Declaratively Describing Responses of Hypermedia-Driven Web APIs

Ruben Taelman Ruben Verborgh

IDLab, Department of Electronics and Information Systems, Ghent University – imec

ABSTRACT

While humans browse the Web by following links, these *hypermedia* links can also be used by machines for browsing. While efforts such as Hydra semantically describe the hypermedia controls on Web interfaces to enable smarter interface-agnostic clients, they are largely limited to the input parameters to interfaces, and clients therefore do not know what response to expect from these interfaces. In order to convey such expectations, interfaces need to declaratively describe the response structure of their parameterized hypermedia controls. We therefore explored techniques to represent this parameterized response structure in a generic but expressive way. In this work, we discuss four different approaches for declaring a response structure, and we compare them based on a model that we introduce. Based on this model, we conclude that a SHACL shape-based approach can be used for declaring such a parameterized response structure, as it conforms to the REST architectural style that has helped shape the Web into its current form.

CCS CONCEPTS

• **Information systems~Web interfaces • Information systems~RESTful web services • Information systems~Resource Description Framework (RDF)** • Information systems~Service discovery and interfaces • Information systems~Web Ontology Language (OWL)

KEYWORDS

Linked Data, Hypermedia, REST, RDF, SHACL, Hydra

ACM Reference Format:

Taelman, R. and Verborgh, R. 2017. Declaratively Describing Responses of Hypermedia-Driven Web APIs. K-CAP 2017: K-CAP 2017: Knowledge Capture Conference, December 4 — 6, 2017, Austin, TX, USA, 4 pages.
DOI: 10.1145/3148011.3154467

1 INTRODUCTION

Humans can browse the Web by following *links* from one page to another. This *human* interface is only one of the possible Web interfaces that exist. Next to humans, machines also heavily make use of the Web through Web Application Programming Interfaces (Web APIs). Two architectural styles for Web APIs can be distinguished, where elements of them are sometimes combined in practise. First, some APIs are based on the concept of Remote Procedure Calling (RPC), in which HTTP requests correspond to procedure or method calls of internal programs. Second, other APIs are based on Representational State Transfer (REST), in which HTTP resources are linked and described to each other, similar to how the human web works. In the case of pure RESTful APIs, *hypermedia controls* declaratively instruct clients on how they can use an interface.

K-CAP 2017, December 4–6, 2017, Austin, TX, USA

ACM. ACM 978-1-4503-5553-7/17/12…$15.00
https://doi.org/10.1145/3148011.3154467

An advantage of REST over RPC is that these hypermedia controls are self-descriptive, and can be reused across different interfaces. Once they are implemented, clients can automatically interact with interfaces through these self-descriptive hypermedia controls without having to refer to external documentation. However, self-descriptiveness is a *relative* notion: depending on the set of primitives supported by a client, an interface exposed by a server might or might not describe itself.

According to the Linked Data principles [2], HTTP URIs should be used to identify concepts on the Semantic Web, which can be seen as *the Web for machines*. As the RDF [3] data model uses URIs as primary data element, hypermedia controls can be encoded using this model, so that machines can use and understand them. Amundsen identifies nine *"Hypermedia Factors"* [4] to identify hypermedia behaviors. While RDF natively supports the *outbound links* hypermedia factor, it provides no support for more advanced *templated links*. The latter corresponds to HTML forms on Web pages, such as a form for searching books through a library's website. One part of the Hydra Core vocabulary [5] attempts to fill this gap by representing HTML controls as Linked Data for machines.

The Hydra Core vocabulary is for example used in the Triple Pattern Fragments (TPF) [6] interface for describing a query form. TPF interfaces expose hypermedia controls that afford triple pattern queries on top of certain datasets. This allows clients to consume data from datasets that are exposed behind TPF interfaces using these self-descriptive controls, as shown in Listing 1. In this example, the `hydra:search` predicate is used to link a search form to a dataset. This search form has an IRI template string which allows certain variables to be filled in. These variables are declared in the range of `hydra:mapping`, which are in this case `'s'`, `'p'` and `'o'`, which respectively are an RDF [3] subject, predicate and object.

```
<http://fragments.dbpedia.org/2014/en#dataset> hydra:search [
    hydra:template "http://fragments.dbpedia.org/2014/en{?s,p,o}";
    hydra:mapping [ hydra:variable "s"; hydra:property rdf:subject ],
                  [ hydra:variable "p"; hydra:property rdf:predicate ],
                  [ hydra:variable "o"; hydra:property rdf:object ]
].
```

Listing 1: Declarative triple pattern query control on the DBpedia TPF interface using the Hydra Core Vocabulary in the TriG syntax.

Without the Hydra Core Vocabulary, clients would need a hard-coded API contract. With this markup, clients can automatically understand that by providing a certain set of parameters, a certain request to the API can be made. However, the Hydra Core Vocabulary is not capable of describing the link's *control data* [4] on *how* these parameters will be used, i.e., what kind of response will be returned based on the given request. In order to reach smarter clients, they also need to know in what way the parameters will contribute to the response. This would not only allow clients to derive *what* parameters are used for a certain request, but also *how* these parameters form the response.

In the case of TPF for example, the subject, predicate and object IRI parameters are described, which make up a triple pattern. It is however nowhere described that the interface necessarily returns all triples in the dataset that *match* with this pattern. The server could for example instead return all triples in the dataset that do *not match* with this pattern, or return the *lexicographical ordering* of the given parameter values, as there is no provided method for distinguishing between these different behaviours with the same input parameters.

In this article, we introduce and compare different approaches for describing the responses of such hypermedia-driven API responses. We discuss approaches based on custom vocabularies, the recent W3C recommendation SHACL [7], the SPIN modeling vocabulary [8], and the Web Ontology Language (OWL) [9].

2 RELATED WORK

In this section, we introduce the related work on Web APIs and Web Services, followed by an overview of technologies for defining constraints in RDF.

2.1 Web APIs And Services

Next to the REST architectural style, the SOAP protocol is often used for letting *Web Services* interoperate. Just, like RPC, SOAP requires custom client-side implementation for each Web Service, which leads to tighter coupling between servers and clients, as opposed to the more generic REST APIs. OWL-S [10] is a vocabulary for describing such Web Services. It allows services to declare their actions, how they can be used, and how they work.

Verborgh et al. distinguish two types of Web APIs [11] in terms of how they expose their functionality. The first type, which are mostly used today, is the *top-down* Web API. This kind of API exposes certain functionality through a *single* interface, and requires clients to understand this specific interface. The second type is the *bottom-up* Web API, where an API exposes different functionalities as different *features*, where each feature should describe its own *functionality*. The second kind of API leads to clients that are not bound to specific providers, but to specific reusable features. The concept of declaring the response structure is in line with these principles of feature-based interfaces.

2.2 RDF Constraints

SHACL is a recent W3C recommended vocabulary that allows RDF shapes to be defined and composed for constraint checking and validation. The SPIN vocabulary [8] can be seen as the predecessor to SHACL for specifying rules and constraints. It is more lightweight than SHACL, but thereby also less expressive. The SPIN vocabulary is based on the SPARQL query language for defining these constraints, where triple patterns can be composed as graph patterns, which in turn can be composed as more complex graph patterns. Alternatively, OWL [9] and RDF Schema (RDFS) [12] could be used to define constraints on certain targets. The main difference between SHACL and OWL is that SHACL works under the *closed world assumption*, while OWL works under the *open world assumption*. In practise, the latter makes data validation more complex, which is part of the motivation for SHACL's creation.

3 MODEL FOR COMPARISON

In this section, we introduce a model for comparing approaches for declaratively representing Web API responses. Our model consists of different criteria that can influence the choice of a certain approach: *RDF complexity*, *expressivity*, *composability*, *discoverability* and *adoptability*. These will be explained hereafter. Each representation approach can receive a qualitative score for each of these criteria. A suitable approach can then be chosen based on the composite score across these criteria, which can possibly be weighted depending on the relative importance of these criteria in the use case.

3.1 RDF Complexity

The level of RDF complexity, i.e., how 'deep' the response structure is represented in RDF, has an influence on how easily such a representation can be used by RDF-based tools.

A response that is based on a SPARQL query could for example be represented as an RDF string literal, or fully reified using the SPIN vocabulary. Both approaches have the same meaning, but the former representation requires less effort to represent in RDF, while the latter provides better compatibility with RDF-based tools, such as reasoners and query engines. If a subset of such a SPARQL query needs to be taken, an RDF reasoner or query engine can more easily do this using the reified RDF representation than the string literal.

3.2 Expressivity

The expressivity of an approach for response declaration of an interface corresponds to the range of responses that can be declared using this approach.

According to the REST principles and the layered architectural style, clients should require no prior knowledge about interface functionality except for the agreed-upon primitives. Simple and few primitives on the one hand lower the barrier for client support, such as the eight well known HTTP [13] methods on the protocol level. These primitives should however be sufficiently expressive, as to allow more advanced operations to be defined on top of them. Many complex and expressive primitives on the other hand make it more complex for clients to support them, but when they are supported, complex operations can more easily be interpreted by clients.

In the case of declaring interface responses, vocabularies can exist at different levels of expressivity. One vocabulary may for example

only enable simple triple pattern queries to be represented, which may be simple for a client to parse and handle, but is not very expressive. Another vocabulary may enable full SPARQL queries to be represented, which may be more complex for a client to use, but is much more expressive. By following the REST principles, a well-defined restricted vocabulary should be agreed upon using which, potentially complex, response types can be declared.

3.3 Composability

As in a feature-based interface [11], response structures can be made up of multiple smaller reusable components that can be *composed* and *extended*. This allows clients to only be required to understand these smaller components, and this could allow more complex composed response types to be interpreted automatically. Different techniques can lead to different levels of composability.

A certain Web API could for example return a list of people based on a certain query. Another similar Web API could annotate all these people with their place of birth, which can be seen as an *extension* to the first API, or a *composition* of the 'query' feature and the 'annotation' feature. Another composition could for example apply some kind of *sorting* or *filtering* feature, possibly based on certain parameters.

3.4 Discoverability

Certain techniques for declaring response structure are more easily *discoverable* than others, meaning that based on the technique, clients may require more or less effort for *finding* and *interpreting* the response structure. A single text-based identifier for a reponse structure could for example be very simple for clients to detect, while a reference to the source code that is used to control the Web API requires much more effort from the client.

3.5 Adoptability

While different approaches for Web API response types exist, the used technologies behind these approaches will have an impact on the adoption rate. The usage of a new, non-standard vocabulary will most likely lead to a lower adoption rate than the usage of vocabulary that is recommended by W3C.

4 APPROACHES FOR DECLARATIVE RESPONSE DESCRIPTION

In this section, we discuss and compare different approaches for declaratively describing the responses of Web APIs based on given parameters. We will use the TPF use case as a running example. For this, we will extend from the hypermedia control from Listing 1, which currently describes the interface input parameters, to describe the responses to triple pattern queries.

The four approaches that will be explained hereafter are Custom types, SHACL shapes, SPIN SPARQL queries and OWL restrictions. For each approach, we will provide a score for the model criteria from Section 3, which are summarized in Table 1.

Criterion	Custom Types	SHACL	SPIN	OWL
RDF Complexity	○	◑	◉	◉
Expressivity	◯	◉	◑	◯
Composability	◯	◉	◉	◉
Discoverability	◉	◑	◑	◑
Adoptability	○	◉	◑	◉

Table 1: Qualitative scores (very low ○, low ◯, medium ◑, high ◉) for three different approaches for declaring interface responses based on the model from Section 3.

4.1 Custom Types

A simple solution is to define a new response type at vocabulary-level for each hypermedia control type that exists. For our use case, we could introduce a (hypothetical) `tpf:TriplePatternQueryResponse` type in a new `tpf` vocabulary, which refers to a triple pattern query, as shown in Listing 2.

```
<http://fragments.dbpedia.org/2014/en#dataset> hydra:search [
    ...
    ex:responseType tpf:TriplePatternQueryResponse.
].
```

Listing 2: Triple pattern query response declaration using a custom type, with `ex:responseType` referring to this type.

The advantages of this approach are that it is very simple to set up, and is easily *discoverable*. However, it has some significant disadvantages. For one, as each response type requires a separate RDF type, and clients need explicit support for a potentially huge number of types. Instead of having small reusable functional building blocks, new types have to be defined for each interface that offers different functionality. As opposed to indirect hierarchical types as is the case with MIME types [14], RDF enables more explicit basic composition of types by attaching multiple types and subclassing.

4.2 SHACL Shapes

The SHACL vocabulary is designed for defining shape constraints to validate RDF graphs against, which allows us to describe the shape of our responses. In our TPF use case, we could make our search form a parameterizable shape, and declare the triple pattern query as a SPARQL [15] `SELECT` query, as shown in Listing 3.

The interface input parameters and the response shape parameters are declared separately. The former is defined using Hydra, while the latter is defined using SHACL. As these parameters are—and should always be—equal for allowing output to be fully defined using input, one of the two methods could be deprecated in favor of the other. The Hydra variables are simpler, but also less expressive. SHACL parameters are much more expressive because they are also SHACL property shapes, which means that the full expressivity of SHACL constraints can be used on these parameter values. SHACL parameter names are, however, not defined in the same way as Hydra variable names. Hydra allows variable names to be set using the `hydra:variable` predicate. Instead, SHACL parameter names are derived from the IRI in `sh:path`.

```
<http://fragments.dbpedia.org/2014/en#dataset> hydra:search [
    ...
    sh:parameter [ sh:path ex:subject;   sh:order 0; sh:nodeKind sh:BlankNodeOrIRI ],
                 [ sh:path ex:predicate; sh:order 1; sh:nodeKind sh:IRI ],
                 [ sh:path ex:object;    sh:order 2 ];
    sh:select """ SELECT  ?subject ?predicate ?object
                  WHERE { ?subject ?predicate ?object } """.
].
```

Listing 3: Triple pattern query response declaration using a SHACL shape that is a subclass of `sh:Parameterizable` and `sh:SPARQLSelectExecutable`.

In summary, SHACL shapes are very expressive for declaring responses of Web APIs. Furthermore, the expressivity from the SPARQL query language and JavaScript are inherited thanks to the SHACL extensions SHACL-SPARQL and SHACL-JS.

4.3 SPIN SPARQL Queries

The SPIN vocabulary [8] allows SPARQL queries to be defined in a triple representation, which SHACL does not support. The advantage of triple-based representations over text-based is that RDF-based tools can directly use and work with these structures, such as reasoners and query engines. The disadvantage of triple-based representations is their verbosity compared to a text-based variant.

As our TPF use case requires the representation of a triple pattern query, we can again trivially represent this as a SPARQL SELECT query using SPIN, as shown in Listing 4.

```
<http://fragments.dbpedia.org/2014/en#dataset> hydra:search [
    ...
    a sp:Select;
    sp:resultVariables ( ex:subject, ex:predicate, ex:object );
    sp:where ([
        sp:subject   ex:subject;
        sp:predicate ex:predicate;
        sp:object    ex:object
    ]).
].
```

Listing 4: Triple pattern query response declaration using a SPARQL SELECT query using the SPIN vocabulary.

As mentioned before, the SPIN-based query in Listing 4 is indeed more verbose than the SHACL-SPARQL SELECT query from Listing 3. That is because SPIN reifies triple patterns, which leads to a large amount of triples, even for simple queries.

As our subject, predicate and object variables are now represented as actual resources, they are explicitly linked with the Hydra variables, which is a semantic advantage.

4.4 OWL Restrictions

While OWL [9] allows restrictions to be placed on RDF graphs, it can not do this at the same level of expressivity as SHACL. Furthermore, the open world assumption on which OWL is based makes it more difficult to describe the closed world of Web API responses.

Our TPF use case can for instance not be represented using OWL restrictions, because OWL restrictions work on class structures, but not on triple structures. That is because OWL has been designed for *inferencing*, and not for defining arbitrary constraints on RDF graphs. Other operations such as type restrictions on instances, or defining the cardinality of certain aspects are however possible.

5 DISCUSSION

Based on our model from Section 3, there are different ways for choosing between the discussed approaches from Section 4. There is no one approach that is the *winner* across all criteria: an appropriate approach depends on the situation. In this section, we discuss the arguments for preferring certain criteria over others, and which approaches may be best suited in certain situations.

In many cases, the *adoption* rate would be of great importance, unless the approach would be used in a closed environment, which makes declarative response types less useful in the first place. According to the best practises for publishing Linked Data [16], standard vocabularies –such as SHACL– should be reused as much as possible, because this helps with the inclusion in the Web of data and these typically have better tool support.

Responses should be declared at a sufficient level of *expressivity*, as long as clients are able to interpret them. Chances of this are higher when standard vocabularies are used, because of the better tool support. Therefore, adoption rate is typically prioritized over expressivity. SHACL for instance provides a higher level of expressivity than OWL, while they are both standard vocabularies.

A response declaration can be defined in an expressive way, but as long as clients are not able to *discover* it, it is of not much use. Custom types are for example very easily discoverable. More expressive response structures may be more complex and difficult to discover, which is why a trade-off between those two criteria exists.

The *composability* criterion is related to expressivity and discoverability. If an approach allows very few and simple building blocks to be combined to reach a high level of expressivity, clients require less hard-coded support for these building blocks, which benefits the discoverability. Approaches that do not allow composability will require more of these building blocks to achieve a high level of expressivity, which can negatively impact discoverability. The composability of the approaches based on SHACL, SPIN and OWL is for instance much higher than that of custom types, because the latter have no standard building blocks.

When RDF-based tools are required for handling the response declarations, *RDF complexity* is important. Nevertheless, even where non-RDF representations are used, RDF-based tool processing could still be done by reifying to RDF in a postprocessing step. SPIN and OWL are examples where responses can be expressed purely in RDF, while SHACL-SPARQL encodes SPARQL queries as RDF literals. The RDF complexity is related to the composability of an approach, as small building blocks that are defined in granular RDF statements, can potentially be reused as part of other more complex and possibly more expressive declarations. A high level of RDF complexity can however negatively impact discoverability due to the higher required interpretation effort.

As mentioned before, choosing an appropriate approach depends on the situation. For instance, if we require an approach that is based on standards, is expressive, discoverable and is made up of easily composable building blocks, then the SHACL shapes approach is best suited. But if we require an approach that has a high level of RDF complexity and is sufficiently expressive, then the SPIN-based approach could be sufficient. Otherwise, if we value discoverability over adoptability, then custom types might be preferred.

6 CONCLUSIONS

Much of the Web's current hypermedia control response types are left to the client's interpretation, which could be done using custom types. This approach is however hard to sustain, as it leads requires new types for each new response type, and leads to tight client-server coupling. If we want to have sustainable and declarative hypermedia response definitions on the Web, a technique is required that revolves around standards with an adequate level of expressivity and composability, but is not too difficult for clients to work with.

The SHACL-based approach that we introduce in this work adheres to these requirements. It allows implementing the self-descriptive message constraint of the REST style in a sustainable way across APIs. It is based on simple building blocks that make it easy for clients to discover and interpret them, and these building blocks can be combined for reaching a higher level of expressivity. Furthermore, as SHACL is a W3C recommendation, it can lead to a higher adoption rate. Practical usage of this approach is already possible without any new vocabularies. If servers expose the *shape* of their control responses, clients that *understand* SHACL and Hydra can interpret this to determine if this control is useful for them.

A response declaration can be seen as the server's *suggested* way of consuming the data behind a control, but not necessarily the only way. Profile-based negotiation [17] on controls can provide *multiple dimensions* on how this data can be consumed, by allowing clients to ask the server for returning the data in a different *application profile*, which may be more convenient for the client to work with.

With such a hypermedia control extension, clients are able to known not only what the *input* of a control is, but also what kind of data is returned as *output* based on certain input. This information is vital for any kind of operation, just like typed programming languages require declaration of both the input and output types of operations. This extension will enable autonomous clients to work with such controls in a better way, as they will be able to know what output to expect from a certain control. This will for instance allow clients to select only the controls that will produce the response structure that the client can work with, or start preparing a response data structure *before* the actual control response is even received. Nevertheless, future work is still required in this direction, to empirically validate the practical feasibility of this approach. Furthermore, algorithms will need to be developed to efficiently generate such declarations server-side, and to work with them client-side.

ACKNOWLEDGEMENTS

We would like to thank Holger Knublauch for providing his insights into the SHACL declarations. The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), and the European Union. Ruben Verborgh is a postdoctoral fellow of the Research Foundation – Flanders.

REFERENCES

[1] Fielding, R.T., Taylor, R.N.: Architectural styles and the design of network-based software architectures. University of California, Irvine Doctoral dissertation (2000).

[2] Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts. 205–227 (2009).

[3] Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: Concepts and Abstract Syntax. W3C, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (2014).

[4] Amundsen, M.: Hypermedia Types. In: REST: From Research to Practice. pp. 93–116. Springer (2011).

[5] Lanthaler, M., Gütl, C.: Hydra: A Vocabulary for Hypermedia-Driven Web APIs. LDOW. 996, (2013).

[6] Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web. Journal of Web Semantics. 37–38, (2016).

[7] Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL). W3C, https://www.w3.org/TR/2017/REC-shacl-20170720/ (2017).

[8] Knublauch, H.: SPIN - Modeling Vocabulary. W3C Member Submission. 22, (2011).

[9] Group, W.C.O.W.L.W.: OWL 2 Web Ontology Language. W3C, https://www.w3.org/TR/2012/REC-owl2-overview-20121211/ (2012).

[10] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., others: OWL-S: Semantic markup for web services. W3C, https://www.w3.org/Submission/owl-s/ (2004).

[11] Verborgh, R., Dumontier, M.: A Web API ecosystem through feature-based reuse. CoRR. abs/1609.07108, (2016).

[12] Brickley, D., Guha, R.V.: RDF Schema 1.1. W3C, https://www.w3.org/TR/2014/REC-rdf-schema-20140225/ (2004).

[13] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol – HTTP/1.1. https://www.w3.org/Protocols/rfc2616/rfc2616.html (1999).

[14] Freed, N., Klensin, J., Hansen, T.: Media type specifications and registration procedures. https://tools.ietf.org/html/rfc6838 (2013).

[15] Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language. W3C, http://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (2013).

[16] Hyland, B., Atemezing, G., Villazón-Terrazas, B.: Best Practices for Publishing Linked Data. W3C, https://www.w3.org/TR/2014/NOTE-ld-bp-20140109/ (2014).

[17] Dataset Exchange Working Group (DXWG). https://www.w3.org/2017/dxwg/wiki/Main_Page (2017).