Enabling Automatic Discovery and Querying of Web APIs at Web Scale using Linked Data Standards

Franck Michel, Catherine Faron Zucker, Olivier Corby, Fabien Gandon

Enabling Automatic Discovery and Querying of Web APIs at Web Scale using Linked Data Standards

Franck Michel
Université Côte d'Azur, CNRS, Inria, I3S, France
franck.michel@cnrs.fr

Catherine Faron-Zucker
Université Côte d'Azur, Inria, CNRS, I3S, France
faron@i3s.unice.fr

Olivier Corby
Université Côte d'Azur, Inria, CNRS, I3S, France
olivier.corby@inria.fr

Fabien Gandon
Université Côte d'Azur, Inria, CNRS, I3S, France
fabien.gandon@inria.fr

ABSTRACT

To help in making sense of the ever-increasing number of data sources available on the Web, in this article we tackle the problem of enabling automatic discovery and querying of data sources at Web scale. To pursue this goal, we suggest to (1) provision rich descriptions of data sources and query services thereof, (2) leverage the power of Web search engines to discover data sources, and (3) rely on simple, well-adopted standards that come with extensive tooling. We apply these principles to the concrete case of SPARQL micro-services that aim at querying Web APIs using SPARQL. The proposed solution leverages SPARQL Service Description, SHACL, DCAT, VoID, Schema.org and Hydra to express a rich functional description that allows a software agent to decide whether a micro-service can help in carrying out a certain task. This description can be dynamically transformed into a Web page embedding rich markup data. This Web page is both a human-friendly documentation and a machine-readable description that makes it possible for humans and machines alike to discover and invoke SPARQL micro-services at Web scale, as if they were just another data source. We report on a prototype implementation that is available on-line for test purposes, and that can be effectively discovered using Google's Dataset Search engine.

KEYWORDS

SPARQL, Web API, discovery, dataset, Web service, Linked Data

ACM Reference Format:
Franck Michel, Catherine Faron-Zucker, Olivier Corby, and Fabien Gandon. 2019. Enabling Automatic Discovery and Querying of Web APIs at Web Scale using Linked Data Standards. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW '19 Companion), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3308560.3317073

1 INTRODUCTION

As data sources multiply all over the Web, it becomes increasingly crucial to enable the automatic discovery of sources suitable to answer a certain query. Multiple portals and services allow to search datasets based on metadata, such as the manifold CKAN-based portals[1] like Datahub[2] and the data portals of European states and institutions. Some portals specialize in specific dataset formats or interface technologies. For instance, ProgrammableWeb.com[3] registers Web APIs, a loosely defined category of lightweight Web services also referred to as REST-like or Lo-REST [25] services, while LODAtlas [26] and SPARQLES [31] focus on RDF datasets and SPARQL endpoints respectively.

Even though some of these portals have gained significant popularity due to the large number of datasets that they index, they suffer relentless flaws. Firstly, they are centralized registries with a somehow restricted scope. Consequently, potential data consumers may have to query several portals one by one, accommodating the various query interfaces, to discover suitable datasets. Secondly, in many cases, datasets are manually registered and annotated by dataset producers, thereby raising concerns about outdated metadata or deprecated services. Thirdly, metadata-based search results have a limited relevance. Typically, searching datasets by keywords and data formats is a first step in the discovery process, but a potential consumer needs deeper insight in the data themselves and the technical interfaces available to query the dataset. In this respect, WSDL-based semantic Web services (e.g. OWL-S [4] or SAWSDL [8]) tackled this question with a thorough description of the exchanged messages, yet often failing to describe the actual dataset being queried. Besides, they were better suited to the controlled environment of companies [20] than the open environment of the Web. By contrast, the VoID vocabulary [1] can help describe RDF datasets with regards to vocabularies, classes and properties used, links to other datasets, etc. But it does not address the description of what properties a resource may typically have nor how the resources relate to each other, which are key criteria in the discovery and selection of datasets.

To spur and enable automatic discovery and consumption of datasets at Web-scale, we believe that a few principles should drive future research and developments.

(1) **Metadata-based search is not enough**. As we pointed out above, metadata-based search using e.g. keywords, data formats, vocabularies or even classes and properties used in an RDF dataset, is just a first step in the discovery process. For example, assume a biologist wants to develop a software agent capable of browsing Linked Data and gathering photos

WWW '19 Companion, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6675-5/19/05.
https://doi.org/10.1145/3308560.3317073

[1] https://ckan.org/about/instances/
[2] https://datahub.ckan.io/
[3] https://www.programmableweb.com/

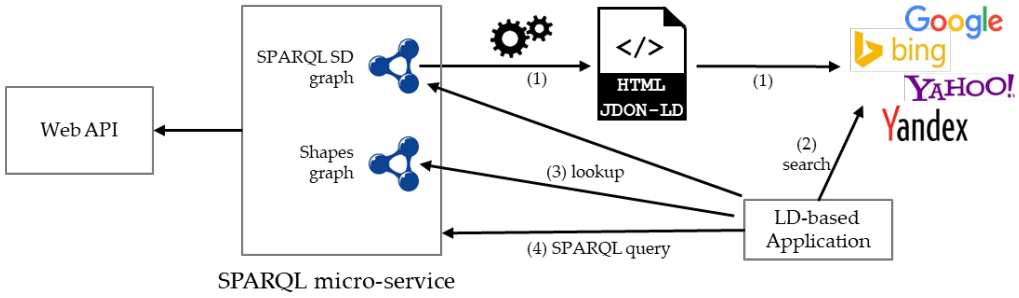

Figure 1: Envisaged use case spanning the indexing, discovery and invocation of a SPARQL micro-service.

related to biological species. The agent may submit several queries to repositories such as LODAtlas looking for datasets whose textual description contains keywords "photo" and "biodiversity", or those whose VoID description (if any) mentions classes representing photographies and biological taxa. Within the matching datasets, however, nothing guarantees that photographic resources do actually depict biological species; photographies may well be scans of academic papers related to the species. Hence, the agent has no choice but to query the dataset in order to get insight into it and find out if it matches the search. This simple example illustrates the lack of in-depth semantic description of datasets, that would consist of the resources (what are the actual properties of photographic and taxonomic resources) along with their mutual relationships.

(2) **The discovery of datasets at Web-scale should leverage the power of Web search engines**. Major search engines such as Google, Yahoo and Bing crawl and index an unprecedented breadth of information every day. They already harvest the content of specialized open data portals, in particular by taking advantage of the growing use of the Schema.org vocabulary [12]. Google has recently opened a beta service specifically dedicated to dataset search[4]. Therefore, despite concerns raised by the Web centralization effect of search engines, it is worth studying how we can take advantage of their services to enable the discovery and querying of datasets at Web scale.

(3) **The description of datasets and their query services should rely on well adopted (de-facto) standards**. Enabling the automatic discovery and querying of datasets at Web-scale means that, at some point, a consensus should be reached with respect to technologies and practices. Such a consensus may emerge only if the selected approaches put little constraints on and require little efforts from those in charge of describing datasets, publishing and maintaining query services thereof. This means relying on existing, well-adopted standards or de-facto standards.
In terms of semantic description, existing vocabularies should be leveraged, ranging from mature and widely used W3C standards to de-facto standards such as Schema.org that benefits from a large and growing adoption even though it still lacks terms in many domains. Additionally, selected approaches should enjoy sufficient and appropriate tooling with APIs in various programming languages. Such tools should be relatively simple in the sense that (i) they should not require a long learning curve from developers, and (ii) they should be easy to deploy and maintain. In this respect, the example of WSDL-based semantic Web service frameworks is inspiring: their deployment and operation required significant efforts that only companies with solid IT services were ready to invest [25]. But when seeking Web-scale adoption, such perceived complexity would have a counterproductive effect.

In a previous work, we defined the SPARQL Micro-Service architecture [23] aimed at querying Web APIs using SPARQL [14], thus bridging the Linked Data and Web API worlds. We suggested that this approach could foster the emergence of an ecosystem of SPARQL services published by independent providers, allowing Linked Data-based applications to glean pieces of data from a wealth of distributed data sources, in a scalable and reliable manner.

In this article, we present further exploratory works aimed at applying the principles set out above, and thereby make SPARQL micro-services effectively discoverable and queryable at Web-scale. We describe and explain our architectural and modeling choices. Let us however underline that alternative choices may be figured out, driven by different incentives or trade-offs. We touch upon these considerations in the last section.

Envisaged use case. Figure 1 outlines the main steps of a typical use case as we see it, along with the main choices that we made. A SPARQL micro-service produces a Web page (step 1) whose primary goal, beyond providing developers with appropriate documentation and a testing interface, is to be processed by Web crawlers. It embeds rich markup data, notably based on Schema.org, to enhance indexing and help search engines yield more accurate results. The Web page is generated dynamically from the service self-description that consists of a SPARQL Service Description (SD) graph [33] and a SHACL shapes graph [16]. Together, they provide various metadata, a description of the graphs that the service typically spawns, the service inputs and outputs and the way they relate to one another. An application willing to carry out a certain task first queries search engines (step 2) for datasets matching certain keywords. From the search results, it extracts and looks up SPARQL endpoint URLs. SPARQL micro-services return an SD document that links to the shapes graph. In turn, the application fetches the shapes graph that allows verifying whether the service is indeed suited for the task

[4] https://www.blog.google/products/search/making-it-easier-discover-datasets/

Figure 2: SPARQL micro-service processing workflow.

(step 3). Based on the description of the service inputs, the application can submit an appropriate SPARQL query to the micro-service (step 4).

The rest of this article is organized as follows. Section 2 briefly summarizes the concepts of SPARQL micro-services and presents a quick example. Section 3 then presents the way we describe micro-services in a machine-readable manner. Section 4 focuses on the way micro-services are made discoverable at Web scale. Related works are discussed in section 5 while the last section brings elements of discussion and suggests future leads.

2 BACKGROUND

In [22], we described the SPARQL Micro-Service architectural principles. Later on in [23], we extended this description and reported on several biodiversity-related use cases. In this section, we briefly summarize these previous works.

The SPARQL Micro-Service architecture addresses the problem of combining Linked Data with data from non-RDF Web APIs. A SPARQL micro-service is a lightweight SPARQL endpoint that provides access to a graph generated at run-time. This graph is shaped by the Web API service being wrapped, the arguments passed to the micro-service and the types of RDF triples it is designed to produce. How the arguments are passed to a SPARQL micro-service, and how the Web API response is transformed into a SPARQL result, are implementation choices.

In accordance with the micro-service architecture principles [24], a SPARQL micro-service is typically designed to be loosely coupled (it is deployed independently of other services, possibly using lightweight container technologies such as Docker[5]) and fine-grained: it provides access to a small, resource-centric graph corresponding to a small fragment of the whole dataset served by the Web API.

Interestingly, this architecture can be used to assign dereferenceable URIs to Web API resources that do not have URIs in the first place: a micro-service responds to SPARQL queries by assigning URIs to Web API resources, while other micro-services are designed to dereference these URIs to RDF content. This entails an effective solution to bridge Web APIs, that are designed as closed worlds, with the open world of Linked Open Data.

Implementation. We implemented a lightweight PHP prototype available on GitHub[6] under the Apache 2.0 license. The prototype focuses on JSON-based Web APIs, and expects arguments of a micro-service to be passed as parameters of the service URL's query string. Figure 2 illustrates how a SPARQL micro-service S_μ evaluates a SPARQL query Q. In step 1, S_μ receives query Q and extracts the set Arg_w of arguments from the HTTP query string. In step 2, it invokes the Web API with the arguments in Arg_w, in addition to any other parameter required by the Web API. In step 3, S_μ translates the JSON response into an RDF graph: it carries out a first mapping towards selected vocabularies by applying a JSON-LD profile [30] to the response; the resulting graph G is loaded into a local triple store; if mappings are needed that JSON-LD cannot express, S_μ runs a SPARQL INSERT query that enriches G with additional triples. Finally, S_μ evaluates Q against G and returns the result to the client.

Alternative argument-passing method. In the method described above, the arguments of a SPARQL micro-service are passed as query string parameters rather than RDF terms. One advantage is that it spares creating new terms whenever a Web API-specific argument has no counterpart in existing vocabularies. Nevertheless, a downside is that the semantics of such a SPARQL micro-service differs from that of a standard SPARQL endpoint. Indeed, the SPARQL protocol treats a service URL as a black box, i.e. it does not identify nor interpret URL parameters apart from those specified in the SPARQL protocol itself. By contrast, in a SPARQL micro-service the query string parameters are meaningful arguments that shape the virtual graph being queried. Therefore, since one of our goals in this article is to comply with standards (principle 3), we have recently implemented an alternative method wherein arguments are passed as regular RDF terms of the SPARQL query graph pattern. To illustrate this, we now introduce an example that we shall reuse throughout the rest of this article.

Running example. Let us consider the service of Flickr's Web API that returns a list of photos matching some criteria[7]. We define $S_{\mu f}$ as a SPARQL micro-service[8] that wraps this Flickr service and returns photos of a given biological species. $S_{\mu f}$ takes as argument the species scientific (taxonomic) name, and searches photos matching this name. It abides by the convention that photos of a species should be tagged with the species scientific name formatted as `taxonomy:binomial=<scientific name>`[9]. $S_{\mu f}$ expects the scientific name argument to be passed as the object of the `dwc:scientificName` predicate.

Listing 1 depicts a query, Q_1, that meets this requirement. It aims at retrieving photos depicting species *Delphinus delphis*, the common dolphin. When it evaluates Q_1, $S_{\mu f}$ first extracts the scientific name argument from the graph pattern (highlighted line) and builds the following Web API invocation URL:

```
https://api.flickr.com/services/rest/?
  format=json&method=flickr.photos.search&
  tags=taxonomy:binomial=Delphinus+delphis
```

It then submits this invocation and translates the JSON response into an RDF graph such as the one exemplified in Listing 2. Finally,

[5] https://www.docker.com
[6] https://github.com/frmichel/sparql-micro-service/tree/0.3.1/

[7] https://www.flickr.com/services/api/flickr.photos.search.html
[8] The code of this service is available at https://frama.link/kVhhnE-v.
[9] This is a common convention used on Flickr for biodiversity resources, in particular the Encyclopedia of Life group (https://www.flickr.com/groups/806927@N20) and the Biodiversity Heritage Library (https://www.flickr.com/photos/biodivlibrary).

```
prefix schema: <http://schema.org/>
prefix dwc:    <http://rs.tdwg.org/dwc/terms/>

SELECT ?title ?img WHERE {
 ?taxon a dwc:Taxon;
   dwc:scientificName "Delphinus delphis";
   schema:image [
     a schema:Photograph;
     schema:name ?title;
     schema:contentUrl ?img.
    ].
}
```

Listing 1: Query Q_1 can be submitted to SPARQL micro-service $S_{\mu f}$ in order to retrieve photos of species *Delphinus delphis*.

```
[] a dwc:Taxon;
  dwc:scientificName "Delphinus delphis";
  schema:image <http://example.org/ld/flickr/photo/31173091626>.

<http://example.org/ld/flickr/photo/31173091626>
 a schema:Photograph;
 schema:name "Delphinus delphis 5 (13-7-16 San Diego)";
 schema:contentUrl
  <https://farm6.staticflickr.com/5718/31173091626_88c410c3f2_z.jpg>;
 schema:mainEntityOfPage
  <https://flickr.com/photos/10770266@N04/31173091626>;
 schema:fileFormat "image/jpeg";
 schema:author [
  schema:identifier "10770266@N04";
  schema:url <https://flickr.com/photos/10770266@N04>
 ].
```

Listing 2: Example graph produced by micro-service $S_{\mu f}$ to evaluate query Q_1.

it evaluates Q_1 against this graph, yielding the response exemplified below in the SPARQL Query Results JSON format [28]:

```
{ "head": { "vars": [ "title", "img" ] }, "results": {
    "bindings": [
      { "title": {
          "type": "literal",
          "value": "Delphinus delphis 5 (13-7-16 San Diego)" },
      { "img": {
          "type": "uri",
          "value": "https://farm6.staticflickr.com/5718/ \
                    31173091626_88c410c3f2_z.jpg" }
} ] } }
```

3 MACHINE-READABLE DESCRIPTION OF SPARQL MICRO-SERVICES

Building on the work presented in section 2, we aim at proposing a mechanism that enables a software agent to discover, select and invoke the SPARQL micro-services that are relevant for a certain task. In section 1, we pointed out three principles that, we believe, should help pursue this goal: (1) have rich descriptions of data sources that go beyond common metadata, (2) leverage Web search engines to discover data sources, and (3) rely on well-adopted standards. This section presents the modeling choices we made with respect to principle (1), section 4 deals with principle (2) while principle (3) is transversal to both sections.

A rich SPARQL micro-service description should span two distinct levels further detailed in this section. The **high-level description** consists of metadata about the data being queried (keywords, publisher, license, vocabularies, graphs, etc.), as well as metadata about the micro-service itself (supported operations, result formats, etc.). The **functional description** describes the actions the services carries out: what are the types of resources involved, what are their recommended/expected properties, what are the service arguments and how they relate to the resources.

3.1 High-level Description

To describe SPARQL micro-services, we use SPARQL Service Description (SD) [33] which is both a vocabulary to describe SPARQL endpoints and a method requiring compliant endpoints to return an SD document when their URL is looked up.

Listing 3 depicts a snippet of the SD document (in the Turtle syntax) for the example service $S_{\mu f}$ introduced in section 2. The service is at the same time an instance of the SD Service class and the class of SPARQL micro-services `sms:Service` (line 17). Common metadata are provided lines 19 to 26, such as a name and description, keywords, supported SPARQL language and result formats. A VoID description can also be embedded here, as exemplified in line 29 (the default dataset is stated to be a void:Dataset) and lines 34 to 36[10]. Additional triples are not depicted here for conciseness, such as the service publisher and an example SPARQL query. Note that many more metadata could be provided, such as common dataset profile features [2]. Furthermore, in the implementation we demonstrate here, we wrote the SD document manually. Future works could consider dataset profiling techniques to (at least partially) automate this generation.

The SD document is obtained by looking up the service URL. Content negotiation is supported such that a Web browser will obtain an HTML page, whereas a Linked Data application would typically require one of the supported RDF serialization syntaxes. The SD document itself is a named graph of the dataset served by the SPARQL micro-service (line 31). The interested reader may view the full range of metadata by looking up the named graph URI[11] in a Web browser (this will typically return an RDF/XML representation) or by issuing the following command on a standard Linux system:

```
curl --header "Accept: text/turtle" \
  http://sms.i3s.unice.fr/sparql-ms/flickr/getPhotosByTaxon_sd/
```

3.2 Functional Description

There exist various options to represent the functional description of a service. In section 5 we discuss some of them. As far as SPARQL micro-services are concerned, we choose to leverage several vocabularies for this purpose: SHACL [16], Schema.org and Hydra [19].

SHACL Description of the Dataset. SHACL, the Shapes Constraint Language, is designed for the validation of RDF graphs (called *data graphs*) against a set of conditions expressed in the form of *shapes graphs*. In our context, instead of using a shapes graph G_{sh} *a posteriori* to validate the data graph produced by a

[10] As an alternative, a VoID description could be made available using the *well-known URIs* mechanism, at path `/.well-known/void`.

[11] http://sms.i3s.unice.fr/sparql-ms/flickr/getPhotosByTaxon_sd/ServiceDescription

```
1 prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
2 prefix sd:     <http://www.w3.org/ns/sparql-service-description#>
3 prefix frmt:   <http://www.w3.org/ns/formats/>
4 prefix dct:    <http://purl.org/dc/terms/>
5 prefix shacl:  <http://www.w3.org/ns/shacl#>
6 prefix void:   <http://rdfs.org/ns/void#>
7 prefix hydra:  <http://www.w3.org/ns/hydra/core#>
8 prefix schema: <http://schema.org/>
9 prefix skos:   <http://www.w3.org/2004/02/skos/core#>
10 prefix dwc:    <http://rs.tdwg.org/dwc/terms/>
11 prefix sms:    <http://ns.inria.fr/sparql-micro-service#>
12
13 @base
14   <http://sms.i3s.unice.fr/sparql-ms/flickr/getPhotosByTaxon_sd/>.
15
16 <>
17   a sd:Service, sms:Service;
18   sd:endpoint <>;
19   sd:supportedLanguage sd:SPARQL11Query;
20   sd:feature sd:BasicFederatedQuery, sd:EmptyGraphs;
21   sd:resultFormat
22       frmt:SPARQL_Results_JSON, frmt:Turtle, frmt:JSON-LD;
23
24   schema:name "Search for Flickr photos by taxon scientific name";
25   schema:description "...";
26   schema:keywords "biodiversity", "lifesciences", "photography";
27
28   sd:defaultDataset [
29     a sd:Dataset, void:Dataset;
30     sd:defaultGraph [a sd:Graph; shacl:shapesGraph <ShapesGraph>];
31     sd:namedGraph   [a sd:Graph; sd:name <ServiceDescription>];
32     sd:namedGraph   [a sd:Graph; sd:name <ShapesGraph>];
33
34     void:vocabulary
35         <http://schema.org/>, <http://rs.tdwg.org/dwc/terms/>;
36     void:sparqlEndpoint <>;
37   ];
38
39   dct:source [
40     # Web API service being wrapped by this service
41     a schema:WebAPI; schema:name "Flickr API";
42     schema:url <https://www.flickr.com/services/api/>;
43     schema:potentialAction [
44       a schema:SearchAction;
45       a hydra:IriTemplate;
46       hydra:template "https://api.flickr.com/services/rest/? \
47         format=json&method=flickr.photos.search& \
48         tags=taxonomy:binomial={name}"
49
50       hydra:mapping [
51         hydra:variable "name";
52         schema:description "Taxon scientific name";
53         hydra:required "true"^^xsd:boolean;
54         skos:example "Delphinus delphis";
55
56         # Use either hydra:property or shacl:sourceShape
57         hydra:property dwc:scientificName;
58       ];
59     ];
60   ].
```

Listing 3: Snippet of the Service Description of SPARQL micro-service $S_{\mu f}$.

SPARQL micro-service, we consider G_{sh} as a specification of the graphs that a SPARQL micro-service can generate.

The shapes graph is linked to the SD document as follows: the default dataset has a default graph that is validated by the shapes graph (property `shacl:shapesGraph` lines 30). The shapes graph is itself one of the named graphs of the default dataset (line 32).

A short snippet of the shapes graph corresponding to service $S_{\mu f}$ is given in Listing 4. The interested reader may check the complete shapes graph on GitHub[12] or by dereferencing its URI[13]. It states that an instance of class `dwc:Taxon` (lines 4-5) should have exactly three properties: `rdf:type` with object `dwc:Taxon` (lines 9-10), `shacl:image` whose object should be validated against another shape (lines 12-13) and property `dwc:scientificName` that should have exactly one literal object (lines 17-18). Notice that the graph pattern of query Q_1 (Listing 1) specifically matches these constraints.

Description of the Input Arguments. We now need to characterize the micro-service input arguments, how they are extracted from a SPARQL graph pattern, and how they map to parameters of the Web API wrapped by the micro-service. We define the Web API as the micro-service data source (line 39 of Listing 3). It is typed as a Schema.org `WebAPI` having one potential action of type `SearchAction` (lines 40-44). Note that an alternative is currently being discussed within the Schema.org community, that links `EntryPoint` objects to a `WebAPI` [27]. The search action is also typed as a Hydra `IriTemplate` whose template string is the Web API invocation URL (lines 46-48). Each mapping (lines 50-58) maps a parameter used in the template string to a term of the SPARQL query by pointing to a specific property using `hydra:property`. In our example, the scientific name, denoted "`{name}`" in the template string (line 48), is mapped to property `dwc:scientificName` (line 57). Upon invocation, the service simply reads the value of property `dwc:scientificName` in the graph pattern, and substitutes it with "`{name}`" in the template string.

This solution is simple and concise, but it presents two downsides: (i) `hydra:property` only names a property but does not put any other constraint such as what is the subject of this property, or how many values are allowed; (ii) there is no explicit relationship between the input argument and the shapes graph. Hence, to specify the input arguments more precisely, an alternative is to map the parameter to a property shape of the shapes graph. In our example, this would be expressed by replacing line 57 with the following:

```
shacl:sourceShape <ShapesGraph#NamePropertyShape>;
```

The referenced property shape is defined in Listing 4 (lines 16-18). Not only it instructs that the scientific name should be given by property `dwc:scientificName`, but also that this property should be attached to an instance of the `dwc:Taxon` class and that there should be only one such property.

Advantages of using SHACL. We believe that using SHACL presents two advantages:
(1) SHACL's expressiveness allows denoting complex relationships between resources (e.g. cardinality, predicate paths). Even though this description is schema-based, it is sufficient to enable SPARQL micro-service discovery and selection since, by construction, the shape of generated graphs is know at design time. By contrast, SPARQL federated query engines generally rely on dynamic instance-based statistics because the graphs being queried can hardly be characterized by a static SHACL description. For instance, it would be impossible to define a precise shapes graph of crowd-sourced graphs such as DBpedia.
(2) A SHACL shapes graph is itself an RDF graph. Therefore, a software agent can leverage existing tooling to reason upon it and verify whether the SPARQL micro-service fulfills the agent's goals. As an illustration, we are currently developing a *SPARQL micro-service federated query engine*[14]. Given an input SPARQL query, the engine searches candidate SPARQL micro-services whose inputs

[12] Complete shapes graph on GitHub: https://frama.link/we_EQWnC
[13] Shapes graph URI: http://sms.i3s.unice.fr/sparql-ms/flickr/getPhotosByTaxon_sd/ShapesGraph

[14] Beta version available at https://frama.link/VWG7r8PF.

```
@base
  <http://sms.i3s.unice.fr/sparql-ms/flickr/getPhotosByTaxon_sd/>.

<ShapesGraph#TaxonShape> a shacl:NodeShape;
  shacl:targetClass dwc:Taxon;
  shacl:property
    <ShapesGraphNamePropertyShape>,
    [ a shacl:PropertyShape;
      shacl:path rdf:type;
      shacl:hasValue dwc:Taxon ],
    [ a shacl:PropertyShape;
      shacl:path schema:image;
      shacl:node <ShapesGraph#PhotographShape> ];
  shacl:closed true.

<ShapesGraphNamePropertyShape> a shacl:PropertyShape;
  shacl:path dwc:scientificName; shacl:nodeKind shacl:Literal;
  shacl:minCount 1; shacl:maxCount 1.

<ShapesGraph#PhotographShape> a shacl:NodeShape;
  ...
```

Listing 4: Snippet of the shapes graph of SPARQL micro-service $S_{\mu f}$.

are satisfied by the query. It then selects those whose shapes graphs validate some triple patterns of the query, and finally rewrites the input query into a UNION of SERVICE clauses that invoke SPARQL micro-services. Each step of the processing (selection, matchmaking, query rewriting) is performed using SPARQL queries that involve the SD documents, the shapes graphs and the input query.

3.3 Invocation

To process an incoming SPARQL query, a SPARQL micro-service needs to extract the input arguments from the query graph pattern. For instance, when a client invokes $S_{\mu f}$ with query Q_1 (Listing 1), $S_{\mu f}$ must extract the object of property `dwc:scientificName` (*Delphinus delphis*) to perform the subsequent invocation of Flickr's Web API. This involves reasoning simultaneously on the query graph pattern, the SD document that describes the arguments mappings, and optionally the shapes graph if the mappings refer to property shapes.

Since a SPARQL graph pattern is not represented in RDF, we first translate the incoming query into its SPIN representation [15] that we load into the local triple store as a temporary graph. A major advantage of this approach is that extracting the input arguments can be carried out declaratively within a single SPARQL query rather than in custom code. This query is shown in Listing 5. The first member of the UNION clause (lines 4-11) matches the case where arguments are denoted with `hydra:property`: it retrieves the object of `hydra:property` (line 8), i.e. `dwc:scientificName`, and looks for it in the SPARQL query SPIN graph (line 11). By contrast, the second member (lines 15-34) matches the case where arguments are denoted with a property shape.

Once the arguments have been extracted, the rest of the SPARQL query evaluation is performed as illustrated in section 2.

Implementation. To implement this solution, we deployed Corese [7], an in-memory triple store, as the SPARQL engine underlying SPARQL micro-services. Corese implements the SPARQL Template Transformation Language (STTL) [5] and comes with a built-in STTL SPARQL-to-SPIN transformation. For greater flexibility, our implementation allows passing arguments with VALUES

```
SELECT DISTINCT ?name ?predicate ?value
WHERE {
 {
  # Predicate given with hydra:property
  [] a sd:Service;
   dct:source [ schema:potentialAction [ hydra:mapping [
   hydra:variable ?name;
   hydra:property ?predicate;
  ]]].

  [] sp:predicate ?predicate; sp:object ?value.
 }
 UNION
 {
  # Predicate given through a property shape
  [] a sd:Service;
   dct:source [ schema:potentialAction [
     hydra:mapping [
       hydra:variable ?name;
       shacl:sourceShape ?propShape;
  ]]].

  ?nodeShape a shacl:NodeShape; shacl:property ?propShape.
  ?propShape a shacl:PropertyShape; shacl:path ?predicate.
  OPTIONAL {
   ?nodeShape shacl:property [
    shacl:path rdf:type; shacl:hasValue ?class
   ]
  }

  [] sp:subject ?subject; sp:predicate ?predicate; sp:object ?value.
  OPTIONAL {
   [] sp:subject ?subject; sp:predicate rdf:type; sp:object ?class.
  }
 }
}
```

Listing 5: Extraction of input argument predicates and values (variables `?predicate` and `?value`) from a SPARQL query submitted to a SPARQL micro-service. The default dataset contains the SPARQL query SPIN graph, the shapes graph and the SPARQL SD graph.

or FILTER clauses, which entails a substantially more complicated query than the one depicted in Listing 5. In particular, it leverages the LDScript [6] SPARQL extension to define functions able to parse the nested RDF lists entailed by the VALUES clause[15].

From a more general perspective, the approach we propose considers the service as a coherent, self-contained, reflexive system where RDF and SPARQL are used internally for the service self-description and configuration, at run-time for the query processing, and as the service external interface.

4 WEB-SCALE DISCOVERY OF SPARQL MICRO-SERVICES

In section 1, we suggested that Web search engines can play a key role in enabling the automatic discovery and querying of data sources at Web-scale. Applied to our context, this means that SPARQL micro-services should be published along with a dedicated Web page to be indexed by search engines. Furthermore, major search engines now recommend the inclusion of markup data in Web pages to enhance indexing and consequently yield more accurate results. Therefore, to spur Web-scale discovery while avoiding redundant work, we propose that SPARQL micro-services dynamically transform their service description into Web pages that embed

[15]The complete query is available at https://github.com/frmichel/sparql-micro-service/tree/0.3.1/src/sparqlms/resources/read_input_from_gp.sparql.

rich markup data meant for search engines. Following content negotiation principles, the micro-service URL dereferences to this Web page if it is looked up by a Web browser, while it dereferences to the SPARQL SD document when requested with appropriate RDF media types.

To standardize such markup data, Google, Yahoo, Bing and Yandex support the Schema.org community project that has become a de-facto standard. In particular, Google's recently launched Dataset Search service[16] exploits Schema.org's `Dataset` term[17] as well as equivalent terms from the DCAT W3C recommendation [21]. A Schema.org `Dataset` consists of a set of distributions represented by means of the `DataDownload` object that, unfortunately, is not suited to depict API resources such as SPARQL endpoints. Ongoing discussions are held within the Schema.org community regarding how to annotate a `Dataset` with the interfaces that allow access it[18]. Until a consensus be eventually adopted, a common workaround implemented by the CKAN data portal[19] is to associate to the `DataDownload` object the encoding format "api/sparql". Although semantically questionable ("api/sparql" is not a standard IANA media type[20]), this practice is a trade-off between the need for valid semantic description and the need for effective Web-scale discovery means. Furthermore, given the popularity of CKAN for hosting data portals, this practice tends to spread out.

In the context of SPARQL micro-services, we mitigate this issue with a twofold approach. On the one hand, we comply with the `DataDownload` + "api/sparql" encoding format practice to ensure maximum discoverability. On the other hand, we embed additional DCAT `Dataset` and `Distribution` objects conveying similar information in a more semantically formal manner. Both ways are depicted in Listing 6, lines 24-30 and 38-49 respectively.

Results. The combination of standard content negotiation, semantic Web standards and current Linked Data practices fuels a human-friendly documentation and testing interface on one side and a machine-readable Linked Data description on another side. Furthermore, this combined use pushes "RDF in HTML" descriptions to Web crawlers and indexes in such a way that the described services can be effectively discovered and called by both humans and machines as if they were just another data source.

As an illustration, at the time of writing, the example service $S_{\mu f}$ can be discovered in Google Dataset Search using the keywords "biodiversity" and "photography". Figure 3 shows a snapshot of the result page. Notice that the available download format is appropriately set to SPARQL. Furthermore, adding keyword "sparql" returns the micro-service as the first result in the result page.

Implementation. The Web page generation is performed using the technologies already introduced in section 3.3. An STTL transformation[21] instantiates HTML templates with elements from the SPARQL SD document. The embedded markup data (exemplified in Listing 6) is generated by a SPARQL CONSTRUCT query whose result is passed to a generic built-in STTL transformation that serializes RDF data in JSON-LD. All these transformations are independent of any service and domain. The whole process happens at run-time upon look-up of the micro-service URL. A snapshot of the Web page generated by service $S_{\mu f}$ is displayed in Figure 4, and the reader may access this page by pointing a Web browser at http://sms.i3s.unice.fr/sparql-ms/flickr/getPhotosByTaxon_sd/.

```
1 {
2  "@context": [
3    "http://schema.org",
4    { "dcat": "http://www.w3.org/ns/dcat#",
5      "frmt": "http://www.w3.org/ns/formats/" }
6  ],
7
8  "@type": "Dataset",
9  "identifier":
10  "http://sms.i3s.unice.fr/sparq-ms/flickr/getPhotosByTaxon_sd/",
11 "name": "Search for Flickr photos by scientific name",
12 "description": "...",
13 "publisher": {
14  "@type": "Organization",
15  "name": "Universite Cote d'Azur, CNRS, Inria, I3S",
16 },
17 "keywords": [ "biodiversity", "lifesciences", "photography" ],
18 "isBasedOn": {
19  "@type": "CreativeWork",
20  "@id": "https://www.flickr.com/services/api/",
21  "name": "Flickr Web API"
22 },
23
24 "distribution": {
25  "@type": "DataDownload",
26  "contentUrl":
27    "http://sms.i3s.unice.fr/sparq-ms/flickr/getPhotosByTaxon_sd/",
28  "name": "SPARQL endpoint",
29  "description": "SPARQL micro-service endpoint",
30  "encodingFormat": "api/sparql"
31 },
32
33 "additionalType": [
34  "dcat:Dataset",
35  "http://ns.inria.fr/sparql-micro-service#Service"
36 ],
37
38 "dcat:distribution": {
39  "@type": "dcat:Distribution",
40  "name": "SPARQL endpoint",
41  "description": "SPARQL micro-service endpoint",
42  "dact:accessUrl":
43    "http://sms.i3s.unice.fr/sparq-ms/flickr/getPhotosByTaxon_sd/",
44  "dcat:mediaType": [
45    "frmt:SPARQL_Results_JSON", "frmt:Turtle", "frmt:JSON-LD"
46  ],
47  "http://www.w3.org/ns/shacl#shapesGraph":
48    "http://sms.i3s.unice.fr/sparq-ms/flickr/ \
49       getPhotosByTaxon_sd/ShapesGraph"
50 }
51 }
```

Listing 6: Snippet of the JSON-LD markup data embedded in the automatically generated Web page of SPARQL micro-service $S_{\mu f}$.

5 RELATED WORKS

The work presented in this article addresses two fundamental questions that have been studied under many different perspectives: capturing the functionality of Web services on one side, and automating their discovering and consumption by software agents on the other side.

Works about semantic Web services, whether "big" WSDL-based (e.g. OWL-S [4], SAWSDL [8]) or REST-based (e.g. WADL [13]), have long tackled the question of capturing the functionality of a service through the semantic description of their inputs, outputs and the way they relate to one another. These models, that

[16] Google Dataset Search: http://g.co/datasetsearch
[17] Schema.org Dataset: https://schema.org/Dataset
[18] https://github.com/schemaorg/schemaorg/issues/1423. Accessed Jan. 30th 2019.
[19] The CKAN project: https://ckan.org/
[20] IANA media types: https://www.iana.org/assignments/media-types/
[21] The whole transformation code is available at: https://frama.link/hBDkM7ep.

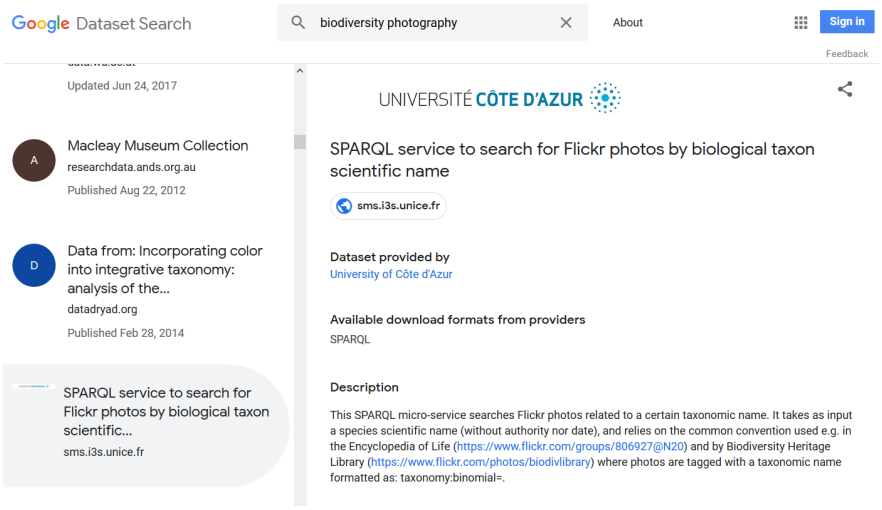

Figure 3: Google Dataset Search result page showing the example SPARQL micro-service $S_{\mu f}$. Clicking on the *sms.i3s.unice.fr* link opens the micro-service Web page depicted in Figure 4.

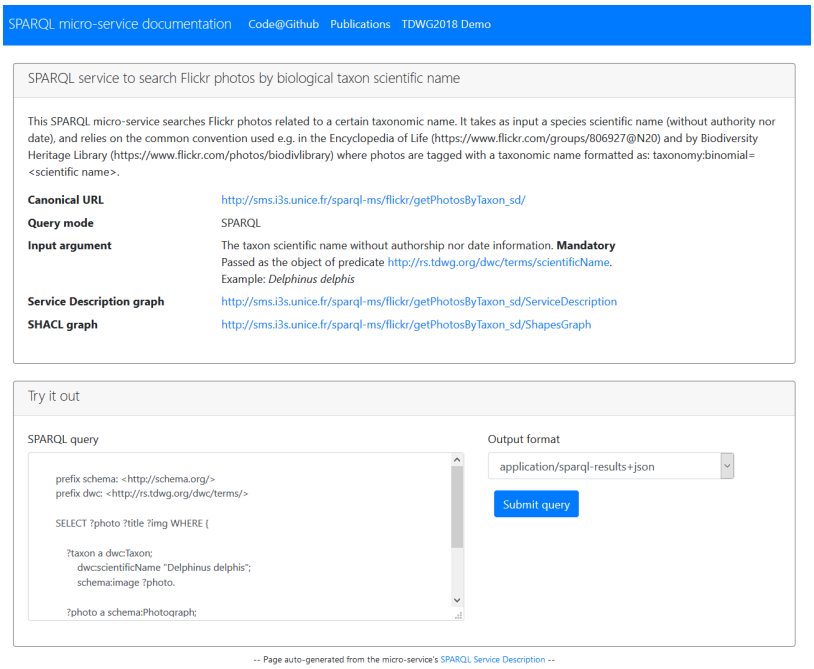

Figure 4: Web page automatically generated from the description of SPARQL micro-service $S_{\mu f}$.

support automatic discovery, invocation and composition of Web services, usually entail the deployment of complex frameworks requiring advanced skills and tooling. Besides, service discovery is made possible using a centralized repository such as the Universal Description Discovery and Integration (UDDI) registry[22]. As a consequence, they are better suited to the controlled environment of companies [20] than the open, loosely constrained environment of the Web that we wish to address.

By contrast, Web APIs are quite simple to deploy and interact with. Still, it is hardly possible to discover and invoke them automatically insofar as they commonly rely on proprietary vocabularies described in Web-based documentation with little concern for semantic interoperability. To fulfill this lack, some initiatives seek to enrich existing human-readable documentation of Web APIs with markup data so as to make it machine-processable. They rely on

[22]UDDI specification: http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

microformats (e.g. hRESTS [17]) possibly joined to existing service ontologies (e.g. MicroWSMO [9]), or RDFa (e.g. SA-REST [10]). These methods are however more concerned with describing the service interface (operations, parameter types) than its actual functionality. Indeed, the description of the resources manipulated is delegated to domain ontologies that provide terms for classes and properties, but often put little constraints on how to use them. By contrast, we harness SHACL specifically to address this lack. SHACL can describe rich constraints on what can be stated, thus making it possible to specify in a comprehensive manner how resources relate to each other.

OpenAPI[23] takes the problem the other way round: it equips a Web APIs with a machine-readable documentation, that, in turn, can be compiled into a Web page. This is closer to our approach, yet, this description remains at a syntactic level essentially enabling the automatic generation of server- and clients-side stubs, very similar to what WSDLs enabled for "heavy" Web services.

Linked REST APIs (LRA) [29] is a framework dedicated to the semantic annotation of Web APIs and the automatic specification of SPARQL query execution plans that invoke these Web APIs. The framework relies on a centralized repository that stores the Web APIs descriptions and offers search services. Several key differences with our work can be pointed out. With SPARQL micro-services, we seek to set up a totally distributed architecture wherein independent service providers may publish SPARQL micro-services that can be discovered using regular Web search engines, rather than a centralized repository. Furthermore, LRA describes a Web API by means of a custom vocabulary and relies on a SPARQL graph pattern to serve as a functional description. To spur large adoption, we instead stick to standard vocabularies, and we use SHACL to describe resources as it allows for more expressiveness than a sheer SPARQL graph pattern.

RESTdesc [32] is a semantic description format for hypermedia APIs. It captures the functional description of APIs in Notation3 [3], a language extending RDF's data model with variables, existential and universal quantifiers, and logical implications. RESTdesc relies on the HTTP mechanisms and RESTful principles for the discovery and invocation of semantically described Web services. Starting from a known URI, an application can follow its nose by resolving links and making sense of Notation3 service descriptions. This is an elegant solution that however requires Notation3 reasoners able to interpret the advanced features of quantification and logical implications. Such reasoners exist but are far less common than SPARQL-based implementations available in many programming languages. Since we seek a solution that can be adopted easily by a large community of independent actors, leveraging more common standards such as regular RDF and SPARQL is probably more promising.

In line with our idea of leveraging Web search engines to discover relevant datasets and query services, SpEnD [34] is a metacrawler designed to discover SPARQL endpoints. It first creates a list of keywords commonly found on Web pages advertising SPARQL endpoints, such as the pages of DataHub. It then looks for these keywords on search engines, explores the result Web pages looking for SPARQL endpoint URLs and looks up these URLs in search for VoID or SPARQL SD documents. This kind of approach is clearly what could be implemented to discover SPARQL micro-services at Web scale. We believe that the usage of well-adopted markup data could help enhance search results and, in this respect, dataset-search services such as Google Dataset Search could be more effective than generic Web search engines.

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

6 CONCLUSION AND PERSPECTIVES

In this article, we address the problem of enabling automatic discovery and consumption of data sources at Web scale. We suggested that three principles should be considered to pursue this goal: (1) provision rich descriptions of data sources and query services, (2) leverage the power of Web search engines to discover data sources, and (3) rely on simple, well-adopted standards that come with extensive tooling. We applied these principles to the concrete case of SPARQL micro-services that aim at querying Web APIs using SPARQL. The proposed solution considers a SPARQL Service Description (SD) document as the description central point. It links to a SHACL shapes graph describing precisely the resources manipulated by the micro-service. It also connects the resources to the micro-service inputs, thereby coming up with a rich functional description that allows a software agent to decide whether this micro-service can help in carrying out a certain task. To enable accurate discovery using common Web crawlers, the SD document can be dynamically transformed into a Web page embedding rich markup data based on Schema.org's Dataset term and the DCAT vocabulary.

From a general perspective, the combination of standard content negotiation, semantic Web standards and Linked Data practices fuels a human-friendly documentation and machine-readable description that make it possible for humans and machines alike to discover and invoke SPARQL micro-services as if they were just another data source.

We showed that our approach is effective as our example SPARQL micro-service can be successfully discovered using the Google Dataset Search engine (as illustrated in Figure 3). From this point on, a framework such as SpEnD (described in section 6) could be extended to accommodate the invocation of SPARQL micro-services. Service composition-based query answering systems could fetch the shapes graphs of candidate SPARQL micro-services, check the compatibility of their inputs and outputs with respect to the query to process, and finally compute and enact valid compositions. In particular, SPARQL query federation is a specific type of Web service composition wherein any piece of data in the federated graphs may play the role of either an input or an output. Existing federated query engines could be extended so as to reason on the description of SPARQL micro-services and come up with query plans that respect SPARQL micro-services' input requirements.

As pointed out in section 4, denoting a SPARQL endpoint using Schema.org terms is still quite unpractical at the moment. From a more general perspective, describing the multiple interfaces that a client may use to access a dataset is an increasingly pressing need. The Schema.org community is currently thinking this through with discussions revolving around the `Dataset`, `WebAPI` and `EntryPoint` terms. Concomitantly, the DCAT community is working out the next version of the W3C DCAT recommendation [11] that defines

the generic concept of `DataService` meant to serve dataset distributions. The term is flexible enough to accommodate various types of interfaces, providing notably a contract the interface conforms to and an out-of-band description that may typically be a SPARQL SD document in our context.

In the current state of our work, SHACL graphs are used as a specification of the graphs that a SPARQL micro-service can generate. We can think of two interesting leads for future works in this respect. Firstly, once a shapes graph is published with its own dereferenceable URI, it can be reused by SPARQL micro-services providers, thereby sparing time and making it possible to share common practices. A second lead could be to consider SHACL as a way for a client to request responses in a certain shape. This would amount to some sort of extended content negotiation where a client could express that it would prefer a response not only favoring a vocabulary over another, but also describing resources and their relationships according to a certain shape, as much as possible.

Finally, whether our approach succeeds in reaching principle (3) (rely on well-adopted standards) is debatable and possibly a matter of perspective and community. Some people contend that Semantic Web standards are not likely to be largely adopted by Web developers [18] due to the perceived complexity of RDF and SPARQL, as compared to RESTful APIs for instance. Besides, SHACL is a rich language, yet perhaps too rich to gain large adoption. In the end however, we do believe that there will be room for different types of interfaces, suited to different contexts and scenarios. This article primarily intends to propose a research direction, not a ready-to-use solution. And we encourage the interested readers to explore alternative architectural and modeling choices.

REFERENCES

[1] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. 2011. Describing Linked Datasets with the VoID Vocabulary. *W3C Recommendation* (2011).

[2] Mohamed Ben Ellefi, Zohra Bellahsene, John G. Breslin, Elena Demidova, Stefan Dietze, Julian Szymański, and Konstantin Todorov. 2018. RDF Dataset Profiling – a Survey of Features, Methods, Vocabularies and Applications. *Semantic Web – Interoperability, Usability, Applicability* 9, 5 (2018), 677–705. https://doi.org/10.3233/SW-180294

[3] Tim Berners-Lee and Dan Connolly. 2011. Notation3 (N3): A Readable RDF Syntax. *W3C Team Submission* (March 2011).

[4] Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. 2004. OWL-S: Semantic Markup for Web Services. *W3C Member Submission* (2004).

[5] Olivier Corby and Catherine Faron-Zucker. 2015. STTL: A SPARQL-Based Transformation Language for RDF. In *11th International Conference on Web Information Systems and Technologies (ISWC)*. Lison, Portugal.

[6] Olivier Corby, Catherine Faron-Zucker, and Fabien Gandon. 2017. LDScript: A Linked Data Script Language. In *Proceedings of the 16th International Semantic Web Conference (ISWC)*. Springer, Vienna, Austria, 208–224.

[7] Olivier Corby and Catherine Faron Faron-Zucker. 2010. The KGRAM Abstract Machine for Knowledge Graph Querying. In *Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*. IEEE, 338–341.

[8] Joel Farrell and Holger Lausen (Eds.). 2007. Semantic Annotations for WSDL and XML Schema. *W3C Recommendation* (2007).

[9] Florian Fischer and Barry Norton. 2009. *MicroWSMO v2 – Defining the Second Version of MicroWSMO as a Systematic Approach for Rich Tagging*. Technical Report.

[10] Karthik Gomadam, Ajith Ranabahu, and Amit Sheth. 2010. SA-REST: Semantic Annotation of Web Resources. *W3C Member Submission* (2010).

[11] Alejandra Gonzalez Beltran, Dave Browning, Simon Cox, and Peter Winstanley. 2019. Data Catalog Vocabulary (DCAT) - revised edition. *W3C Editor's Draft 01 February 2019* (2019). https://w3c.github.io/dxwg/dcat/ Accessed: 2019-02-01.

[12] R. V. Guha, Dan Brickley, and Steve MacBeth. 2015. Schema.Org: Evolution of Structured Data on the Web. *ACM Queue - Strutured Data* 13, 9 (2015), 10:10–10:37. https://doi.org/10.1145/2857274.2857276

[13] Marc Hadley. 2009. Web Application Description Language. *W3C Member Submission* (2009).

[14] Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 Query Language. *W3C Recommendation* (2013). http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[15] Holger Knublauch. 2013. SPIN - SPARQL Syntax. *W3C Member Submission* (2013).

[16] Holger Knublauch and Dimitris Kontokostas (Eds.). 2017. Shapes Constraint Language (SHACL). *W3C Recommendation* (2017).

[17] Jacek Kopecky, Karthik Gomadam, and Tomas Vitvar. 2008. hRESTS: An HTML Microformat for Describing RESTful Web Services. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI-IAT)*. 619–625. https://doi.org/10.1109/WIIAT.2008.379

[18] Markus Lanthaler and Christian Gütl. 2011. A Semantic Description Language for RESTful Data Services to Combat Semaphobia. In *Proceedings of the IEEE International Conference on Digital Ecosystems and Technologies*. 47–53. https://doi.org/10.1109/DEST.2011.5936597

[19] Markus Lanthaler and Christian Gütl. 2013. Hydra: A Vocabulary for Hypermedia-Driven Web APIs. In *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW2013)*. CEUR-WS.

[20] Moussa Lo and Fabien Gandon. 2007. Semantic Web Services in Corporate Memories. In *Proceedings of the Second International Conference on Internet and Web Applications and Services*. https://doi.org/10.1109/ICIW.2007.59

[21] Fadi Maali, John Erickson, and Phil Archer. 2014. Data Catalog Vocabulary (DCAT). *W3C Recommendation* (Jan. 2014).

[22] Franck Michel, Catherine Faron-Zucker, and Fabien Gandon. 2018. SPARQL Micro-Services: Lightweight Integration of Web APIs and Linked Data. In *Proceedings of the Linked Data on the Web Workshop (LDOW2018)*, Vol. 2073. CEUR, Lyon, France, 10.

[23] Franck Michel, Catherine Zucker, Olivier Gargominy, and Fabien Gandon. 2018. Integration of Web APIs and Linked Data Using SPARQL Micro-Services—Application to Biodiversity Use Cases. *Information* 9, 12 (Dec. 2018), 310. https://doi.org/10.3390/info9120310

[24] Sam Newman. 2015. *Building Microservices*. O'Reilly Media.

[25] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. 2008. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In *Proceedings of the 17th International World Wide Web Conference (WWW2008)*. Beijnig, China, 805–814. https://doi.org/10.1145/1367497.1367606

[26] Emmanuel Pietriga, Hande Gözükan, Caroline Appert, Marie Destandau, Šejla Čebirić, François Goasdoué, and Ioana Manolescu. 2018. Browsing Linked Data Catalogs with LODAtlas. In *Proceedings of the 17th International Semantic Web Conference (ISWC)*. Monterey, USA, 17.

[27] Mike Ralphson and Ivan Goncharov. 2017. WADG0001 WebAPI type extension. *Draft Community Group Report* (2017). https://webapi-discovery.github.io/rfcs/rfc0001.html

[28] Andy Seaborne. 2013. SPARQL 1.1 Query Results JSON Format. *W3C Recommendation* (2013).

[29] Diego Serrano, Eleni Stroulia, Diana Lau, and Tinny Ng. 2017. Linked REST APIs: A Middleware for Semantic REST API Integration. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*. IEEE, Honolulu, HI, USA, 138–145. https://doi.org/10.1109/ICWS.2017.26

[30] Manu Sporny, Dave Longly, Gregg Kellog, Markus Lanthaler, and Niklas Lindström. 2014. JSON-LD 1.0. A JSON-based Serialization for Linked Data. *W3C Recommendation* (2014). https://www.w3.org/TR/2014/REC-json-ld-20140116/

[31] Pierre-Yves Vandenbussche, Jürgen Umbrich, Luca Matteis, Aidan Hogan, and Carlos Buil-Aranda. 2017. SPARQLES: Monitoring Public SPARQL Endpoints. *Semantic Web* 8, 6 (Aug. 2017), 1049–1065. https://doi.org/10.3233/SW-170254

[32] Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Jos De Roo, Rik Van de Walle, and Joaquim Gabarro Vallés. 2011. Description and Interaction of RESTful Services for Automatic Discovery and Execution. In *Proceedings of the International Workshop on Advanced Future Multimedia Services*. Future Technology Research Association International (FTRA), Jeju, South Korea.

[33] Gregory Tood Williams (Ed.). 2013. SPARQL 1.1 Service Description. *W3C Recommendation* (2013).

[34] Semih Yumusak, Erdogan Dogdu, Halife Kodaz, Andreas Kamilaris, and Pierre-Yves Vandenbussche. 2017. SpEnD: Linked Data SPARQL Endpoints Discovery Using Search Engines. *IEICE Transactions on Information and Systems* E100.D, 4 (2017), 758–767. https://doi.org/10.1587/transinf.2016DAP0025