J2RM: an ontology-based JSON-to-RDF Mapping tool

Abstract. This paper introduces *J2RM*: a tool to process mappings from JSON data to RDF triples guided by an OWL2 ontology structure. The mappings are defined as annotation properties associated with each ontology entity of interest. They are embedded in an ontology file so that they can be readily deployed to automate RDF-graph creation. Section 1 presents the motivation, main contributions, and related work. Section 2 describes some mapping definitions based on an example; it presents their formal grammar and explains how these mappings work. Last, section 3 presents the conclusions and ongoing work.

1 Positioning

Quite often data transformation tasks consume a lot of engineering effort when dealing with heterogeneous data models and formats. Specifically, creating an RDF-graph based on data extracted from a closed and proprietary information system can be a daunting task. A simple approach to extract the required and curated data from these systems is to expose the data in an "easy-to-process" format, usually, JSON, as an intermediary representation. JSON has been used extensively in a variety of processing tasks as a serialization format becoming the *lingua franca* for data interchange on the Web[2]. Frequently, software engineering teams do not have a deep understanding of Semantic Web technologies. In such cases, a tool that could abstract all the time-consuming complexities of creating and storing RDF triples –on-the-fly– from any JSON data set, would be an ideal solution to have. Moreover, it would be great to solely focus on designing the underlying JSON mappings following the ontology structure. This paper introduces *J2RM*, a tool that gives a versatile solution for these use cases[1]. Its main goal is to automate RDF-graph creations from JSON data following an OWL2 ontology structure. The mappings are defined as annotation properties associated with each ontology entity of interest such as classes and object/datatype-/annotation properties. The mappings are embedded in an ontology file so they can be readily deployed to automate the graph creation from a "standardized" JSON structure, tailored from any information systems' data (see Figure 1). With *J2RM*, one could work with different JSON structures: each type of mappings embedded in a specific ontology file. Some transformation and mapping languages have been proposed to generate RDF from non-RDF data, including SPARQL-Generate [9], XSPARQL [4], R2RML [7], RML [3], SAURON [6], Elda [8], and [5]. While most of these methods consider a given mapping, in this paper we consider the use of an OWL2 ontology for extracting the schema of the target RDF data. To the best of our knowledge, while there are many tools that follow different approaches to map JSON data to RDF, none of them embed the mappings in ontology files.

[1] A series of demo videos of the tool can be found at `https://bit.ly/3h5iE5M`

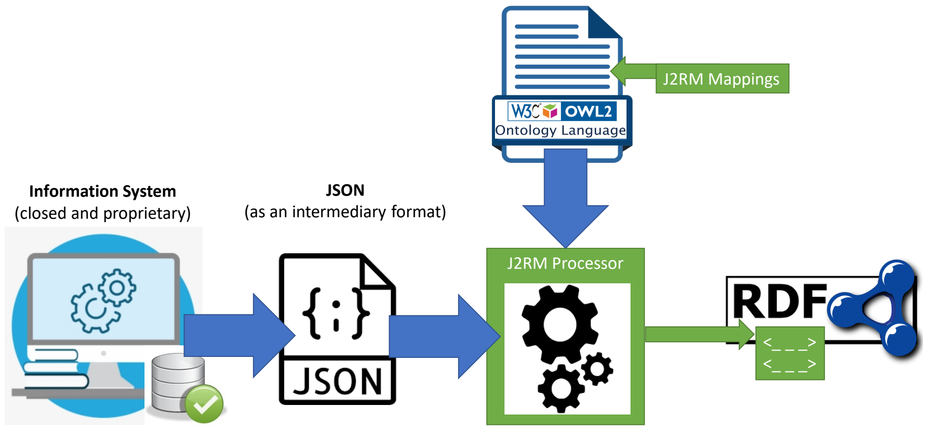

Fig. 1: *J2RM* general functional architecture

```
"doc": {
    "id": "A19453",
    "year": "2011",
    "scheme": "ABC-Grant",
    "app_AdminInstName": "Bright Institute of Neuroscience",
    "finalScore": "5.1125",
    "title": "Correlation between strokes and dementia:
    Cognition changes following stroke",
    "Keywords": "stroke | neuropsychology | cognitive disorders",
    "Broad_Research_Area": "Clinical Medicine",
    "FoR_code": "12908", "FoR": "Central Nervous System",
    "scoreCriteria1": "4.91",
    "publicData": [
    {   "title": "Cognition following stroke: is it neurodegenerative?",
        "id": "19453",
        "researchers": [ "Dr Peter Parker", "Dr Susan Storm" ],
        "startDate": 2012, "endDate": 2016,
        "funder": [ "Medical Research Council" ],
        "adminInst": null,
        "principalAnalyst": [ "Dr Susan Storm" ],
        "links": [ { "href": "http://test.com/key/19453" } ]
    } ],
    "team": [
      { "ind_id": "401636", "role": "CIA",
        "name": "Dr Susan Storm",
        "org": "Bright Institute of Neuroscience",
        "ORCID": "",
        "FoR_Data": [
          { "FoR_code": "12908", "FoR": "Central Nervous System",
            "FOR_Date_Start": "2001-02-01"
          },
          { "FoR_code": "72099", "FoR": "General Cognitive Science",
            "FOR_Date_Start": "2001-02-01"
          } ]
      },
      { "ind_id": "401443", "role": "CIB",
        "name": "Dr Peter Parker",
        "org": "Bright Institute of Neuroscience",
        "ORCID": "https://orcid.org/0X30-01X1-68X0-083X",
        "FoR_Data": [
          { "FoR_code": "60939", "FoR": "Psychology and Cognitive Sciences",
            "FOR_Date_Start": "2006-03-01"
          },
          { "FoR_code": "10791", "FoR": "General Neurosciences",
            "FOR_Date_Start": "2006-03-01"
          } ]
      } ]
}
```

Fig. 2: Excerpt of a JSON document about research medical grants

2 Implementation: Mapping Definitions

Figure 2 presents an excerpt of a modified JSON document about medical research grants, while table 1 presents an excerpt of a grants ontology definition along with the *J2RM* mappings. *J2RM* mappings follow the path-based syntax presented in figure 3. The mappings are designed as JSON-Pointer[1] extensions with their own primitives that define basic transformations and operations applied to the JSON data. Below, we briefly describe how these mappings work.

Class mappings: create an instance for each mapped value with the structure `<data#ID> a <class>`. `#1` generates the triple `d:Eval#A19453 a m:Eval`. `#2` maps to an array of values (`["","https://orcid.org/0X30-01X1-68X0-083X"]`)[2]. In this case, the meta-character "`#`" states that the mapped value is used "as is" as an IRI[3]. The meta-character "`!`" in `#3` states that the value is used to generate an IRI: `d:Area#Clinical-Medicine a m:Area`. `#4` maps to an array formed of composite values based on the tree structure: `["A19453-401636", "A19453-401443"]`, which are used to generate two instances of `m:ChiefAnalyst`.

[2] Empty mapped values ("", `null`, –or non-existent–) are not processed.
[3] The created triple is `<https://orcid.org/0X30-01X1-68X0-083X> a m:ORCID`

```
            ⟨path⟩ ⊨ ⟨JSON-pointer⟩⟨OWL-res⟩? | ⟨map⟩
    ⟨JSON-pointer⟩ ⊨ / ⟨step⟩⟨JSON-pointer⟩ | / ⟨step⟩
             ⟨map⟩ ⊨ ⟨pointer⟩ | ⟨multi-pointer⟩ | ⟨OP-map⟩
         ⟨pointer⟩ ⊨ ⟨simple-p⟩ ⟨OWL-res⟩? | ⟨complex-p⟩
   ⟨multi-pointer⟩ ⊨ ⟨simple-p⟩ (\n+⟨simple-p⟩) +
          ⟨OP-map⟩ ⊨ D=⟨simple-p⟩? \nR=⟨simple-p⟩
        ⟨simple-p⟩ ⊨ ⟨step⟩/⟨simple-p⟩ | ⟨step⟩
       ⟨complex-p⟩ ⊨ ⟨simple-p⟩⟨operation⟩
       ⟨operation⟩ ⊨ ⟨single-path-op⟩ ⟨OWL-res⟩?
       ⟨operation⟩ ⊨ ⟨multiple-path-op⟩

      ⟨single-path-op⟩ ⊨ ⟨meta-char⟩ | ⟨split⟩ | ⟨rel-value⟩
    ⟨multiple-path-op⟩ ⊨ ⟨condition⟩ | ⟨equal-values⟩
           ⟨condition⟩ ⊨ %⟨and-expression⟩
        ⟨equal-values⟩ ⊨ |=|⟨simple-p⟩
               ⟨split⟩ ⊨ ("string-delimiter")
           ⟨rel-value⟩ ⊨ ⟨relational-operator⟩⟨value⟩
             ⟨OWL-res⟩ ⊨ @entity-name⟨range⟩?
               ⟨range⟩ ⊨ -⟩entity-name
                ⟨step⟩ ⊨ name | number
 ⟨relational-operator⟩ ⊨ = | ⟩ | ⟨ | ⟩= | ⟨=
           ⟨meta-char⟩ ⊨ # | ! | ~ | ⟨
```

Fig. 3: Excerpt of the *J2RM* mappings formal grammar expressed in simple Extended Backus-Naur Form (EBNF) notation

Table 1: Excerpt of a grant ontology definition along with the *J2RM* mappings

#	OWL2 Entity	QName	J2RM Mappings (defined as annotation properties)
1	Class	`m:Eval`	`/doc/id`
2	Class	`m:ORCID`	`doc/team/ORCID#`
3	Class	`m:Area`	`doc/Broad_Research_Area!`
4	Class	`m:ChiefAnalyst`	`doc\n+/id\n+/team/ind_id`
5	Datatype Prop.	`m:fullName (xsd:string)`	`doc/team/name~`
6	Datatype Prop.	`m:keyword (xsd:string)`	`doc/Keywords("|")`
7	Datatype Prop.	`m:CGcriterion (xsd:float)`	`/doc/scoreCriteria1%/doc/scheme="ABC-Grant"&&/doc/year<2019`
8	Datatype Prop.	`m:link (xsd:anyURI)`	`doc/publicData/links/href<`
9	Object Prop.	`m:hasFoR rdfs:range(m:FoR)`	`/doc@GrantApp` `doc/team/FoR_Data@FoR_cat`
10	Object Prop.	`m:hasORCID rdfs:range (m:ORCID)`	`doc/team`
11	Object Prop.	`m:hasResearcher-1 (rdfs:subPropertyOf)`	`D=\nR=doc/team/role="CIA"`

12	Object Prop.	`m:about`
		`rdfs:range(m:FoR,m:Organization)`
			`doc/team/FoR_Data@CV->FoR`
			`doc/team/org|=|doc/publicData/adminInst`

13	Annotation Prop.	`dc:title`	`doc/title~@GrantApp` `doc/publicData/title~@Grant`
"*m*" (model) and "*d*" (data) are namespace prefixes defined in the ontology ("*m*") and in the config. file ("*d*")			

Datatype (dp) and annotation (ap) prop. mappings: create a triple for each mapped value with the structure `<data#ID> <dp|ap> "value"^^<xsd:type>`. For each class (and sub-classes) that has `<dp|ap>` as a class restriction, *J2RM* will create a triple for each mapped instance. One example of `#5` is `d:Analyst#401636 m:fullName "Dr Susan Storm"^^xsd:string` considering that `m:Analyst` has `m:fullName` in its class restrictions. In this case, the meta-character "`~`" states that the mapped value is used to automatically create an `rdfs:label` triple as well (`#13` presents similar examples). `#6` creates a triple for each value found when splitting the mapped values using the delimiter " `|` " and, thus, it will generate three keywords. `#7` defines a "conditional path": it will create a triple with the mapped value of "`4.91`" because the restriction (after meta-character "`%`") evaluates to `true`: `scheme` and `year` values are mapped and evaluated correctly. In `#8`, the meta-character "`<`" defines a mapping to a common `JSONObject` ancestor: for the `m:Grant` class with instances mapped as `doc/publicData/id`, the ancestor is `publicData`[4].

Object prop. mappings: create triples between sets of mapped values for each identified class that is applicable in the analyzed context (class hierarchies, sub-properties, etc). The structure generated is `<domainData#ID> <op> <rangeData#ID>`, where `<domainData#ID>` correspond to the mapped instances

[4] The created triple is `d:Grant#19453 m:link "http://test.com/key/19453"^^xsd:anyURI`

of each `<op>` domain class, and `<rangeData#ID>` correspond to the mapped instances of each `<op>` range class. The mappings are paths that define the connection between `<domainData#ID>` and `<rangeData#ID>`. Simple cases, such as `#9`[5] and `#10`, find the connection between the instances in a single path: in `#9`, `/doc` connects the domain instances `/doc/id="A19453"` with the range instances `/doc/FoR_code="12908"`, creating the triple `d:GrantApp#A19453 m:hasFoR d:FoR#12908`. The meta-character "`@`" is used (`#9`, `#12`, `#13`) to indicate the entity (domain class) attached to the path (useful for entity disambiguation). In `#10`, when applying to the domain class `m:Analyst`, the mapping results in an array of values for both, the domain (`["401636","401443"]`) and the range (same as `#2`). The tool keeps track of the context for each mapped `JSONObject` that could result in a valid connection. `#11` defines a mapping based on two different paths: for domain (`D=`, states the usage of the already known instances from the domain classes) and range (`R=...`, states the mapping to the values that are equal to `"CIA"`). `#12` defines two mappings: one that explicitly disambiguate the domain and range classes (`CV->FoR`), and other, `<p1>|=|<p2>`, where it will map to values of `<p1>` only if those are equal to values of `<p2>`.

Along with each mapping, one can specify the target endpoint and graph. **Target endpoint** is a label that identifies a SPARQL endpoint access[6] where the triples will be created. Examples: `test, prod`. **Target graph** is the named graph where the triples will be created. It is defined as a namespace prefix in the ontology file. Examples: `g0-testing, g0-prod`. The IRI of the namespace prefix will be used as the named graph for the triple creation for that mapping.

3 Conclusions and Ongoing Work

J2RM provides a simple mechanism to define the necessary mapping rules for an automated RDF-graph creation task guided by an OWL2 ontology structure from any JSON data. The key aspect is that the mappings are embedded in an ontology file (this does not imply that the JSON structure is intrinsically tied to the OWL2 model): for different JSON structures, one could define each type of mappings in different ontology files. *J2RM* is in its early development stages. It has been tested on three different domain ontologies. We will increase the support of more complex JSON mappings and more OWL2 axioms. The major contributions are: the ability to selectively extract data and perform some operations on the source JSON, the "portability" of the mappings embedded in the OWL2 ontology file as annotation properties, and its ease of use while hiding the complexity of creating RDF triples following OWL2 axioms.

References

1. JavaScript Object Notation (JSON) Pointer. Request for comments, Internet Engineering Task Force (IETF) (April 2013), `https://tools.ietf.org/html/rfc6901`

[5] `#9` defines two mappings that apply to distinct domain classes.
[6] Defined in the *J2RM* configuration file.

2. ECMA-404: The JSON Data Interchange Syntax. Standard, ECMA International (December 2017), `https://www.json.org/`
3. RDF Mapping Language (RML). Unofficial draft, Ghent University (July 2020), `https://rml.io/specs/rml/`
4. Akhtar, W., Kopecký, J., Krennwallner, T., Polleres, A.: Xsparql: Traveling between the xml and rdf worlds – and avoiding the xslt pilgrimage. In: The Semantic Web: Research and Applications. pp. 432–447. Springer Berlin Heidelberg (2008)
5. Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J.: A direct mapping of relational data to rdf (2012)
6. Bareau, C., Blache, F., Bolle, S., Ecrepont, C., Folz, P., Hernandez, N., Monteil, T., Privat, G., Ramparany, F.: Semi-automatic rdfization using automatically generated mappings. In: ESWC Posters and Demos Track (2020)
7. Das, S., Sundara, S., Cyganiak, R.: R2rml: Rdb to rdf mapping language (2012)
8. Elda, a Linked Data API implementation. `https://github.com/epimorphics/elda`
9. Lefrançois, M., Zimmermann, A., Bakerally, N.: Sparql-generate: RDF generation from heterogeneous data sources. In: EKAW Satellite Events (2016)