RDF2JSON-OM: Dynamic ontology serialization using ontology mapping paths

Patrik Kompuš[1]

[1]Prague University of Economics and Business, nám. W. Churchilla 1938/4, Prague, 130 67, Czech Republic

Abstract

Processing semantically enriched data in RDF or Turtle format is not trivial for traditional development teams, who usually consume data in XML or JSON format. To improve the developers' experience of using the Semantic Web serialization formats, e.g., RDF or Turtle, an intermediate solution might be beneficial. This work presents a way for dynamic serialization of ontology metadata graphs, the *possible* structure, with ontology mapping paths, the *required* structure, to be later used by software development teams. Presented idea is implemented as a tool, RDF2JSON-OM, and is demonstrated as a web service endpoint.

Keywords

Dynamic ontology structures, Semantic Web technologies adoption, Ontology serialization

1. Introduction

As the amount of the data processed and stored by companies is rising in the last decades [1], these companies are joining the *data transformation journey* to become more aware about their own datasets and offer better products [2]. Major task in this journey is to ensure fast and precise data exchange and interoperability, for which stakeholders need to spend a lot of resources [3]. Semantic Web technologies have the means to tackle this task by design, yet the adoption of these technologies in a business environment is hard, especially when business operations and sales results depend heavily on traditional ways of delivering data to internal or external systems. It requires the whole chain of command to be on board with this change, backed by evaluation and reassurement that the solution will work out and improve current processes. Also, the developer toolbox must become comparable to the one that *normal full-stack app* developers enjoy, convincible enough to jump on it for the next project [3]. To help businesses overcome this in the long run, an intermediate solution can be introduced, thus incorporate Semantic Web technologies gradually.

JSON-LD [4] introduces a JSON tree structure with a *context* key, which holds the semantics of the information delivered. However, for heavily JSON-dependent platforms, it can be verbose and costly to handle context discovery for any possible variation, namely in an event-based data delivery. Detailed performance tests were done by JSON-LD creators in 2016 already, with results showing that JSON-LD parsing can be 7551 times slower then plain JSON [5]. Advised solution to overcome this at least partially, is to cache the *context*. Unfortunately, for many use cases that cannot implement any caching mechanism, this is a show stopper. The specification also states: *The syntax is designed to not disturb already deployed systems running on JSON, but provide a smooth upgrade path from JSON to JSON-LD.* This is not actually happening. Lanthaler et al. [6] in 2012 talked about RDF/XML being there for over decade with very little uptake, thus introducing RESTful services powered by JSON-LD. The same is happening to JSON-LD. Introduced in 2010, JSON-LD libraries for various programming languages have maximum version of 1.1 [7], Github repositories with only about 40 contributors and around 180 users in average. This is not passing even the innovators level of adoption mentioned by Pavlov et al. [3]. Vast majority of users are using JSON-LD solely for SEO purposes. Big companies will not start

EKAW 2024: EKAW 2024 Workshops, Tutorials, Posters and Demos, 24th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2024), November 26-28, 2024, Amsterdam, The Netherlands

qkomp00@vse.cz (P. Kompuš)

https://www.vse.cz/ (P. Kompuš)

0009-0001-7816-2179 (P. Kompuš)

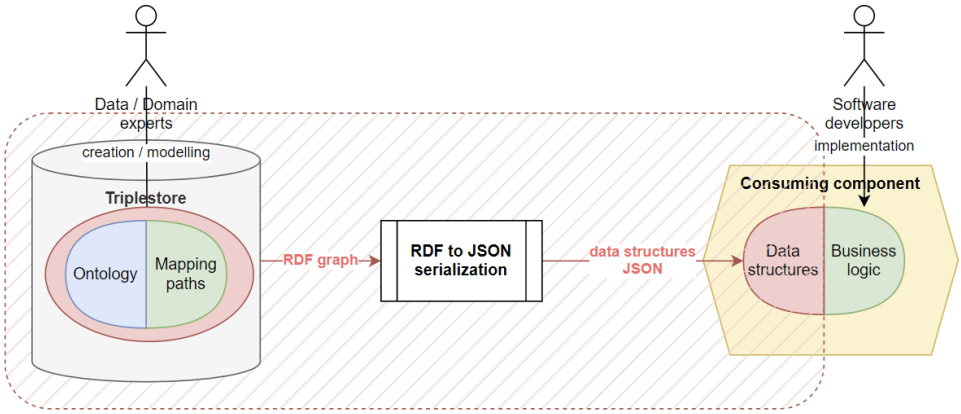

Figure 1: RDF to JSON serialization process

using Semantic Web technologies for their core activities, because the masses of software developers are not willing to adopt it [8]. Here is where it makes sense to use the presented hybrid solution: linked data, but in a JSON tree structure with a simple context, as can be seen in Figure 3 - right. For external interoperability, generating JSON-LD out of consumed JSON with context is easier than consuming it.

RDF/JSON [9] is also a root-oriented structure. The presented solution algorithm extends and improves the one described in the W3C Note. It does not require branches of the tree to be arrays; that is optional based on the modeled reality. Also, the cited algorithm does not offer any solution to cyclic structures, not even mentioning them. RDF/JSON did not become a W3C recommendation in favour of JSON-LD. Reasoning included opinions about the purpose and the final target group of such format [6], which at that time was decided to be primarily Web developers [10][11]. Unfortunately, nowadays we need to process data not only on the Web, but also in large amounts in data pipelines, messaging queues, etc., to be quickly used in other consuming components hidden to Web developers.

Ekaputra et al. proposed a modelling framework called *SOyA* [8] for modeling more interoperable structures with JSON-LD in scope. The work concentrates on modelling activities, mostly of a metadata graph, but is not going further to discuss the usage of JSON-LD structures by software developers, therefore not helping to bridge the adoption gap.

The presented solution (see Figure 1) shows a way of delivering semantically enriched data to consuming components in a more developer friendly format: *JSON* [12]. The data/domain expert is responsible for modeling the ontology and the desired structures to be serialized. The software developer then receives that structure in JSON format and uses it to implement further business logic. Although there are already several standardized JSON serialization formats [13], their main purpose it to publish data via services [12]. They deliver information in a graph-like structure, mocking the original graph. At the same time, structures representing a graph do not have one key limitation that JSON tree structure has: cycles / circular structures. This issue is solved by using newly introduced *mapping paths* over the ontology metadata graph. It also concentrates on separating the concerns of data modeling, where data structures are often not modeled by data/domain experts, but by software developers. This can speed up the adoption of mentioned SOyA framework among the consumers of data, not only the creators. This is a crucial step towards clean data delivery and interoperability.

The presented tool, which we call RDF2JSON-OM (where OM stands for ontology mapping), together with the demo service implementation can be found on Github [14].

2. RDF to JSON serialization through mapping paths

A very simple *Person* ontology can be seen in Figure 2. The section highlighted in blue represents a cycle. When attempting to serialize such an ontology graph into a tree structure, an infinite loop of

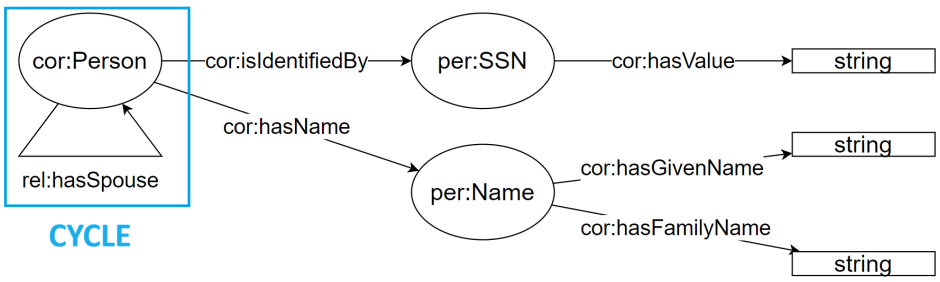

Figure 2: Person ontology with highlighted loop

operations is created, since no end of cycle is specified. Also, let's think of the use-case where the *Spouse* object should not have the *isIdentifiedBy* property, because that is not part of the requirement (e.g. lack of datasets, legislation constraints, etc.).

2.1. Traditional JSON structure

```
{
  "Person": {
    "identifier": {
      "ssn": "string"
    },
    "age": "int",
    "name": {
      "familyName": "string",
      "givenName": "string"
    },
    "spouse": {
        "name": {
          "familyName": "string",
          "givenName": "string"
        }
    }
  }
}
```

```
{
  "per:Person": {
    "per:hasName": {
      "per:Name": {
        "per:hasFamilyName": "http://www.w3.org/2001/XMLSchema#string",
        "per:hasGivenName": "http://www.w3.org/2001/XMLSchema#string"
      }
    },
    "per:hasSpouse": {
      "per:Person": {
        "per:hasName": {
          "per:Name": {
            "per:hasFamilyName": "http://www.w3.org/2001/XMLSchema#string",
            "per:hasGivenName": "http://www.w3.org/2001/XMLSchema#string"
          }
        }
      }
    },
    "per:isIdentifiedBy": {
      "per:SSN": {
        "per:hasIdentifierValue": "http://www.w3.org/2001/XMLSchema#string"
      }
    }
  },
  "@context": {
    "per": "http://www.example.org/person/"
  }
}
```

Figure 3: Left: Example of traditional JSON tree structure. Right: Generated JSON structure of a Person

A JSON tree is a 2-dimensional key-value structure that is easy to consume, parse, and get the content from. The keys have to be unique on the same level of the tree, and values can be either a simple literal, an object or an array of objects that can expand further internally. An example of a possible JSON tree structure can be seen in Figure 3 - left.

2.2. Structure generated by RDF2JSON-OM

In the presented solution, the generated structure is a tree with a root, the serialized object; see Figure 3 - right. This serialization process is based on the ontology it is meant to serialize. There is no hard-coded pattern, therefore the whole process is dynamic towards the modeled reality, preferably created by data/domain experts. Software developers then get the JSON structure they can work with. It provides them with the keys, that are IRIs, pointing to the ontology for further metadata discovery, if needed. The populated datatype properties (e.g. *hasFamilyName*), can carry any relevant information, e.g. the datatype of the value, which the developers should use.

Ontologies can be provided as a parameter, but can also be stored in a triple store. That is a big benefit, since multiple ontologies complementing each other, e.g., *Ontology network*, can be stored in the same graph, and the structure generator will create a JSON consisting of classes and properties from all ontologies touching any of the elements in the desired tree of the serialized object. The storage can be either persistent, or virtual, built during the generation.

2.3. Implementation

The solution implements the RDF2JSON-OM structure generator using the JENA API library, which loads the ontology model as RDF triples and populates a generic Map<Object,Object> object based on the selected root. Afterwards, it creates the corresponding branches based on the modeled *datatype* and *object properties*. *Datatype property* ends with a leaf to store only the final value, whereas the above creation is repeated in recursion for *object properties*, until the final ontology *class* has no more object properties. It takes *subClasses*, *subProperties* and restrictions into consideration as well. The required part of the generated structure is the context, which carries all prefixes used in the structure. The implemented demo service is a REST endpoint accepting an ontology in RDF/XML format and root object IRI, returning the generated structure [14].

2.4. Mapping paths

One of the biggest challenges to overcome was the unavailability to convert circular structures to JSON, although an IETF draft exists [15]. When there is a cycle in the ontology, e.g., objects are pointing to each other, the JSON generator would enter an endless loop. There are several non-semantic techniques to overcome this, e.g., content replacements or substitutions, but that leads to the need for content restoration and adds more complexity.

Instead, the presented solution involves more descriptive techniques and also benefits the content creators, as they now have the power to specify what the final structure is supposed to look like. The creator of the model uses restricted paths to *draw* only the relevant branches of the final tree structure. For the infinite loop to happen, the creator would need to create an infinite path, which is impossible in the real world and also impossible to *draw*. Generic JSON-LD serialization would include the property *isIdentifiedBy* for *Spouse* object, since it is part of the ontology. With mapping paths we generate only what we *draw* to be generated, thus fulfilling the requirements set by stakeholders, yet still keeping the model as designed by data experts. Moreover, the modeled paths can be re-used in various structures. They are interoperable, as long as they are logically correct.

An example of mapping paths is highlighted in Figure 4.

3. Conclusions and future work

This work describes the idea and implementation of the ontology graph serialization into a JSON tree structure, including overcoming the problem with circular structures. It allows the content creators to explicitly restrict parts of the graph from being serialized and gives them the power to specify how deep the circular nesting should go. The structure is generated automatically and dynamically, without any further configuration. These structures are to be sent out as serialized data structures to consumers, who can either read them with semantics in mind and populate with content for further event based message processing or decompose them and store them in any arbitrary structure, which raises the possibility of adopting Semantic Web technologies in a broader community. The final generated structure also conforms to most of the best practices listed by JSON-LD creators [12] at the moment of writing this paper, therefore foreseeing the pitfalls of data consumption and thus confirming the maturity of the solution.

We plan to include the restriction validation functionality in our tool. *OWL axioms* themselves can assure some level of validation, e.g. *Person* can be declared as disjoint from *Vehicle*, which then makes a reasoner trigger an inconsistency even if the data experts makes a mistake and tries to model the

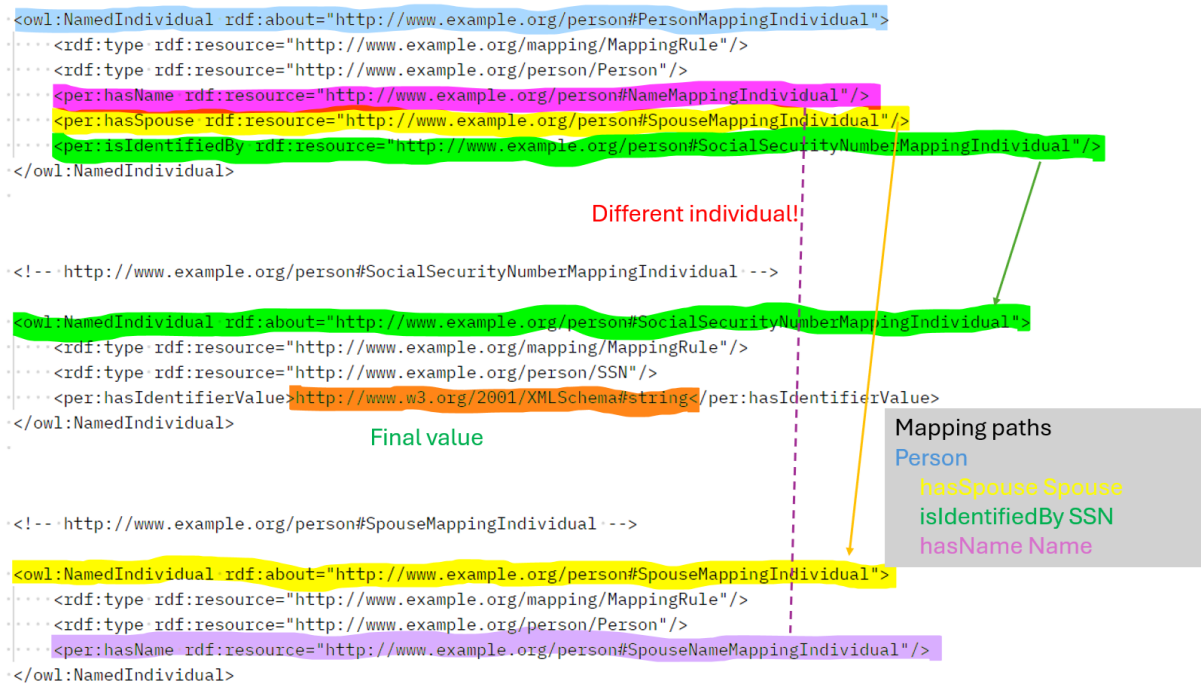

Figure 4: Person ontology with mapping paths

mapping path that way. For much extensive validation, this can be then complemented with *SHACL* [16] shapes modelled by data experts.

Another benefit and usage of the proposed solution is that the same algorithm can be used to generate a *JSON schema* in parallel. This schema can be later used for the structures validation during the future data transfer process, where they will be validated before they reach the software developers, effectively lowering the amount of tests needed to be prepared.

To improve developers' experience with consuming and understanding the delivered structures, we plan to include the generation of *Swagger documentation* out of RDF, solely based on data experts' models. This is another step towards Semantic Web technologies adoption.

Acknowledgments

This work has been supported by the EU's Horizon Europe grant no. 101058682 (Onto-DESIDE).

References

[1] F. Provost, T. Fawcett, Data Science and its Relationship to Big Data and Data-Driven Decision Makingt, 2013. URL: https://doi.org/10.1089/big.2013.1508.

[2] Natuvion, Transformation 2022, The Study, 2023. URL: https://www.natuvion.com/newsroom/challenges-2023.

[3] G. Pavlov, P. Genevski, Semantic technologies for the masses, in: Proceedings of 2015 Big Data, Knowledge and Control Systems Engineering, 2015, pp. 33–43.

[4] G. Kellog, D. Longley, P. Champin, JSON-LD 1.1, 2020. URL: https://www.w3.org/TR/json-ld11/.

[5] M. Sporny, D. Longley, JSON-LD Best Practice: Context Caching, 2016. URL: https://web.archive.org/web/20230131235929/https://manu.sporny.org/2016/json-ld-context-caching/.

[6] M. Lanthaler, C. Gütl, On Using JSON-LD to Create Evolvable RESTful Services, in: Proceedings of the 3rd International Workshop on RESTful Design (WS-REST 2012) at WWW2012, 2012.

[7] JSON-LD.org, JSON-LD.org, 2020. URL: https://json-ld.org/.
[8] F. E. Ekaputra, C. Fabianek, G. Unterholzer, E. Gringinger, The Semantic Overlay Architecture for Data Interoperability and Exchange, in: Proceedings of the 2023 IEEE International Conference on Data and Software Engineering (ICoDSE), 2023, pp. 232–237.
[9] I. Davis, T. Steier, A. J. H. Hors, RDF 1.1 JSON Alternate Serialization (RDF/JSON), 2012. URL: https://www.w3.org/TR/rdf-json/.
[10] T. Steiner, JSON Emergency break, 2011. URL: https://blog.tomayac.com/2011/08/23/json-emergency-brake-184758/.
[11] M. Sporny, Linked JSON: RDF for the Masses, 2011. URL: https://web.archive.org/web/20210224110211/http://manu.sporny.org/2011/linked-json/.
[12] G. Kellog, JSON-LD Best Practices, 2024. URL: https://w3c.github.io/json-ld-bp/.
[13] W3C, JSON Serializaton Examples, 2011. URL: https://www.w3.org/2011/rdf-wg/wiki/JSON-Serialization-Examples.
[14] P. Kompuš, RDF2JSON-OM at Github, 2024. URL: https://github.com/TimotejFox/rdf2json.
[15] P. Bryan, L. Zyp, JSON Reference, 2012. URL: https://datatracker.ietf.org/doc/html/draft-pbryan-zyp-json-ref-0.
[16] H. Knublauch, D. Kontokostas, Shapes Constraint Language (SHACL), 2017. URL: https://www.w3.org/TR/shacl/.