DATABASE
The Journal of Biological Databases and Curation

Database, 2019, 1–9
doi: 10.1093/database/baz088
Original article

Original article

An effective biomedical data migration tool from resource description framework to JSON

Jian Liu[1,*], Mo Yang[1], Lei Zhang[2] and Weijun Zhou[3,*]

[1]School of Computer Science and Technology, Harbin Institute of Technology, Harbin, 150001, China, [2]Zhejiang University of Science and Technology, 310023, Hangzhou, China and [3]Department of Hematology, Zhujiang Hospital, Southern Medical University, Guangzhou, 510282, China

*Corresponding author. Tel: +86-18686896700; Email: jianliu@hit.edu.cn

Citation details: Liu,J., Yang,M., Zhang,L. *et al.* An effective biomedical data migration tool from resource description framework to JSON. *Database* (2019) Vol. 2019: article ID baz088; doi:10.1093/database/baz088

Received 21 March 2019; Revised 6 June 2019; Accepted 9 June 2019

Abstract

Resource Description Framework (RDF) is widely used for representing biomedical data in practical applications. With the increases of RDF-based applications, there is an emerging requirement of novel architectures to provide effective supports for the future RDF data explosion. Inspired by the success of the new designs in National Center for Biotechnology Information dbSNP (The Single Nucleotide Polymorphism Database) for managing the increasing data volumes using JSON (JavaScript Object Notation), in this paper we present an effective mapping tool that allows data migrations from RDF to JSON for supporting future massive data explosions and releases. We firstly introduce a set of mapping rules, which transform an RDF format into the JSON format, and then present the corresponding transformation algorithm. On this basis, we develop an effective and user-friendly tool called RDF2JSON, which enables automating the process of RDF data extractions and the corresponding JSON data generations.

Database URL: https://github.com/lyotvincent/rdf2json

Introduction

With the increasing adoption of semantic web technologies (22–25) and formalisms in biomedical and biomolecular areas, many popular database applications (such as Uniprot (36), Ensembl (9), BioModels (19), etc.) provide accessible data represented in a Resource Description Framework (RDF) format (10, 13, 27). As the World Wide Web Consortium (W3C) recommended standard, the graph-based RDF model is well suitable for explicitly publishing life science data and linking the diverse data resources (5, 7, 11, 28). For instance, the RDF model is chosen in GlycoRDF (32) for the glycomics-based data resource integration and representation. In (31), by using the RDF model, the DisGeNET platform interconnects multiple gene-disease associations and pharmacological data sources obtained from several drug discovery applications for helping us study molecular mechanisms underpinning human diseases. To effectively publish the cross-reference information about diseases and abnormal states extracted from disease ontology and abnormality ontology, Disease Compass (17) linked the causal chains of diseases by using the RDF model.

The emergence of numerous RDF-based applications lead to the generation of massive RDF data resources, which naturally attracts the interest of seeking for novel architectures and providing supports for the future RDF data explosion (2). In recent years, JSON (JavaScript Object Notation) becomes a popular format for representing and publishing massive data resources over the web application (8, 16, 18, 35, 38). JSON documents could be used to store records in MongoDB database (3), which is a cross-platform and support distributed processing of data sets with a large size. Previous attempts (4, 14) partitions RDF data graph into several subgraphs by duplicating no-literal nodes, and then organizes these partitioning subgraphs in MongoDB. This partitioning approach duplicates no-literal nodes twice and costs some extra storage space.

JSON has been accepted as a major format for the future data explosion in National Center for Biotechnology Information dbSNP (34). Now the architecture of dbSNP is redesigned to provide products by using JSON files, which suit for the programmatic approaches well and could effectively provide supports for the increasing volume of data. Orphanet (26) chooses JSON as a new data format for the mission providing the scientific community with freely available data sets related to rare diseases and orphan drugs in a reusable format. As a completely language-independent format, JSON has a higher compression ratio in coding, and this property makes it take up less space (30). This property also makes JSON become a popular format for data interchange on the web (15, 20, 21, 39). Therefore, many web-based services, such as Semantic-JSON (16), Open chemistry (12), etc., choose JSON as the data representation formats. In order to provide access to the linked life science database, Semantic-JSON (16) develops an interface using a representational state transfer (REST) web service to the retrieval of the Semantic Web data in JSON formats. Open chemistry (12) develops a web application by providing access to the open source chemical science data in JSON formats. It facilitates data exchanges across different languages, and makes it easy to send the chemical science data to the web client developed in JavaScript. Moreover, the Ensembl (39) provides a web service to retrieves its data through the REST service, and JSON is chosen as its main endpoint data exchange format.

The adoption of a new data format will trigger the requirement of data migrations from the historical format to the new one (33). There have been some mapping tools that allow data migrations in previous works, e.g. Bio2RDF (6) and VCF2RDF (29), etc. Bio2RDF creates a knowledge space based on RDF documents by converting public bioinformatics databases into RDF documents and linking them together with normalized URIs (Uniform Resource Identifiers) in a standardized way. VCF2RDF presents the isomorphic mapping between VCF and RDF to make portability and interoperability of the self-contained description of the genomic variation in next-generation sequencing results. Unfortunately, although JSON has been employed to model future data explosion, and the available data size of RDF sources is rapidly increasing, relatively little work focuses on the data migrations from RDF to JSON (1, 37). In particular, in the new era of big data, the studies on the mapping rules and an effective mapping tool from RDF to JSON for biomedical users without programming skills obviously lag behind.

Currently, developing an effective mapping technique deal with data migrations from the RDF model to the JSON model in a uniform way is still an open problem. In order to solve this problem, in this paper we present an effective mapping tool that allows data migrations from RDF to JSON for supporting future massive data explosion. After giving a set of mapping rules which transform the RDF format into the JSON format, the corresponding transformation algorithm from RDF to JSON is presented. On this basis, we develop a user-friendly tool called RDF2JSON, which enables automating the process of the RDF data extraction and the corresponding JSON data generation. Finally, experimental evaluations are carried out to verify the advantages of the proposed tool by using real-world data sets.

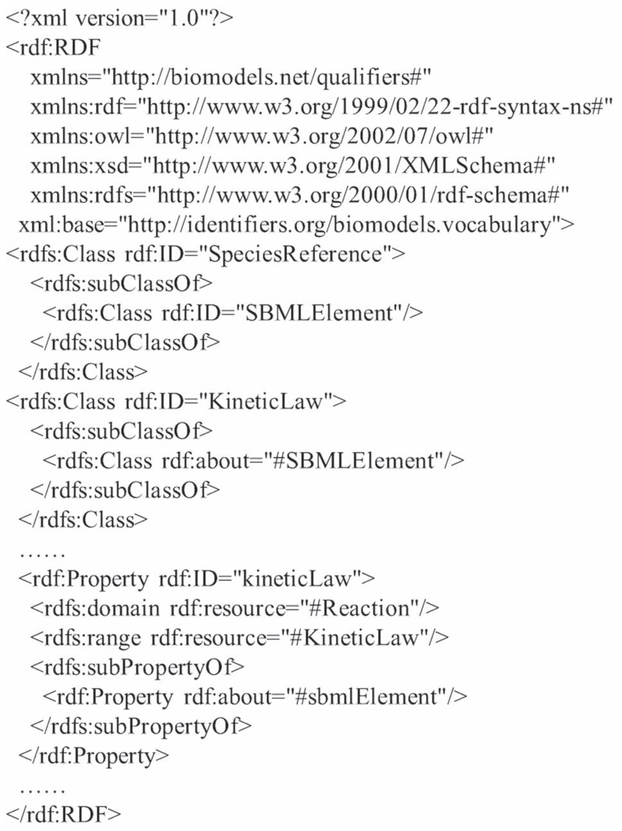
Materials and methods

In this section, we firstly give the RDF graph model parsing process, and then introduce the mapping rules. Based on these mapping rules to transform, the corresponding data migration algorithm is developed. The implementation details about the mapping tool RDF2JSON is given in the end of this section.

RDF graph parsing

RDF is based on the triples consisting of resources, properties and values. A resource is an entity accessible by an URI, a property defines a binary relation between resources or literals and a value is a literal or a resource. RDF Schema (RDFS) (27) provides a data modeling vocabulary and syntax to RDF descriptions. RDFS allows the definitions of the class and the property, respectively, which have global effects. It is flexible to add new properties to existing classes. In the following, a fragment of an RDF schema about computational models of biological processes is given in Figure 1.

In Figure 1, there are some classes such as ‘SBMLElement’, ‘SpeciesReference’ and ‘KineticLaw’, and some properties such as ‘sbmlElement’ and ‘kineticLaw’. Since



```
<?xml version="1.0"?>
<rdf:RDF
  xmlns="http://biomodels.net/qualifiers#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xml:base="http://identifiers.org/biomodels.vocabulary">
<rdfs:Class rdf:ID="SpeciesReference">
  <rdfs:subClassOf>
   <rdfs:Class rdf:ID="SBMLElement"/>
  </rdfs:subClassOf>
 </rdfs:Class>
<rdfs:Class rdf:ID="KineticLaw">
  <rdfs:subClassOf>
   <rdfs:Class rdf:about="#SBMLElement"/>
  </rdfs:subClassOf>
 </rdfs:Class>
 ……
 <rdf:Property rdf:ID="kineticLaw">
  <rdfs:domain rdf:resource="#Reaction"/>
  <rdfs:range rdf:resource="#KineticLaw"/>
  <rdfs:subPropertyOf>
   <rdf:Property rdf:about="#sbmlElement"/>
  </rdfs:subPropertyOf>
 </rdf:Property>
 ……
</rdf:RDF>
```

Figure 1. A fragment of an RDF schema.

classes and properties can be refined in subclasses and subproperties, there is a hierarchy of classes and a hierarchy of properties existing in RDFS. For example, the tag ‘rdfs:subClassOf’ represents that the class ‘KineticLaw’ is a subclass of the class ‘SBMLElement’, and the tag ‘rdfs:subPropertyOf’ depicts that the property ‘kineticLaw’ is a subproperty of the property ‘sbmlElement’. In the domain and range mechanisms of RDFS, a property is defined according to a domain and has an associated range that can be a literal or a class, which build the relations among classes and properties. For example, the property ‘kineticLaw’ is limited by the property ‘domain’ and the property ‘range’, which means that the types of values are instances of the class ‘KineticLaw’ and the class to which the property ascribes to is ‘Reaction’.

An RDF statement consists of a subject, a predicate and an object, in which the subject is a class and the object is a class or a literal, the predicate is a property. Since a hierarchy of classes and a hierarchy of properties are built by RDFS using the vocabulary in the RDF statements, we can take advantage of this hierarchy to construct an RDF graph model for data extractions. Figure 2 shows the constructed graph model of the RDF schema above based on the hierarchy relationships of classes and properties. In particular, in this graph model, vertices depict the classes and literals, and edges represent properties.

Mapping rules from RDF to JSON

This section will introduce the mapping rules from RDF to JSON. The details of the mapping rules are shown as follows.

RULE 1 For the root element (depicted as rdf: RDF) in the RDF document, the namespaces are transformed in the following format:

“namespaces”: {prefix: namespace, . . .},

where the prefix is to refer to the RDF namespace, and the namespace is identified by the URI.

RULE 2 For the basic RDF description, the key is the property. The value of the property is divided into two categories to describe:

(a) Literal type: The value object includes value and datatype. If the language property of the property value is specified, the language attribute (*lang*) is also included.

(b) URIRef type: The value object includes ‘rdf:resource’, which is called by all things described by RDF.

RULE 3 For the value of the property being a blank node identifier, all the triples, in which the identifier is the resource, are extracted from the model. And these triples are processed recursively according to the basic RDF description.

In the case where the value of a property is a set of values, the W3C recommendation defines container such as bags, sequences and alternatives in order to hold such values. The parsing rules are following:

RULE 4 For a set of values, the following elements under the child relation are mapped into key-value pairs, in which the key is the ordinal attribute and the value is the value of the element. The type of the container is retained.

For describing a group containing only the specified members and complying with the grammar rules of the triple, the elements are nested according to the order of elements.

RULE 5 For a group of the specified elements, the nested structure is parsed to generate a list of key-value pairs and the type information is retained.

RULE 6 For the hierarchy relation of classes, the key-value pair, in which the key is the property name of subclass and the value is the URI of superclass, is put under the resource JSON object.

Figure 3 shows a schematic diagram for the process of hierarchy relation of the class, in which C1 and C2 are class instances. The solid line with label represents that C1 is a

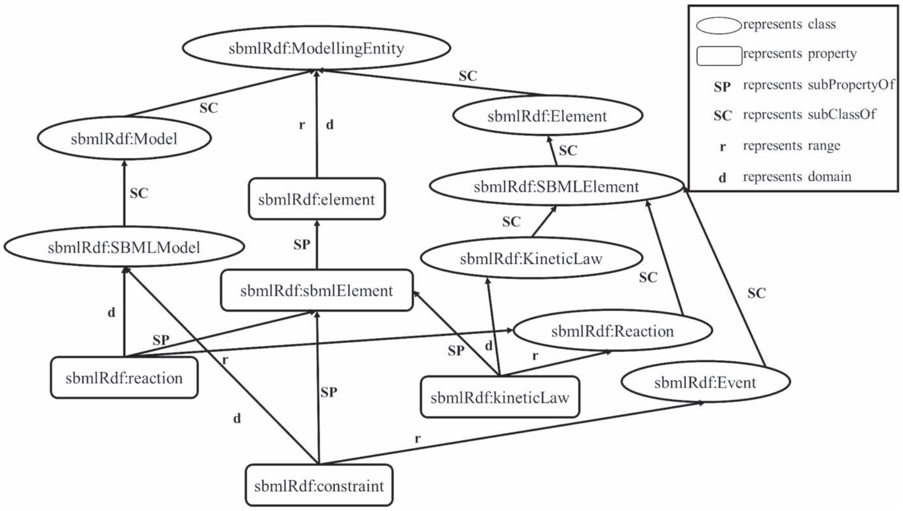
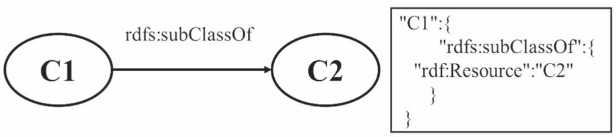
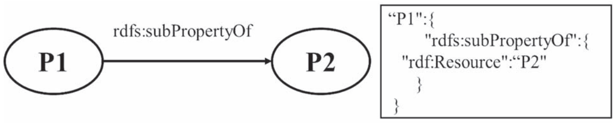


Figure 2. The partial graph model based on the given instance of RDFS.



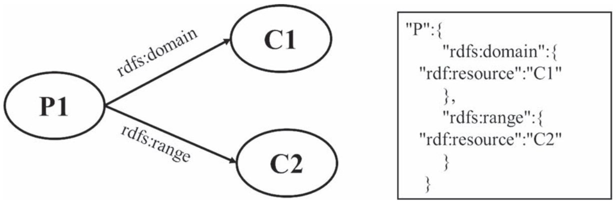
```
"C1":{
      "rdfs:subClassOf":{
  "rdf:Resource":"C2"
      }
}
```

Figure 3. A mapping example of the hierarchy relation of classes.



```
“P1":{
      "rdfs:subPropertyOf":{
  "rdf:Resource":“P2"
      }
}
```

Figure 4. A mapping example of the hierarchy relation of properties.



```
"P":{
      "rdfs:domain":{
  "rdf:resource":"C1"
      },
      "rdfs:range":{
  "rdf:resource":"C2"
      }
}
```

Figure 5. A mapping example of the domain and the range of properties.

subclass of C2. The relation is extracted from the triple set of the description about C1. The relation is transformed into a pair of key and value, which is put under the JSON object of C1.

RULE 7 For the hierarchy relation of properties, the key-value pair, in which the key is the property name of the subproperty and the value is the URI of super property, is put under the resource JSON object.

Figure 4 shows a schematic diagram for the process of hierarchy relation of property. The solid line with label represents that P1 is a subproperty of P2, and P1 and P2 are property instances. The relation is extracted from the triple set of the description about P1. The relation is transformed into a pair of key and value, which is put under the JSON object of P1.

RULE 8 For the domain and range of a property, two key-value pairs, in which the keys are the names of properties and the values are the value of the property, are put under the property JSON object.

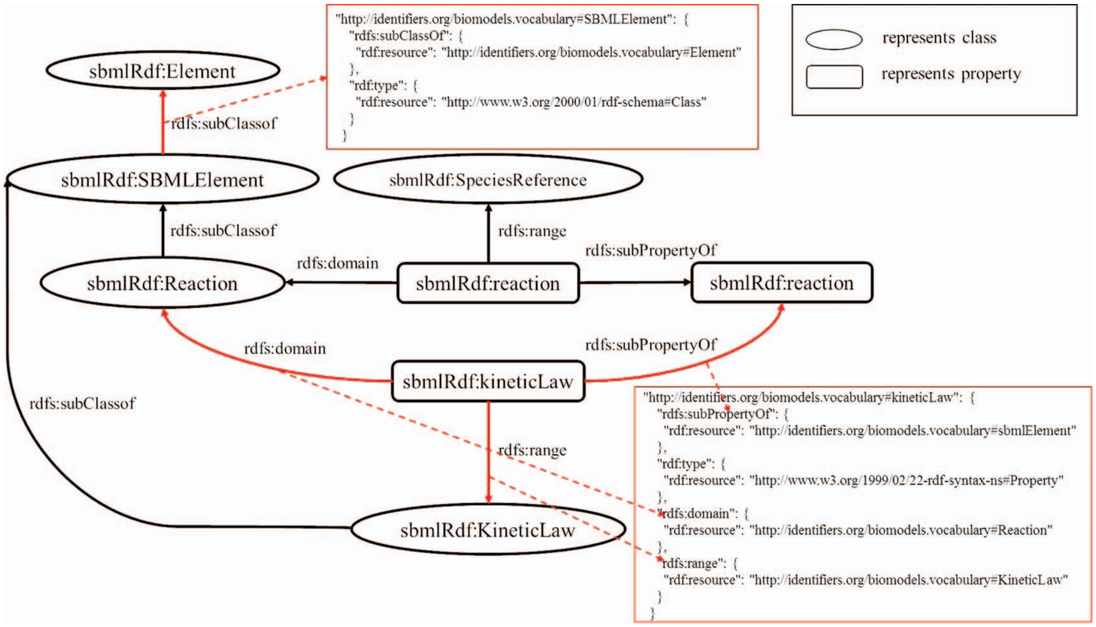
Figure 5 shows a schematic diagram for the process of the domain and range of a property. C1 and C2 are class instances defining the resources denoted by the subjects and objects of the triples whose predicate is P. The relation is extracted from the triple set of the description about P. The relations are transformed into two pairs of key and value, which are put under the JSON object of P.

RULE 9 For repetitive properties of the same resource, the values are merged into an array.

RULE 10 For the property that does not contain a specific value, a blank string is used to describe the value.

Figure 6 shows an example that illustrates the mapping between RDF and JSON. For simplicity, only two representative instances are shown in the figure. The property ‘sbmlRdf:kineticLaw’ is defined with domain property ‘sbmlRdf:Reaction’ and range property ‘sbmlRdf:KineticLaw’ and has a superproperty ‘sbmlRdf:sbmlElement’. According to rules 7 and 8, these properties could be mapped into the JSON object of the property ‘sbmlRdf:kineticLaw’, which has been defined by the base descriptions of the property ‘sbmlRdf:kineticLaw’ according to rules 2 and 3. Similarly, the class ‘sbmlRdf:SBMLElement’ has a superclass ‘sbmlRdf:Element’, and the hierarchical relationship could be mapped into the JSON object of the class ‘sbmlRdf:SBMLElement’ according to rule 6.

Figure 7 gives the data migration process based on the above mapping rules (Algorithm 1). First, we use RDFLib to



```
"http://identifiers.org/biomodels.vocabulary#SBMLElement": {
  "rdfs:subClassOf": {
    "rdf:resource": "http://identifiers.org/biomodels.vocabulary#Element"
  },
  "rdf:type": {
    "rdf:resource": "http://www.w3.org/2000/01/rdf-schema#Class"
  }
}
```

```
"http://identifiers.org/biomodels.vocabulary#kineticLaw": {
  "rdfs:subPropertyOf": {
    "rdf:resource": "http://identifiers.org/biomodels.vocabulary#sbmlElement"
  },
  "rdf:type": {
    "rdf:resource": "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"
  },
  "rdfs:domain": {
    "rdf:resource": "http://identifiers.org/biomodels.vocabulary#Reaction"
  },
  "rdfs:range": {
    "rdf:resource": "http://identifiers.org/biomodels.vocabulary#KineticLaw"
  }
}
```

Figure 6. A mapping instance from RDF to JSON.

Algorithm 1 Transformation from RDF to JSON

```
Input: RDF file rdf_file
Output: Corresponding JSON file json_file
 1: parse rdf_file into a graph model g
 2: create root object resources
 3: create resources["namespaces"] by applying RULE 1
 4: for each s ∈ g.subjects do
 5:   create a dictionary p
 6:   if s is a blank node then
 7:     pass
 8:   end if
 9:   for each t ∈ g.predicate_objects(s) do
10:     create a dictionary tmp
11:     q_name ← g.q_name(t[0])
12:     if t[1] is a LiteralType then
13:       map the properties of t[1] by applying RULE 2 to tmp
14:     end if
15:     if t[1] is a URIRefType then
16:       map the properties of t[1] by applying RULE 2 to tmp
17:     end if
18:     if t[1] is a BNode then
19:       if t[1] ∈ collection then
20:         map the properties of t[1] by applying RULE 5 to tmp
21:       else
22:         map the properties of t[1] by applying RULE 3 and RULE 4 to tmp
23:       end if
24:     end if
25:     p[q_name] ← tmp
26:   end for
27:   resources[s] ← p
28: end for
29: call json.dump to transfer resources to json_file
30: return json_file
```

Figure 7. The mapping algorithm.

parse the RDF file into a graph model. RDFLib is a python library for working with RDF, a simple yet powerful language for representing information. Then, the namespaces used to abbreviate the URI are extracted from the model into the root object by the rule1. In the main loop (4–28), all the resources contained in the model are processed step by step. And the properties associated with each resource are extracted to form the next layer of the loop (9–28). In the inner loop, we map the properties according to rules 2 to 7. Finally, the root object is mapped to the JSON file. Since the algorithm mainly consists of two layers of loop, the time complexity is $O(n^2)$.

RDF2JSON framework and implementation

Figure 8 shows that the RDF2JSON application is designed and implemented on the basis of client–server architecture with the intention of utilizing and separating each module into independent pieces. The server-side component is implemented by Python (version 3.6) providing access to using several libraries such as zerorpc and rdflib. Zerorpc (version 0.9.7) is a flexible remote procedure call implementation, which serves as a known remote server to execute a specified procedure with supplied parameters. The preprocessing of the data conversion uses the rdflib library to parse the input file to get RDF statements before applying the mapping rules. To write the transformed data into the output file we use JSON module in Python. The returned data is saved under the same path of the source.

The client-side component handles interaction with user, which is mainly implemented in electron (version 6.0.0), an open-source framework for creating native applications with web technologies such as JavaScript, HTML5 and CSS. The page architecture, design and functionality use the Bootstrap framework together with several jQuery (version 3.x) plugin to enhance user interactivity. To improve

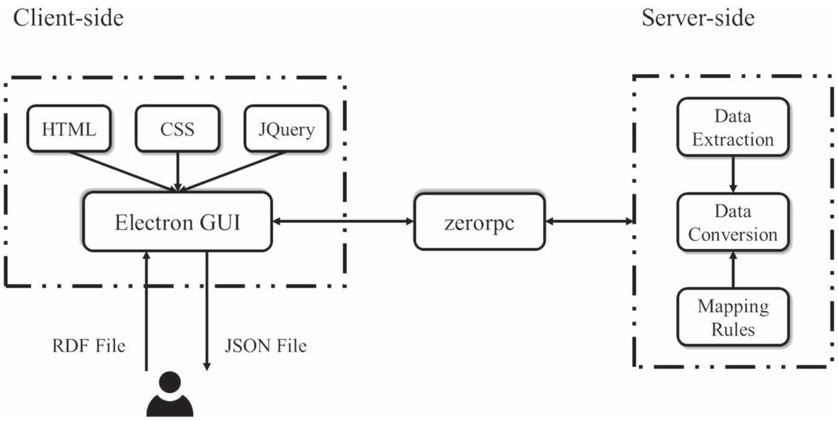


Figure 8. The framework of RDF2JSON.

user friendliness, a common display layout is adopted and maintained between application functions. The file upload area is located on the right side of the page. As a desktop application, it allows a cross-platform compatibility among the most used operating systems (Windows 7 or above) and Linux (version 16.04).

Load JSON file into MongoDB

MongoDB could use the transformed JSON files to store massive records obtained from RDF files. The mongoimport tool is provided for importing the JSON file into MongoDB; the system command line is as follows:

mongoimport –d < database> –c <collection> –file <filename>,where "–d" specifies the name of the database, "–c" specifies the collection to import and "–file" specifies the location and name of a file containing the data to import. Note that the field names of the document cannot contain "." or null, and cannot start with "$" for the system reference. Thus, if these characters appear in the document fields, they need to be replaced with other specific character such as Unicode characters.

After storing the RDF records in MongoDB, the basic MongoDB query mechanism could be used for searching the desired results. In particular, a 'find' function, which receives two parameters in JSON documents, 'query' and 'projection', and returns a cursor to the matching documents, could be used for the searches. The goal of 'query' is to specify the conditions that determine which documents to select, in which the <field>:<value> expressions is used to specify the equality conditions and query operator expressions. The 'projection' is used to specify the fields to return in the documents. The MongoDB query also provides aggregation pipeline methods called 'aggregate' to process a series of documents (such as COUNT, GROUP, SORT, LIMIT, SKIP, etc.) to provide aggregate computations. Besides, the expression operators can be used to construct the query expressions such as arithmetic expression operators, boolean expression operators, etc.

Results

In order to test the performance of RDF2JSON, two real-world data sets UniprotKB and BioModels containing RDF data sources are chosen in our experiments. UniProtKB is a comprehensive resource for protein sequence and annotation data. BioModels is a repository of computational models of biological processes. All the experiments are running on the Ubuntu 18.04 LTS 64-bit operating platform with the following system features: Intel® core i5-8500 CPU @3.00GHz×6 and 32GB main memory. The approaches are programmed in Python 3.6.

Table 1 gives the results of the experiment running on the UniprotKB data set. From Table 1, we observe that, for the experimental data set, JSON provides a compression storage and the compression rate for the tested data is about 40%. The best compression happens in R1 with the rate of 46% (the size of using RDF is 8.75 MB and the size of using JSON is 4.64 MB). As the RDF is an extension of XML and it is a complete markup language, it uses redundant tags for the content descriptions, which may result in redundant storage. JSON organizes data as an ordered list of 'name/value' pairs, which reduces the redundant tags.

Figure 9 illustrates the results of the experiments running on the BioModels data set. In Figure 9, similar to the experimental results obtained in the UniprotKB data set above, a consistent result is observed and the average compression rate is about 14%. We also investigate the scalability of RDF2JSON, by varying number of the input RDF files. Figure 10 reports the running times when the size of input RDF files increase. From the figure, we can see that the running time will increase when the input files increase, and it approximates linear growth.

Table 1. Experimental results running on UniprotKB

	File name	The numbers of subjects	RDF size (MB)	JSON size (MB)
R1	uniprotkb_eukaryota_oxymonadida_66288.rdf	11 662	8.75	4.64
R2	uniprotkb_eukaryota_glaucocystophyceae_38254.rdf	35 767	23.41	13.71
R3	uniprotkb_eukaryota_alveolata_33630_1000000.rdf	763 169	541.61	310.40
R4	uniprotkb_eukaryota_rhizaria_543769.rdf	1 973 509	1307.53	785.36
R5	uniprotkb_eukaryota_parabasalia_5719.rdf	2 424 903	1571.67	950.61
R6	uniprotkb_eukaryota_rhodophyta_2763.rdf	4 031 384	2678.03	1583.91
R7	uniprotkb_eukaryota_opisthokonta_fungi_4751_8000000.rdf	4 448 979	3173.19	1815.40

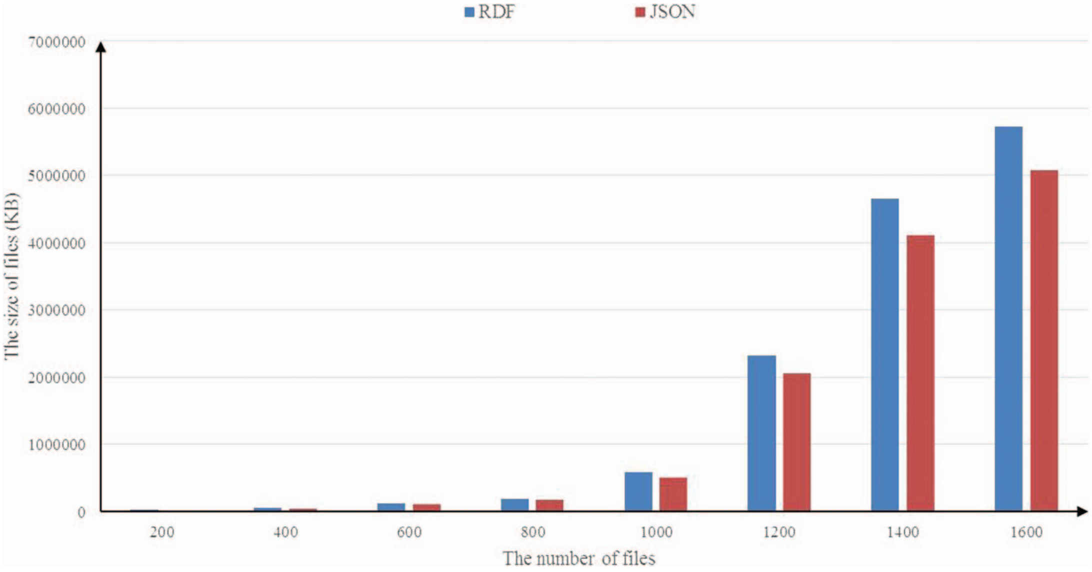


Figure 9. Occupied storage comparisons.

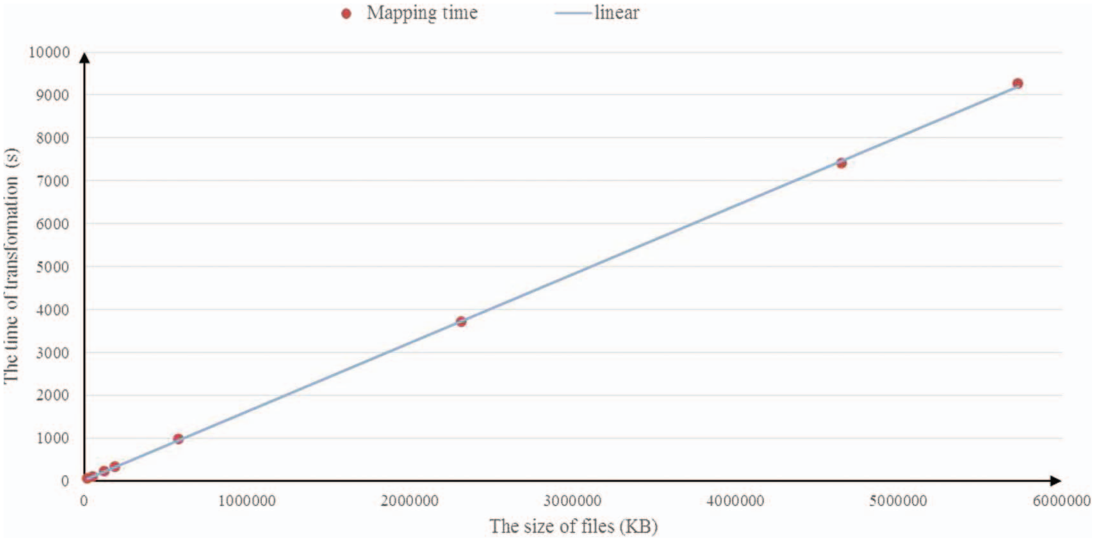


Figure 10. Running times by varying the input size.

Discussion

With the emergence of rapidly increasing size of RDF data sources, there is an urgent need of a novel infrastructure to provide effective supports for future data explosion. In this paper, we deal with the RDF data explosion issue by introducing the JSON model, and show the benefits of using JSON. After parsing the RDF schema, we present a set of mapping rules, which transforms an

RDF schema into the JSON schema, and then propose an effective and specified algorithm to complete data migrations from RDF to JSON. We complement the work with a user-friendly and cross-platform tool RDF2JSON to help users without programming skills. Through the final experiment results, we also demonstrate the performance and advantages of RDF2JSON by using the real-world Uniprot and BioModels data sets. In the future work, we plan to integrate efficient query processing and optimization approach by using Spark/MapReduce, which has promising processing performance over massive data.

Acknowledgements

We gratefully acknowledge Jiao Chen for the helpful comments and suggestions.

Funding

National Key R&D Program of China (2017YFC1200200, 2017YFC1200205, 2018YFC1603800, 2018YFC1603802); National Natural Science Foundation of China (61602130, 61872115), China Postdoctoral Science Foundation funded project (2015M581449, 2016T90294); Heilongjiang Postdoctoral Fund (LBH-Z14089); Natural Science Foundation of Heilongjiang Province of China (QC2015067); Fundamental Research Funds for the Central Universities (HIT.NSRIF.2017036).

Conflict of interest. None declared.

References

1. Alexander,K. (2008) RDF in JSON: a specification for serialising RDF in JSON. In Proceedings of the 4th International Workshop on Scripting for the Semantic Web, Semantic Technology Institutes International, Tenerife, Spain, 1–6.
2. Antezana,E., Kuiper,M. and Mironov,V. (2009) Biological knowledge management: the emerging role of the Semantic Web technologies. *Brief. Bioinform.*, **10**, 392–407.
3. Aswamenakul,C., Buranarach,M. and Saikaew,K.R. (2014) A Review and Design of Framework for Storing and Querying RDF Data Using NoSQL Database. In the 4th Joint International Semantic Technology Conference, Springer. Chiang Mai, Thailand, pp. 144–147.
4. Banane,M., Belangour,A. and El Houssine,L. (2017) Storing RDF data into big data NoSQL databases. In: *First International Conference on Real Time Intelligent Systems*. Springer, Cham, pp. 69–78.
5. Beckett,D. and McBride,B. (2004) RDF/XML syntax specification (revised). Technical report In: *W3C Recommendation, 10(2.3)* https://www.w3.org/TR/REC-rdf-syntax/.
6. Belleau,F., Nolin,M.A., Tourigny,N. *et al.* (2008) Bio2RDF: towards a mashup to build bioinformatics knowledge systems. *J. Biomed. Inform.*, **41**, 706–716.
7. Chiba,H. and Uchiyama,I. (2017) SPANG: a SPARQL client supporting generation and reuse of queries for distributed RDF databases. *BMC Bioinform.*, **18**, 93.
8. Fabregat,A., Sidiropoulos,K., Viteri,G. *et al.* (2017) Reactome diagram viewer: data structures and strategies to boost performance. *Bioinformatics*, **34**, 1208–1214.
9. Flicek,P., Amode,M.R., Barrell,D. *et al.* (2013) Ensembl 2014. *Nucleic Acids Res.*, **42**, D749–D755.
10. García Godoy,M.J., López-Camacho,E., Navas-Delgado,I. *et al.* (2013) Sharing and executing linked data queries in a collaborative environment. *Bioinformatics*, **29**, 1663–1670.
11. Gray,A.J., Groth,P., Loizou,A. *et al.* (2014) Applying linked data approaches to pharmacology: architectural decisions and implementation. *Semant. Web*, **5**, 101–113.
12. Hanwell,M.D., de Jong,W.A. and Harris,C.J. (2017) Open chemistry: RESTful web APIs, JSON, NWChem and the modern web application. *J. Cheminform.*, **9**, 55.
13. Jupp,S., Malone,J., Bolleman,J. *et al.* (2014) The EBI RDF platform: linked open data for the life sciences. *Bioinformatics*, **30**, 1338–1339.
14. Kalogeros,E., Gergatsoulis,M. and Damigos,M. (2018) Document based RDF storage method for efficient parallel query processing. In: *Research Conference on Metadata and Semantics Research*. Springer, Cham, pp. 13–25.
15. Kersey,P.J., Allen,J.E., Allot,A. *et al.* (2017) Ensembl Genomes 2018: an integrated omics infrastructure for non-vertebrate species. *Nucleic Acids Res.*, **46**, D802–D808.
16. Kobayashi,N., Ishii,M., Takahashi,S. *et al.* (2011) Semantic-JSON: a lightweight web service interface for Semantic Web contents integrating multiple life science databases. *Nucleic Acids Res.*, **39**, W533–W540.
17. Kozaki,K., Yamagata,Y., Mizoguchi,R. *et al.* (2017) Disease Compass—a navigation system for disease knowledge based on ontology and linked data techniques. *J. Biomed. Semantics*, **8**, 22.
18. Laird,M.R., Langille,M.G. and Brinkman,F.S. (2015) GenomeD3Plot: a library for rich, interactive visualizations of genomic data in web applications. *Bioinformatics*, **31**, 3348–3349.
19. Li,C., Donizelli,M., Rodriguez,N. *et al.* (2010) BioModels Database: an enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Syst. Biol.*, **4**, 92.
20. Li,Y., Katsipoulakis,N.R., Chandramouli,B. *et al.* (2017) Mison: a fast JSON parser for data analytics. *Proc. VLDB Endowment*, **10**, 1118–1129.
21. Lisena,P. and Troncy,R. (2018) Transforming the JSON Output of SPARQL Queries for Linked Data Clients. In: *Companion of the The Web Conference 2018 on The Web Conference 2018*. International World Wide Web Conferences Steering Committee, Lyon, France. pp. 775–780.
22. Liu,J. and Yan,D.L. (2016) Answering approximate queries over XML data. *IEEE Trans. Fuzzy Syst.*, **24**, 288–305.
23. Liu,J. and Zhang,X.X. (2017) Efficient keyword search in fuzzy XML. *Fuzzy Sets Syst.*, **317**, 68–87.
24. Liu,J., Zhang,X.X. and Zhang,L. (2017) Tree pattern matching in heterogeneous fuzzy XML databases. *Knowl. Based Syst.*, **122**, 119–130.
25. Liu,J., Liu,Q., Zhang,L. *et al.* (2019) Enabling massive XML-based biological data management in HBase. *IEEE/ACM Trans*

Comput. Biol. Bioinform., doi: 10.1109/TCBB.2019.2915811, in press.

26. Maiella,S., Rath,A., Angin,C. *et al.* (2013) Orphanet and its consortium: where to find expert-validated information on rare diseases. *Rev. Neurol.*, **169**, S3–S8.
27. McBride,B. (2004) The resource description framework (RDF) and its vocabulary description language RDFS. In: *Handbook on Ontologies*. Springer, Berlin, Heidelberg, pp. 51–65.
28. Otegui,J. and Guralnick,R.P. (2016) The geospatial data quality REST API for primary biodiversity data. *Bioinformatics*, **32**, 1755–1757.
29. Penha,E.D.S., Iriabho,E., Dussaq,A. *et al.* (2016) Isomorphic semantic mapping of variant call format (VCF2RDF). *Bioinformatics*, **33**, 547–548.
30. Pezoa,F., Reutter,J.L., Suarez,F. *et al.* (2016) Foundations of JSON schema. In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, Montréal, Québec, Canada, pp. 263–273.
31. Queralt-Rosinach,N., Pinero,J., Bravo,À. *et al.* (2016) DisGeNET-RDF: harnessing the innovative power of the Semantic Web to explore the genetic basis of diseases. *Bioinformatics*, **32**, 2236–2238.
32. Ranzinger,R., Aoki-Kinoshita,K.F., Campbell,M.P. *et al.* (2014) GlycoRDF: an ontology to standardize glycomics data in RDF. *Bioinformatics*, **31**, 919–925.
33. Rigden,D.J. and Fernández,X.M. (2017) The 2018 Nucleic Acids Research database issue and the online molecular biology database collection. *Nucleic Acids Res.*, **46**, D1–D7.
34. Sherry,S.T., Ward,M.H., Kholodov,M. *et al.* (2001) dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res.*, **29**, 308–311.
35. Smelter,A., Astra,M. and Moseley,H.N. (2017) A fast and efficient python library for interfacing with the Biological Magnetic Resonance Data Bank. *BMC Bioinformatics*, **18**, 175.
36. UniProt Consortium (2014) UniProt: a hub for protein information. *Nucleic Acids Res.*, **43**, D204–D212.
37. World Wide Web Consortium. (2014) JSON-LD 1.0: a JSON-based serialization for linked data, https://www.w3.org/TR/2014/REC-json-ld-20140116/.
38. Xin,J., Afrasiabi,C., Lelong,S. *et al.* (2018) Cross-linking BioThings APIs through JSON-LD to facilitate knowledge exploration. *BMC Bioinformatics*, **19**, 30.
39. Yates,A., Beal,K., Keenan,S. *et al.* (2014) The Ensembl REST API: Ensembl data for any language. *Bioinformatics*, **31**, 143–145.