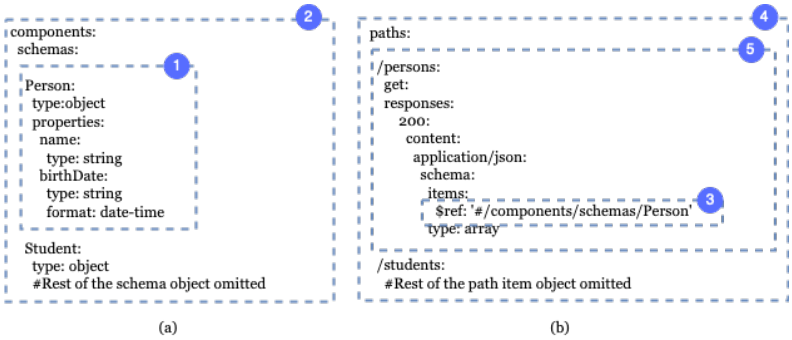

Fig. 1. Example of OAS definitions: a) A `Component Object` which contains some `Schema Object` definitions; b) A `Path Object` wich includes some `Path Item Objects`.

Table 1. Prefixes used in this document

Prefix	Namespace
`owl`	http://www.w3.org/2002/07/owl#
`rdfs`	http://www.w3.org/2000/01/rdf-schema#
`rdf`	http://www.w3.org/1999/02/22-rdf-syntax-ns#
`xsd`	http://www.w3.org/2001/XMLSchema#
`ex`	https://w3id.org/example#

Table 2. Classes and Properties Mapping

OWL	OAS	Implementation details
Classes		
`owl:Class`	`Schema Object`	The `Schema Object` must be defined as a `type: object` in the `Component Object`.
`rdfs:subClassOf`	`Schema Object` and `allOf`	The `Schema Object` must be defined in the same manner as the `owl:Class`. The `allOf` field must also be included referencing to the parent class defined as a `Reference Object`.
Properties		
`owl:DatatypeProperty`	`properties`	Defined as a `Schema Object`'s property.
`owl:ObjectProperty`	`properties`	Defined as a `Schema Object`'s property.
`rdfs:domain`	`Schema Object`	Defined as the `Schema Object` where the property should be defined.
`rdfs:range`	`type`	Defined as the property type value.
`owl:FunctionalProperty`	`maxItems`	The property must be defined as a `type: array` with 1 as the maximum number of items (`maxItems: 1`).

OWL class has a `Path Item Object` (as shown in Figure 1). Depending on the method used (GET, POST, PUT or DELETE), a class may be referenced as part of the `Response` or `Request Body`.

Table 3 shows the similarities between OWL restrictions and OAS. In general, properties must be defined as an array, except for properties with a specific value restriction (`owl:hasValue`). If that is the case, the property must be defined as `default` and it must specify its corresponding literal (for data properties) or its individual URI value (for object properties). When a property is restricted to

Table 3. Restrictions Mapping

OWL	OAS	Implementation details
owl:onProperty	properties	The restriction must refer to the property name where it is applied.
owl:onClass	Schema Object	The restriction must refer to the schema name where it is applied.
owl:hasValue	default	The restriction must be defined as a default property. Depending on whether it is on a data or object property, it will be a literal or an indvidual URI value.
owl:someValuesFrom	type, nullable	The property must be defined as a not nullable (nullable: false) array (type: array). Depending on whether it is on a data or object property the item's type value will be the restricted data type or Reference Object.
owl:allValuesFrom	type, nullable	The property must be defined as a nullable (nullable: true) array (type: array). Depending on whether it is on a data or object property the item's type value will be the restricted data type or Reference Object.
owl:minCardinality	minItems	The restriction must be defined as the minimum number of array items (minItems).
owl:maxCardinality	maxItems	The restriction must be defined as the maximum number of array items (maxItems).
owl:cardinality	minItems and maxItems	The restriction must be defined as the same minimum (minItems) and maximum (maxItems) number of array items.
owl:minQualifiedCardinality	minItems	The restriction must be defined as the minimum number of array items (minItems).
owl:maxQualifiedCardinality	maxItems	The restriction must be defined as the maximum number of array items (maxItems).
owl:qualifiedCardinality	minItems and maxItems	The restriction must be defined as the same minimum (minItems) and maximum (maxItems) number of array items.

owl:someValuesFrom or owl:allValuesFrom, a type of value must define the type of array items as a data type or as a Reference Object. When a property has a cardinality restriction, the maximum or minimum number of array items should be adjusted accordingly. Note that we used a Close World Assumption for translating the existential constructs because it is what developers expect when inserting and retrieving instances from an API.

Table 4 presents the translation of OWL boolean combinations into OAS. In general, these combinations may be applied to the Schema Object's properties. Depending on the combination, it is allowed to represent that a property value must be compliant with all or any schema type, or that it must not be valid against a certain schema. Also, a property may have one value included in an enumeration list.

Table 5 summarizes the coverage of our mapping specification in terms of SROIQ expressiveness, the description logic underlying OWL 2 DL. The mapping covers functional properties (F), concept negation (C), hierarchies (H), nominal values (O), cardinality (N) and qualified cardinality restrictions (Q), and data types (D). Role disjointness (R) and inverse properties (I) are not covered because OAS does not have a similar definition to represent them. OAS can represent unions (U), but we consider to be partially covered, as OWL allows defining complex combinations that are not possible in OAS, like the union of the intersection between concepts. Similarly, the existential qualification expression (E)

Table 4. Boolean combinations

OWL	OAS	Implementation details
`owl:intersectionOf`	`allOf`	The combination must be defined as `allOf` which validates the property value against all the schemas.
`owl:unionOf`	`anyOf`	The combination must be defined as `anyOf` which validates the property value against any (one or more) of the schemas.
`owl:complementOf`	`not`	The combination must be defined as `not` which validates that the property value is not valid against the specified schema.
`owl:oneOf`	`enum`	The combination must be defined as `enum` which holds the possible property values.

Table 5. Mapping OWLToOAS specification coverage (✓= covered, - = partially covered, and x = not covered)

Expressivity	Coverage	Expressivity	Coverage
F	✓	O	✓
E	-	I	x
U	-	N	✓
C	✓	Q	✓
H	✓	D	✓
R	x		

is also partially covered in our mapping, except when complex combinations are included in OWL.

2.3 Mapping example

To showcase our mapping, we provide a code snippet of an example ontology and its OAS representation. The OAS definitions are provided in YAML and the OWL constructs in Turtle. Listing 1.1 presents an ontology snippet with the *Professor* class and its data and object properties, e.g. *Professor* has a degree (*hasDegree*) from a list of values. *Professor* is subclass of *Person*, thus it inherits all the restrictions from the *Person*. Restrictions defined over properties, e.g. a *Professor belongsTo* a *Department*, must be also taken into account as part of the *Professor* class definition.

```
ex:Professor rdf:type owl:Class ;
    rdfs:subClassOf :Person ,
    [ rdf:type owl:Restriction ;
        owl:onProperty :hasDegree ;
            owl:someValuesFrom [ rdf:type owl:Class ;
            owl:oneOf (<https://w3id.org/example/resource/Degree/MS>
            <https://w3id.org/example/resource/Degree/PhD>)]] .
ex:Person rdf:type owl:Class ;
    rdfs:subClassOf [ rdf:type owl:Restriction ;
        owl:onProperty :address ;
        owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
        owl:onDataRange xsd:string] ;
    rdfs:comment "A human being regarded as an individual."@en ;
    rdfs:label "Person"@en .
ex:belongsTo rdf:type owl:ObjectProperty ;
    rdfs:domain :Professor ;
    rdfs:range :Department ;
    rdfs:label "belongs to"@en .
```

Listing 1.1. Ontology code excerpt

Listing 1.2 shows the OAS definitions corresponding to the previous ontology. This snippet was obtained from the YAML generated by our mapping implementation (described in Section 3). The `Component Object` includes the *Professor* schema (a `Schema Object`) which represents the *Professor* class including its own properties and those inherited from the *Person*.

```
components:
  schemas:
    Professor:
      type: object
        description: A university academic.
        properties:
          address:
            items:
              type: string
            maxItems: 1
            nullable: true
            type: array
          belongsTo:
            items:
              $ref: '#/components/schemas/Department'
            nullable: true
            type: array
          hasDegree:
            items:
              enum:
              - <https://w3id.org/example/resource/Degree/MS>
              - <https://w3id.org/example/resource/Degree/PhD>
              format: uri
              type: string
            type: array
```

Listing 1.2. OAS snippet of a `Component Object` that includes the *Profesor* schema

Listing 1.3 shows the path assigned to *Professor* in the API. This `Path Item` defines a GET operation, including a successful response which delivers a *Professor* schema:

```
paths:
  /professors:
    get:
      description: Gets a list of all instances of Professor
      responses:
        200:
          content:
            application/json:
              schema:
                items:
                  $ref: '#/components/schemas/Professor'
                type: array
```

Listing 1.3. OAS snippet of a `Path Item Object` generated from the OWL ontology

3 Mapping implementation

We implemented our mapping specification by extending the Ontology Based APIs Framework (OBA) [3], an existing tool for helping users create REST APIs from ontologies. OBA generates a server with a REST API based on an OpenAPI specification, and includes the automated management of SPARQL query templates for common operations; but has a limited support for OWL constructs (mostly limited to `rdfs:Class`, `rdfs:domain` and `rdfs:range`).

We extended the OBA Specification Generator module, which takes the ontology code and generates the OAS document in YAML, to support our mapping specification. It is worth mentioning that OBA does not check for consistency or syntactic correctness of an ontology, assuming that it has been evaluated before. The implementation release is available at the OBA's GitHub repository.[7]

Our implementation allows generating API definitions for ontologies of different sizes with a reasonable overhead. We tested our implementation in an average laptop (Intel Core i7 2.6Ghz with 16 GB of RAM) with the example ontology we defined to illustrate our mapping specification,[8] an ontology which contains 119 logical axioms, including 36 class axioms, 42 object property axioms, and 37 data property axioms. The corresponding OAS was generated in 6 seconds. We also tested the DBPedia ontology,[9] which contains over 6000 logical axioms, including over 700 class axioms, over 2000 object property axioms, and over 2000 data property axioms. The DBpedia OAS took 64 seconds to build. In both cases, we generated only GET operation for each path in the specification.

4 Related Work

Several efforts have attempted to promote and facilitate Semantic Web technology adoption by web developers, providing Web APIs to allow developers accessing and managing data from knowledge graphs. Specifications like the Linked Data API[10] (LDA), the Linked Data Platform (LDP) [9], and the Triple Pattern Fragments (TPF) [11] have been proposed to describe how to define such interfaces. LDA details how to define read-only interfaces to Linked Data, LDP describes how to design read/write HTTP interfaces to RDF data, and TFP defines read-only interfaces to specific triples from a dataset to provide an efficient client-side querying execution. However, they do not use ontologies as templates for the API generation, hence developers have to deal with them to manage APIs. Works like BASIL [2] and GRLC [7] have been proposed to generate Web APIs on top of SPARQL endpoints; both generate Swagger specifications for the resulting APIs. However, these specifications are generated from the SPARQL queries, GitHub query repositories and SPARQL decorator notation which have to be defined manually by developers. Thanks to our mapping implementation, a full OAS can be generated from an ontology without human intervention.

Other recent approaches focus on generating API definitions from ontologies. In this regard, two efforts have recently appeared: the Ontology Based APIs Framework (OBA) [3] (which we extended with our mapping implementation) and the OWL2OAS Converter.[11] Both efforts generate OAS documents from OWL, and were efforts developed in parallel. The main differences between them and our implementation are summarized in Table 6. On the one hand, OBA

[7] https://github.com/KnowledgeCaptureAndDiscovery/OBA/releases/tag/3.4.0
[8] http://tiny.cc/3eyjsz
[9] https://wiki.dbpedia.org/services-resources/ontology
[10] https://code.google.com/archive/p/linked-data-api
[11] https://dev.realestatecore.io/OWL2OAS

Table 6. Comparison ontology-based APIs approaches (✓= included, x = not included, - = less coverage, and + = more coverage of OWL constructs)

Proposal	**OWL2OAS**	**OBA**	**Our approach**
OWL to OAS mapping available	X	✓	✓
Expressiveness	+	-	++
REST Server	X	✓	✓

provides an initial mapping to describe details on translation of OWL constructs into OAS. However, the expressiveness covered by OBA is basic; restricted to the translation of classes, subclasses, object and data properties with singleton ranges. Despite its basic expressiveness, OBA provides extra functionality for implementing the API as a REST API server which allows validating requests from users, generating tests for API calls, etc. On the other hand, OWL2OAS does not include a specification of its coverage from OWL into OAS. By manually inspecting the OWL2OAS's code repository we can see that, in addition to OBA's coverage, it includes support for functional properties, class and property restrictions (`owl:onClass`, `owl:onProperty`), existential and some cardinality restrictions. However, it does not support specific values (`owl:hasValue`) and boolean combinations (`owl:oneOf`, `owl:unionOf`, `owl:intersectionOf`, `owl:complementOf`) which we cover (partially) in our mapping.

Our contribution proposes a detailed mapping between OWL and OAS, aimed at facilitating a specification, including examples on how to transform an ontology into an API definition, and an implementation to automatically generate an OAS document based on this mapping. Our mapping and its implementation have been built on top of OBA, reusing the work previously done.

5 Conclusions and Future Work

In this work, we proposed a mapping specification to translate OWL ontologies into OAS. This specification facilitates the creation of REST APIs that enables developers to access ontology-based data. Our mapping includes examples and details on how to define each OWL construct as an OAS definition using a Close World Assumption. Since manually editing OAS definitions can be tedious, time consuming, and error-prone, we extended the Ontology Based API framework to automatically translate OWL constructs into OAS. However, not all OWL constructs are covered in our specification, because they do not have an equivalent OAS definition. For example we cannot represent the equivalence between classes or restrictions that include complex unions and intersections.

As future work, we plan to extend OBA to improve the schema and path naming strategy used in the API. We would like to generate these names from the ontology class and property labels instead of the URI fragments as OBA currently does. We also plan to use the smartAPI specification [14], an extended version of OAS for defining key API metatada, to annotate our resulting APIs to maximize their FAIRness. This way, API providers may publish their APIs into

the smartAPI registry to make them more discoverable, connected and reusable. Finally, given the limitations of OAS to represent some OWL constructs, we will explore how to combine our API definitions with the Shapes Constraint Language (SHACL) [5]. SHACL was created for data validation and therefore allows defining the restrictions that data from knowledge graphs must fulfill. With SHACL, those OWL constructs that are not covered in our mapping may be defined as shapes to validate the requests received by an API.

Acknowledgments

This work has been supported by a Predoctoral grant from the I+D+i program of the Universidad Politécnica de Madrid and the Spanish project DATOS 4.0: RETOS Y SOLUCIONES (TIN2016-78011-C4-4-R).

References

1. Brickley, D., Guha, R.V., McBride, B.: RDF Schema 1.1. W3C recommendation (2014)
2. Daga, E., Panziera, L., Pedrinaci, C.: A BASILar approach for building web APIs on top of SPARQL endpoints. In: CEUR Workshop Proceedings. vol. 1359, pp. 22–32 (2015)
3. Garijo, D., Osorio, M.: OBA: An Ontology-Based Framework for Creating REST APIs for Knowledge Graphs (Jul 2020), https://arxiv.org/abs/2007.09206
4. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S., et al.: OWL 2 web ontology language primer. W3C recommendation (2009)
5. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL). W3C recommendation (2017)
6. McGuinness, D.L., Van Harmelen, F., et al.: OWL web ontology language overview. W3C recommendation **10**(10), 2004 (2004)
7. Meroño-Peñuela, A., Hoekstra, R.: grlc makes GitHub taste like linked data APIs. In: European Semantic Web Conference. pp. 342–353. Springer (2016)
8. Peterson, D., Gao, S., Malhotra, A., Sperberg-McQueen, C.M., Thompson, H.S., Biron, P.: W3C XML schema definition language (XSD) 1.1 part 2: Datatypes. W3C Recommendation **5** (2012)
9. Speicher, S., Arwe, J., Malhotra, A.: Linked data platform 1.0. W3C recommendation (2015)
10. Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: principles and methods. Data & knowledge engineering **25**(1-2), 161–197 (1998)
11. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., et al.: Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. Journal of Web Semantics **37**, 184–206 (2016)
12. Vigo, M., Bail, S., Jay, C., Stevens, R.: Overcoming the pitfalls of ontology authoring: Strategies and implications for tool design. International Journal of Human-Computer Studies **72**(12), 835–845 (2014)
13. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., et al.: The FAIR Guiding Principles for scientific data management and stewardship. Scientific data **3** (2016)
14. Zaveri, A., Dastgheib, S., Wu, C., Whetzel, T., Verborgh, R., Avillach, P., Korodi, G., Terryn, R., Jagodnik, K., et al.: smartAPI: towards a more intelligent network of web APIs. In: European Semantic Web Conference. pp. 154–169. Springer (2017)