Management of Implicit Ontology Changes Generated by Non-conservative JSON Instance Updates in the τJOWL Environment

Management of Implicit Ontology Changes Generated by Non-conservative JSON Instance Updates in the τJOWL Environment

Safa Brahmia[1], Zouhaier Brahmia[1], Fabio Grandi[2] and Rafik Bouaziz[1]

[1]MIRACL laboratory, Faculty of Economics and Management, University of Sfax, Tunisia.
safa.brahmia@gmail.com, zouhaier.brahmia@fsegs.rnu.tn,
rafik.bouaziz@usf.tn

[2]Department of Computer Science and Engineering (DISI), University of Bologna, Italy.
fabio.grandi@unibo.it

Abstract. τJOWL (Temporal OWL 2 from Temporal JSON) is a framework we proposed to allow users of Big Data projects to automatically create, with the closed world assumption (CWA), a temporal OWL 2 ontology from time-varying JSON-based Big Data. Such an ontology, providing the semantics to the data, facilitates complex tasks like Big Data querying, analytics and reasoning, in an environment that supports temporal versioning at both data instance and schema levels. Update operations on temporal JSON Big Data may not respect the JSON Schema associated to them, like renaming a JSON object or a JSON object member, or replacing the current value of an object member with a new one that is not conformant to its type as specified in the corresponding JSON Schema. These update operations are called "non-conservative updates" and require implicit JSON Schema changes to be performed so that they could be correctly executed. Such JSON Schema changes need then to be propagated to the OWL 2 ontology in order to maintain semantic alignment. To this purpose, in this paper, we propose an approach for automatic management of implicit OWL 2 ontology schema change operations that are generated by non-conservative updates to temporal JSON Big Data instances, in the τJOWL framework.

Keywords: Big Data, JSON, JSON Schema, NoSQL, Ontology, OWL 2, Instance update, Schema change, Implicit schema change, τJOWL

1 Introduction

Nowadays, big data [1,2] are being widely used in several application domains like e-business, e-health, online social networks, cloud computing, Internet of Things, smart grids, and smart cities. JSON [3] is among the lightweight and semi-structured data formats, including also XML and YAML, which are broadly used to represent, exchange and store Big Data. Although JSON Big Data are in general schemaless [4], schemas for them, specified using the JSON Schema language [5,6] recommended by the Internet Engineering Task Force (IETF), can be discovered/extracted through the use of some appropriate tool (like json-schema-inferrer [7], Schema Guru [8], or

Clojure JSON Schema Validator & Generator [9]). Moreover, Big Data can also be accompanied by an ontology [10], which can be very useful to provide the semantics of these data [11], from one hand, and to make easy and more efficient some complex tasks t like Big Data querying, Big Data analytics, and reasoning over Big Data, from the other hand.

The adoption of an ontology allows the formal specification of the data semantics in an information system or of the knowledge of an application domain. It is widely used in Semantic Web-based software, knowledge base management systems, and artificial intelligence-based systems. It facilitates data access, query answering, and knowledge inference. The OWL 2 Web Ontology Language [12], informally known as OWL 2, is the World Wide Web Consortium (W3C) recommendation for ontology specification in the Semantic Web. In the literature, two viewpoints have been proposed for the management of inferences and deductions from ontology statements: the Open World Assumption (OWA) [13] and the Closed World Assumption (CWA) [14]. The OWA, which means that "what is not known to be true or false is unknown", is in general applied in environments where data are considered to be incomplete (e.g., Knowledge Bases, Semantic Web repositories), whereas the CWA, which means that "what is not known to be true must be false", is in general applied in environments where data are considered to be complete (e.g., traditional databases). It is worth mentioning that the CWA can be used for ontology management via the notion of Data Box (DBox) [15]. Furthermore, with the CWA, an ontology definition (concepts, relations between concepts, axioms, …) acts as a database schema and, therefore, can be named an "ontology schema"; the ontology individuals are instances of this ontology schema.

Besides, Big Data are rapidly evolving over time, due to their "velocity" feature. In addition to this fact, several JSON-based Big Data applications require keeping track of the whole history of updates that are executed on these data. Consequently, JSON-based Big Data representation and storage need to become temporal: Big Data values are time-dependent and multi-version, so that each of them can be made of several consecutive temporal versions. Accordingly, it becomes difficult not only to manage (query, update, …) and analyze JSON Big Data, but also to formally define their semantics. Hence, it is of great help to have systems and tools that facilitate the definition of a temporal and a multi-versions ontology for temporal and multi-versions JSON-based Big Data. For that reason, and after studying the state of the art of ontologies for Big Data [16-30] and noticing that there is no work that have dealt with an automatic building of a temporal ontology from some temporal and multi-version Big Data, we have proposed in [31] a framework, called τJOWL (Temporal OWL 2 from Temporal JSON), which allows users of a Big Data project (i) to automatically create a temporal OWL 2 ontology of data, with the CWA viewpoint, from time-varying JSON-based Big Data via the use of an intermediary JSON Schema, and (ii) to manage the incremental maintenance of such a temporal ontology schema following the evolution of the underlying temporal JSON Big Data, in a multi-version setting.

As far as instance updates in τJOWL are concerned, we distinguished in [31] between conservative updates, which have no effect on the (JSON) schema of the updated JSON instance document, and non-conservative updates, which have an impact

on such a schema since they require that some schema changes must be performed before executing the instance updates. Hence, the versioning of temporal ontology schemas is driven by non-conservative (temporal) updates to JSON Big Data instances. Moreover, since in [31] we have focused on the automatic building of a new temporal OWL 2 ontology schema for a temporal JSON instance document, in the present paper we complete our framework by focusing on the ontology maintenance issue. More precisely, we deal with non-conservative (temporal) JSON instance updates and show how they trigger ontology schema changes, which produce a new (temporal) ontology schema version.

The rest of the paper is organized as follows. Sec. 2 briefly recalls the τJOWL environment. Sec. 3 proposes our approach for implicit ontology schema changes that result from JSON-based big data instance updates, in τJOWL which supports temporal versioning at both instance and schema levels. Sec. 4 illustrates our proposal via an application example. Sec. 5 concludes the paper.

2 The τJOWL Environment

This section provides an overview of our τJOWL framework (as shown in Fig. 1).

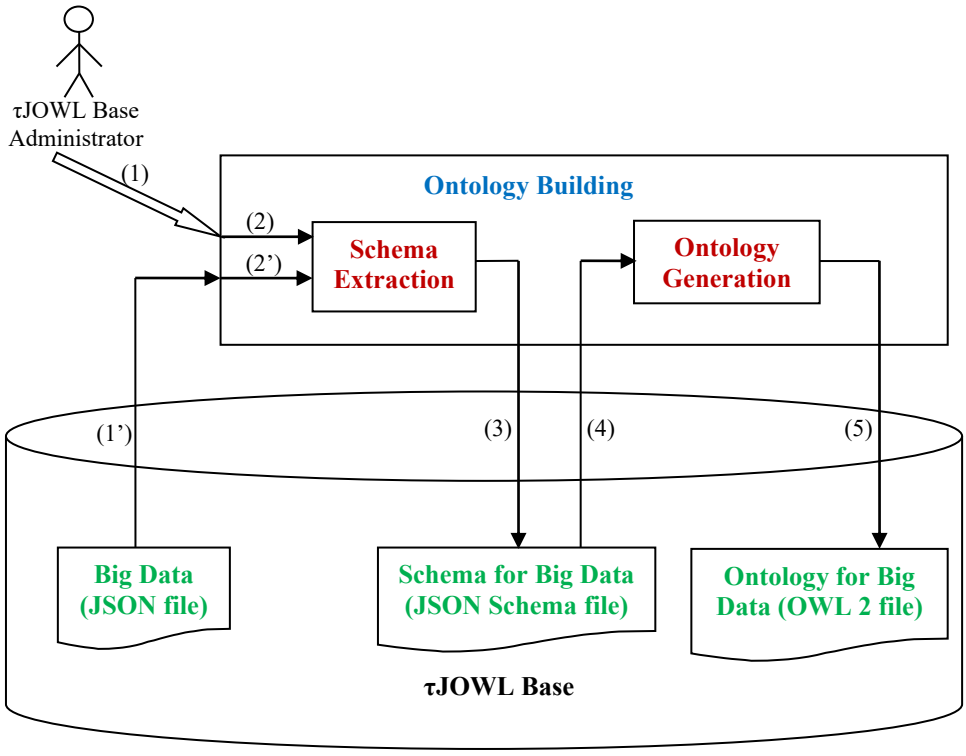

Fig. 1. The τJOWL environment

To build a temporal OWL 2 ontology of data, according to the CWA philosophy, for some temporal JSON-based Big Data instances, the τJOWL Base Administrator (arrows 1 and 2) starts by supplying a JSON instance document [3] (arrows 1' and 2'), which stores temporal big data, as input to the "Ontology Building" module. This latter first generates a JSON Schema [5,6] (arrows 3 and 4) for the supplied JSON instance document, by calling the sub-module "Schema Extraction", and then converts this JSON Schema file into an OWL 2 ontology schema (arrow 5), by calling the sub-module "Ontology Generation".

More details on the architecture of τJOWL and the implementation of the two sub-modules "Schema Extraction" and "Ontology Generation" can be found in [31, Sec. 2]. Notice that the τJOWL Base is designed to store both big data and their ontologies. For that reason, to support the storage, at the same time, of JSON files, JSON Schema files, and OWL 2 files, the τJOWL Base could be either a (document-oriented) NoSQL database [40,41], or a multi-model database [42].

Notice also that τJOWL considers OWL 2 ontologies with an RDF/XML syntax [32] that is the only syntax that must be supported by all OWL 2 tools, as required in the OWL 2 specification document [33].

3 Management of Implicit Ontology Schema Changes Triggered by Non-conservative JSON Instance Updates

In this section, we present our approach for managing implicit ontology schema changes generated by non-conservative JSON instance updates, in τJOWL.

As mentioned above, τJOWL is temporal and supports versioning at both instance and schema levels. Consequently, each instance update IU, when applied on some JSON instance document version D_V_i ($i>=1$) that is conformant to some JSON Schema version S_V_j ($j>=1$) and some OWL 2 ontology schema version O_V_j associated to (and derived from) S_V_j, leads to a new JSON instance document version D_V_{i+1}. If D_V_{i+1} is still conformant to S_V_j, then IU is conservative; otherwise, IU is non-conservative and, as a consequence, the execution of two tasks is implicitly triggered (within the same update transaction): (1) a new JSON Schema version S_V_{j+1}, to which D_V_{i+1} is conformant, is generated, and (2) a new OWL 2 ontology schema version O_V_{j+1} is automatically derived from S_V_{j+1}.

Notice that the instance update IU is a sequence of JSON instance update operations, specified in some JSON update language like JUpdate [34] or τJUpdate [35]. Therefore, IU must include at least one instance update operation that has an effect on the schema version of the updated JSON document version, like renaming a JSON object or a JSON object member, or replacing the current value of an object member with a new value that is not conformant to the type of this object member (specified in the corresponding JSON Schema version).

Non-conservative instance updates require then implicit JSON Schema changes to be executed, at schema level, so that they could be correctly performed. Moreover, the execution of such implicit JSON Schema changes has an immediate impact on the ontology side of τJOWL: the (temporal) OWL 2 ontology schema, which is related to

the implicitly changed (temporal) JSON Schema, is consequently changed in an implicit and automatic way. Notice that, for changing ontology schema, we are re-using the set of ontology schema change operations that we have defined in our previous work [38,39]. In these previous papers, we have shown that each ontology schema change operation is sound, i.e., its application to a (consistent) ontology leads to a new consistent ontology (and does not give rise to any inconsistency).

The functioning of our approach is formalized via the algorithm shown in Fig. 2 and named "UpdateJSONInstancesWithImplicitOntologySchemaVersioning".

ALGORITHM UpdateJSONInstancesWithImplicitOntologySchemaVersioning

```
  Inputs: JIU, JID_Vn, JSD_Vm, OSD_Vm
  Outputs: JID_Vn+1, IJSC, JSD_Vm+1, OSC, OSD_Vm+1
BEGIN
  UpdateJSONInstanceDocument(JIU, JID_Vn, JID_Vn+1, IJSC);
  Display("Execution of instance updates.");
  If (empty(IJSC)==false) Then
    GenerateNewJSONSchemaVersion(IJSC, JSD_Vm, JSD_Vm+1);
    PropagateSchemaChangesToJSONInstances(IJSC, JSD_Vm);
    DeriveOntologySchemaChanges(IJSC, JSD_Vm, OSD_Vm, OSC);
    GenerateNewOntologySchemaVersion(OSC, OSD_Vm, OSD_Vm+1);
    Display("With execution of implicit schema changes.");
  Else
    Display("No implicit schema changes are required.");
  EndIf
END
```

Fig. 2. The algorithm for applying updates to a JSON instance document version possibly with implicit ontology schema versioning

This algorithm uses a set of input and output variables and calls a function and a set of procedures. They are all described below.

- Input variables:
  - JIU: a (valid) sequence of JSON instance updates specified by the τJOWL base administrator using the JUpdate or the τJUpdate language.
  - JID_V_n: a (valid) JSON file that represents the current JSON instance document version (number n, with n>=1).
  - JSD_V_m: a (valid) JSON Schema file that represents the current JSON Schema

document version (number m, with m>=1) to which JID_V_n is conformant.

- OSD_V_m: a (valid) OWL 2 file that represents the current OWL 2 ontology schema version (number m) which results from the conversion of JSD_V_m to an OWL 2 file.

- Output variables:
  - JID_V_{n+1}: a (valid) JSON file that represents the new JSON instance document version (number n+1, with n>=1).
  - IJSC: a (valid) sequence of implicit JSON Schema changes automatically generated by the system during the execution of the sequence of JSON instance updates JIU.
  - JSD_V_{m+1}: a (valid) JSON Schema file that represents the new JSON Schema document version (number m+1, with m>=1) to which JID_V_{n+1} is conformant.
  - OSC: a (valid) sequence of OWL 2 ontology schema changes automatically generated by the system based on the sequence of implicit JSON Schema changes IJSC.
  - OSD_V_{m+1}: a (valid) OWL 2 file that represents the new OWL 2 ontology schema version (number m+1) which results from the conversion of JSD_V_{m+1} to an OWL 2 file.
- Function:
  - empty(IJSC): it returns either true if the sequence of implicit JSON Schema changes IJSC (passed as argument) is empty, or false otherwise.
- Procedures:
  - UpdateJSONInstanceDocument(JIU, JID_V_n, JID_V_{n+1}, IJSC): it applies the sequence of JSON instance updates JIU on the JSON instance document version JID_V_n (n>=1) to produce the new JSON instance document version JID_V_{n+1} and possibly a sequence of implicit JSON Schema changes IJSC that could require interaction with the τJOWL base administrator.
  - Display(message): it displays the message (a string) passed as argument.
  - GenerateNewJSONSchemaVersion(IJSC, JSD_V_m, JSD_V_{m+1}): it creates a new JSON Schema version JSD_V_{m+1} (m>=1) by applying the sequence of implicit JSON Schema changes IJSC on the current JSON Schema version JSD_V_m.
  - PropagateSchemaChangesToJSONInstances(IJSC, JSD_V_m): it propagates the effects of the sequence of implicit JSON Schema changes IJSC to all JSON instance documents that are conformant to the previous JSON Schema version JSD_V_m (m>=1). Hence, this procedure generates a new version of each one of these JSON instance documents, which must be conformant to the new JSON

Schema version JSD_V_{m+1}.

- DeriveOntologySchemaChanges(IJSC, JSD_V_m, OSD_V_m, OSC): it derives, from the sequence of implicit JSON Schema changes IJSC that act on the current JSON Schema version JSD_V_m, the sequence of OWL 2 ontology schema changes OSC that act on the current OWL 2 ontology schema version OSD_V_m.
- GenerateNewOntologySchemaVersion(OSC, OSD_V_m, OSD_V_{m+1}): it creates a new OWL 2 ontology schema version OSD_V_{m+1} (m>=1) by applying the sequence of OWL 2 ontology schema changes OSC on the current OWL 2 ontology schema version OSD_V_m.

4 Application Example

To exemplify the functioning of our approach, let us consider the case of a JSON-based NoSQL data store used by a scientific research laboratory for the management of researchers' data. Let us assume that, in order to make such a management more "intelligent" as far as access, query and analysis of researchers' data are concerned, the director of the laboratory decides to deploy an ontology of data, with the CWA, within a τJOWL framework.

Assume that on March 01, 2023, the τJOWL Base Administrator created (or imported) a JSON instance document, named "ResearchersJSONInstances_V1.json" (as shown in Fig. 3), which stores data on researchers (the ORCID, the name, the discipline, characterized by an id and a description, and the h-index of each researcher). Due to space limitations, this example presents only one researcher named "Tarak Ziad", his ORCID is "0000-1111-2222-3333", his discipline is "Computer Science" (description) with an id equal to 1, and his h-index is 9. After that, he/she invoked the "Ontology Building" module (as shown in Fig. 1) of τJOWL to automatically build an OWL 2 ontology for these JSON data. In fact, this module starts by extracting the JSON Schema file, named "ResearchersJSONSchema_V1.json" (as shown in Fig. 4), of the provided JSON instance document (of Fig. 3), by calling the "Schema Extraction" sub-module, and then it generates the OWL 2 ontology file, named "ResearchersOntologySchema_V1.owl" (as shown in Fig. 5), corresponding to the extracted JSON schema, by calling the "Ontology Generation" sub-module.

```
{ "researcher":{
    "ORCID": "0000-1111-2222-3333",
    "name": "Tarak Ziad",
    "discipline": { "id": 1,
                    "description": "Computer Science"},
    "h-index": 9 } }
```

Fig. 3. The JSON instance document of researchers ("ResearchersJSONInstances_V1.json") on March 01, 2023

```
{ "type": "object",
  "properties":
    { "researcher":
        { "type": "object",
          "properties":
            { "ORCID": { "type": "string" },
              "name": { "type": "string" },
              "discipline":
                { "type": "object",
                  "properties":
                    { "id": { "type": "integer" },
                      "description": { "type": "string" } } },
              "h-index": { "type": "integer" } } } } }
```

Fig. 4. The generated JSON Schema of researchers ("ResearchersJSONSchema_V1.json") on March 01, 2023

```
<rdf:RDF>
 <owl:Ontology rdf:about="http://my_onto/researcher_ontology#">
  <owl:Class rdf:about="researcher"/>
  <owl:DatatypeProperty rdf:about="ORCID">
   <rdfs:domain rdf:resource="researcher"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="name">
   <rdfs:domain rdf:resource="researcher"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:Class rdf:about="discipline"/>
  <owl:DatatypeProperty rdf:about="id">
   <rdfs:domain rdf:resource="discipline"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="description">
   <rdfs:domain rdf:resource="discipline"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:about="researcher_Has_discipline">
   <rdfs:domain rdf:resource="researcher"/>
   <rdfs:range rdf:resource="discipline"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:about="h-index">
   <rdfs:domain rdf:resource="researcher"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
```

```
  </owl:Ontology>
</rdf:RDF>
```

Fig. 5. The generated OWL 2 ontology schema of researchers ("ResearchersOntologySchema_V1.owl") on March 01, 2023

Furthermore, assume that on April 01, 2023, the τJOWL Base Administrator updated the current JSON instance document version (as shown in Fig. 6), by applying the following JSON instance updates on the researcher whose ORCID is "0000-1111-2222-3333":

- update the value of the object member "name" from "Tarak Ziad" to "Tarak Ben Ziad";
- replace the old value of the object member "discipline" (having an "object" type) with a new value "Computer Science" with string type;
- add to the "researcher" object two new object members: the first "citations" with the integer value 222 and the second "publications" with the integer value 39;
- rename the object members "name" and "h-index" to "fullName" and "scopusH-index", respectively.

In a more technical way, we suppose that the τJOWL Base Administrator has specified the above instance updates using the JUpdate language [34]. The sequence of JSON instance update operations could be as follows:

```
UPDATE ResearchersJSONInstances_V1.json
OBJECT $.researcher[?(@.ORCID=='0000-1111-2222-3333')]
SET name="Tarak Ben Ziad", discipline="Computer Science";

ALTER DOCUMENT ResearchersJSONInstances_V1.json
OBJECT $.researcher[?(@.ORCID=='0000-1111-2222-3333')]
ADD MEMBER citations VALUE 222;

ALTER DOCUMENT ResearchersJSONInstances_V1.json
OBJECT $.researcher[?(@.ORCID=='0000-1111-2222-3333')]
ADD MEMBER publications VALUE 39;

ALTER DOCUMENT ResearchersJSONInstances_V1.json
OBJECT $
RENAME MEMBER $.researcher.name TO fullName;

ALTER DOCUMENT ResearchersJSONInstances_V1.json
OBJECT $
RENAME MEMBER $.researcher.h-index TO scopusH-index;
```

Notice that, in the above sequence, except the first instance update operation that is conservative (i.e., it respects the JSON Schema version, as shown in Fig. 4, associated

to the JSON instance document version which is under update), the other operations are all non-conservative.

The execution of the above sequence of instance update operations on the first JSON instance document version of researchers ("ResearchersJSONInstances_V1.json", shown in Fig. 3) generates a new version named "ResearchersJSONInstances_V2.json" (as shown in Fig. 6). Furthermore, and since the above sequence includes non-conservative operations, such an execution also produces, in a transparent way, a sequence of implicit JSON Schema change operations. This latter is automatically applied on the first JSON Schema version ("ResearchersJSONSchema_V1.json", shown in Fig. 4) to obtain a new JSON Schema version, named "ResearchersJSONSchema_V2.json" (as shown in Fig. 7), for the new JSON instance document version (shown in Fig. 6) Then, and within the same update transaction, the system also derives, from the generated sequence of implicit JSON Schema change operations, an equivalent sequence of OWL 2 ontology schema change operations, and applies it on the first OWL 2 ontology schema version ("ResearchersOntologySchema_V1.owl", shown in Fig. 5) to obtain a new OWL 2 ontology schema version, named "ResearchersOntologySchema_V2.owl" (as shown in Fig. 8). Notice that "ResearchersOntologySchema_V2.owl" is the second OWL 2 ontology schema version corresponding to the updated JSON Big Data instances stored in "ResearchersJSONInstances_V2.json" of Fig. 6. Changes are evidenced in red bold typeface.

Notice that we have defined all necessary JSON Schema change operations in our previous work on schema versioning in a temporal JSON-based NoSQL setting [36,37], and all useful OWL 2 ontology schema change operations in our previous work on schema versioning in a temporal OWL 2 Semantic Web context [38,39].

As for the implicit relation between non-conservative JSON instance update operations, JSON Schema change operations, and OWL 2 ontology schema change operations, we illustrate it below via the simple case of rename something. For example, for the following JSON instance update operation:

```
ALTER DOCUMENT ResearchersJSONInstances_V1.json
OBJECT $
RENAME MEMBER $.researcher.name TO fullName;
```

the following JSON Schema change operation will be implicitly generated:

```
RenameProperty(ResearchersJSONSchema_V1.json,
     $.properties.researcher.properties.name, fullName)
```

and the following OWL 2 ontology schema change operation will be also derived:

```
RenameDataTypeProperty(ResearchersOntologySchema_V1.owl,
     name, fullName)
```

```
{ "researcher":{
    "ORCID": "0000-1111-2222-3333",
    "fullName": "Tarak Ben Ziad",
    "discipline": "Computer Science",
```

```
    "scopusH-index": 10,
    "citations": 222,
    "publications": 39 } }
```

Fig. 6. The JSON instance document of researchers ("ResearchersJSONInstances_V2.json") on April 01, 2023

```
{ "type": "object",
  "properties":
    { "researcher":
        { "type": "object",
          "properties":
            { "ORCID": { "type": "string" },
              "fullName": { "type": "string" },
              "discipline": { "type": "string" },
              "scopusH-index": { "type": "integer" },
              "citations": { "type": "integer" },
              "publications": { "type": "integer" } } } } }
```

Fig. 7. The generated JSON Schema of researchers ("ResearchersJSONSchema_V2.json") on April 01, 2023

```
<rdf:RDF>
 <owl:Ontology rdf:about="http://my_onto/researcher_ontology#">
  <owl:Class rdf:about="researcher"/>
  <owl:DatatypeProperty rdf:about="ORCID">
   <rdfs:domain rdf:resource="researcher"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="fullName">
   <rdfs:domain rdf:resource="researcher"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="discipline">
   <rdfs:domain rdf:resource="researcher"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="scopusH-index">
   <rdfs:domain rdf:resource="researcher"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="citations">
   <rdfs:domain rdf:resource="researcher"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="publications">
```

```
    <rdfs:domain rdf:resource="researcher"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
   </owl:DatatypeProperty>
  </owl:Ontology>
</rdf:RDF>
```

Fig. 8. The derived OWL 2 ontology of researchers ("ResearchersOntologySchema_V2.owl") on April 01, 2023

5 Conclusion

In this paper, we have proposed an approach for an automatic management of implicit ontology schema changes that are triggered by non-conservative updates to JSON-based Big Data instances, in the τJOWL environment.

We think that our approach makes the management of ontology schema versioning more flexible and more user-friendly, as it allows to apply JSON instance updates that do not "respect" the current implicit JSON schema version and consequently the current ontology schema version. Moreover, our proposal facilitates the specification, in a progressive manner and driven by instance updates, of the correct ontology schema of some JSON-based Big Data instance document whose complete and precise structure is not fully known at its creation time. We think also that our approach could easily be applied in a context that is different from τJOWL since it is designed in an environment that is independent from any technology (i.e., from any DBMS, API, software, JSON tool, programming language, …). It depends only on (i) the JSON format, which is the most widely used model for big data storage and management, (ii) the JSON Schema language, which is recommended by the IETF for defining structures of JSON documents, and (iii) the OWL 2 language, which is recommended by the W3C for Semantic Web ontologies.

In order to show the feasibility of our proposal, we plan in the near future to implement our approach as a new module within the τJOWL-Manager tool that is being developed to support the whole τJOWL framework. Such a module will allow us to experimentally evaluate the scalability and performances of our proposal.

Last but not least, in the future we plan to deal with the effects of implicit ontology schema changes on the programs which are using these ontologies. In fact, when an ontology schema evolves, all ontology instance update statements (specified e.g. in SPARQL 1.1 Update [43]) and ontology queries (specified e.g. in SPARQL [44], T-SPARQL [45], SWRL [46], SQWRL [47], or τSQWRL [48]), which are written against such an ontology schema, should be revised in order to take into account the new ontology schema version. These revisions could be automatically performed on the program source codes, in parallel with the execution of implicit ontology schema changes.

References

1. Rao, T. R., Mitra, P., Bhatt, R., and Goswami, A. (2019). The big data system, components, tools, and technologies: a survey. *Knowledge and Information Systems, 60*(3), 1165-1245.
2. Davoudian, A., and Liu, M. (2020). Big Data Systems: A Software Engineering Perspective. *ACM Computing Surveys (CSUR), 53*(5), Article 110.
3. IETF. (2017). The JavaScript Object Notation (JSON) Data Interchange Format. Internet Standards Track document, December 2017. https://tools.ietf.org/html/rfc8259 (accessed: 2023-03-28)
4. Banerjee, S., Shaw, R., Sarkar, A., and Debnath, N. C. (2015). Towards logical level design of Big Data. In *Proceedings of the IEEE 13th International Conference on Industrial Informatics (INDIN 2015)*, Cambridge, UK, 22-24 July 2015, pp. 1665-1671. IEEE.
5. Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., and Vrgoč, D. (2016). Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web (WWW'2016)*, Montréal, Québec, Canada, 11-15 April 2016, pp. 263-273.
6. IETF. (2018). JSON Schema: A Media Type for Describing JSON Documents. Internet-Draft, 19 March 2018. https://json-schema.org/latest/json-schema-core.html (accessed: 2023-03-28)
7. json-schema-inferrer: java library for inferreing JSON schema from sample JSONs. https://github.com/saasquatch/json-schema-inferrer (accessed: 2023-03-28)
8. Schema Guru. https://github.com/snowplow/schema-guru (accessed: 2023-03-28)
9. Clojure JSON Schema Validator & Generator. https://github.com/luposlip/json-schema (accessed: 2023-03-28)
10. Guarino, N., (Ed.) (1998). *Formal Ontology in Information Systems*. IOS Press, Amsterdam, Netherlands.
11. Ceravolo, P., Azzini, A., Angelini, M., Catarci, T., Cudré-Mauroux, P., Damiani, E., Mazak, A., Van Keulen, M., Jarrar, M., Santucci, G., Sattler, K.-U., Scannapieco, M., Wimmer, M., Wrembel, R., and Zaraket, F. (2018). Big data semantics. *Journal on Data Semantics, 7*, 65-85.
12. W3C. (2012). OWL 2 Web Ontology Language – Primer (Second Edition). W3C Recommendation, 11 December 2012. http://www.w3.org/TR/owl2-primer/ (accessed: 2023-03-28)
13. Patel-Schneider, P. F., and Horrocks I. (2007). A comparison of two modelling paradigms in the Semantic Web. *Journal of Web Semantics, 5*(4), 240-250.
14. Etzioni, O., Golden, K., and Weld, D. S. (1997). Sound and efficient closed-world reasoning for planning. *Artificial Intelligence, 89*(1-2), 113-148.
15. Seylan, İ., Franconi, E., and De Bruijn, J. (2009). Effective query rewriting with ontologies over DBoxes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, Pasadena, CA, USA, 11-17 July 2009, pp. 923-929.
16. Hoppe, A., Nicolle, C., and Roxin, A. (2013). Automatic ontology-based user profile learning from heterogeneous web resources in a big data context. *Proceedings of the VLDB Endowment, 6*(12), 1428-1433.
17. Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., and Horrocks, I. (2013). OptiqueVQS: towards an ontology-based visual query system for big data. In *Proceedings of the 5th International Conference on Management of Emergent Digital EcoSystems (MEDES'2013)*, Luxembourg, Luxembourg, 29-31 October 2013, pp. 119-126.
18. Jayapandian, C., Chen, C. H., Dabir, A., Lhatoo, S., Zhang, G. Q., and Sahoo, S. S. (2014). Domain ontology as conceptual model for big data management: application in biomedical informatics. In *Proceedings of the 2014 International Conference on Conceptual Modeling*

(ER 2014), Atlanta, GA, USA, 27-29 October 2014, pp. 144-157. Springer, Cham.
19. Shah, T., Rabhi, F., and Ray, P. (2015). Investigating an ontology-based approach for Big Data analysis of inter-dependent medical and oral health conditions. *Cluster Computing, 18*(1), 351-367.
20. Verhoosel, J. P., and Spek, J. (2016). Applying ontologies in the dairy farming domain for big data analysis. In *Joint Proceedings of the 3rd Stream Reasoning (SR 2016) and the 1st Semantic Web Technologies for the Internet of Things (SWIT 2016) workshops co-located with 15th International Semantic Web Conference (ISWC 2016)*, Kobe, Japan, 17-18 October 2016, pp. 91-100.
21. Kim, A. R., Park, H. A., and Song, T. M. (2017). Development and evaluation of an obesity ontology for social big data analysis. *Healthcare Informatics Research, 23*(3), 159-168.
22. Abbes, H., and Gargouri, F. (2018). MongoDB-based modular ontology building for big data integration. *Journal on Data Semantics, 7*(1), 1-27.
23. Globa, L. S., Novogrudska, R. L., and Koval, A. V. (2018). Ontology model of telecom operator big data. In *Proceedings of the 2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom 2018)*, Batumi, Georgia, 4-7 June 2018, pp. 1-5. IEEE.
24. Wongthongtham, P., and Salih, B. A. (2018). Ontology-based approach for identifying the credibility domain in social Big Data. *Journal of Organizational Computing and Electronic Commerce, 28*(4), 354-377.
25. Nadal, S., Romero, O., Abelló, A., Vassiliadis, P., and Vansummeren, S. (2019). An integration-oriented ontology to govern evolution in big data ecosystems. *Information Systems, 79*, 3-19.
26. Rani, P. S., Suresh, R. M., and Sethukarasi, R. (2019). Multi-level semantic annotation and unified data integration using semantic web ontology in big data processing. *Cluster Computing, 22*(5), 10401-10413.
27. Djebouri, D., and Keskes, N. (2020). Exploitation of ontological approaches in Big Data: A State of the Art. In *Proceedings of the 10th International Conference on Information Systems and Technologies (ICIST'2020)*, Lecce, Italy, 4-5 June 2020, Article no. 45, pp. 1-6.
28. Aghdam, M. Y., Tabbakh, S. R. K., and Chabok, S. J. M. (2021). Ontology generation for flight safety messages in air traffic management. *Journal of Big Data, 8*(1), 1-21.
29. Mhammedi, S., El Massari, H., and Gherabi, N. (2021). Cb2Onto: OWL Ontology Learning Approach from Couchbase. In: Gherabi N., and Kacprzyk J. (Eds.), *Intelligent Systems in Big Data, Semantic Web and Machine Learning*. Advances in Intelligent Systems and Computing (AISC), vol. 1344, pp. 95-110. Springer, Cham.
30. Mountasser, I., Ouhbi, B., Hdioud, F., and Frikh, B. (2021). Semantic-based Big Data integration framework using scalable distributed ontology matching strategy. *Distributed and Parallel Databases, 39*(4), 891-937.
31. Brahmia, Z., Grandi, F., and Bouaziz, R. (2022). τJOWL: A Systematic Approach to Build and Evolve a Temporal OWL 2 Ontology Based on Temporal JSON Big Data. *Big Data Mining and Analytics, 5*(4), 271-281.
32. W3C. (2004). RDF/XML Syntax Specification (Revised). W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ (accessed: 2023-03-28)
33. W3C. (2012). OWL 2 Web Ontology Language – Document Overview (Second Edition). W3C Recommendation, 11 December 2012. http://www.w3.org/TR/owl2-overview/ (accessed: 2023-03-28)
34. Brahmia, Z., Brahmia, S., Grandi, F., and Bouaziz, R. (2022). JUpdate: a JSON update language. *Electronics, 11*(4), 508.

35. Brahmia, Z., Grandi, F., Brahmia, S., and Bouaziz, R. (2022). τJUpdate: A Temporal Update Language for JSON Data. In *Proceedings of the 11th International Conference on Model and Data Engineering (MEDI 2022)*, Cairo, Egypt, 21-24 November 2022, pp. 250-263. Cham: Springer Nature Switzerland.
36. Brahmia, Z., Brahmia, S., Grandi, F., and Bouaziz, R. (2021). Versioning schemas of JSON-based conventional and temporal big data through high-level operations in the τJSchema framework. *International Journal of Cloud Computing, 10*(5-6), 442-479.
37. Brahmia, S., Brahmia, Z., Grandi, F., and Bouaziz, R. (2017). Temporal JSON Schema Versioning in the τJSchema Framework. *Journal of Digital Information Management, 15*(4), 179-202.
38. Zekri, A., Brahmia, Z., Grandi, F., and Bouaziz, R. (2016). τOWL: A systematic approach to temporal versioning of semantic web ontologies. *Journal on Data Semantics, 5*(3), 141-163.
39. Zekri, A., Brahmia, Z., Grandi, F., and Bouaziz, R. (2017). Temporal schema versioning in τOWL: a systematic approach for the management of time-varying knowledge. *Journal of Decision systems, 26*(2), 113-137.
40. Davoudian, A., Chen, L., and Liu, M. (2018). A survey on NoSQL stores. *ACM Computing Surveys (CSUR), 51*(2), 1-43.
41. NoSQL Databases List by Hosting Data – Updated 2023. https://hostingdata.co.uk/nosql-database/
42. Lu, J., and Holubová, I. (2019). Multi-model databases: a new journey to handle the variety of data. *ACM Computing Surveys (CSUR), 52*(3), 1-38.
43. W3C. (2013). SPARQL 1.1 Update. *W3C Recommendation*, 21 March 2013. https://www.w3.org/TR/sparql11-update/
44. W3C. (2008). SPARQL Query Language for RDF. *W3C Recommendation*, 15 January 2008. https://www.w3.org/TR/rdf-sparql-query/
45. Grandi, F. (2010). T-SPARQL: A TSQL2-like Temporal Query Language for RDF. In *Local Proceedings of the 14th East-European Conference on Advances in Databases and Information Systems (ADBIS'2010)*, Novi Sad, Serbia, 20-24 September 2010. CEUR Workshop Proceedings (CEUR-WS.org), Vol. 639, pp. 21-30.
46. W3C. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. https://www.w3.org/Submission/SWRL/
47. O'Connor, M., and Das, A. (2009). SQWRL: a query language for OWL. In *Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009)*, Chantilly, VA, USA, 23-24 October 2009. CEUR Workshop Proceedings (CEUR-WS.org), Vol. 529, https://ceur-ws.org/Vol-529/owled2009_submission_42.pdf
48. Brahmia, Z., Grandi, F., and Bouaziz, R. (2023). τSQWRL: A TSQL2-Like Query Language for Temporal Ontologies Generated from JSON Big Data. *Big Data Mining and Analytics, 6*(3), 288-300.