BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 01/06 pp271 – 281
Volume 5, Number 4, December 2022
DOI: 10.26599/BDMA.2021.9020019

τJOWL: A Systematic Approach to Build and Evolve a Temporal OWL 2 Ontology Based on Temporal JSON Big Data

Zouhaier Brahmia*, Fabio Grandi, and Rafik Bouaziz

Abstract: Nowadays, ontologies, which are defined under the OWL 2 Web Ontology Language (OWL 2), are being used in several fields like artificial intelligence, knowledge engineering, and Semantic Web environments to access data, answer queries, or infer new knowledge. In particular, ontologies can be used to model the semantics of big data as an enabling factor for the deployment of intelligent analytics. Big data are being widely stored and exchanged in JavaScript Object Notation (JSON) format, in particular by Web applications. However, JSON data collections lack explicit semantics as they are in general schema-less, which does not allow to efficiently leverage the benefits of big data. Furthermore, several applications require bookkeeping of the entire history of big data changes, for which no support is provided by mainstream Big Data management systems, including Not only SQL (NoSQL) database systems. In this paper, we propose an approach, named τJOWL (temporal OWL 2 from temporal JSON), which allows users (i) to automatically build a temporal OWL 2 ontology of data, following the Closed World Assumption (CWA), from temporal JSON-based big data, and (ii) to manage its incremental maintenance accommodating the evolution of these data, in a temporal and multi-schema environment.

Key words: big data; JavaScript Object Notation (JSON); JSON schema; temporal JSON; ontology; temporal ontology; τJSchema; τOWL

1 Introduction

Ontologies[1] have been invented to specify, in a formal way, the knowledge of some domain, or the data semantics in some application. They are very useful in artificial intelligence, knowledge engineering, and Semantic Web environments, to access data, to answer queries, and to infer new knowledge. Several ontology languages have been proposed in the literature like Resource Description Framework (RDF), RDFS, DAML+OIL, SHOE, Web Ontology Language (OWL), OWL2QL, DL-Lite, and OWL 2[2]; OWL 2 is the most popular one since it is the World Wide Web Consortium (W3C) recommendation for specifying ontologies in the Semantic Web. In the ontology world, the traditional viewpoint is the Open World Assumption (OWA)[3] which means that "what is not known to be true or false is unknown"; it is used in contexts where data are assumed to be incomplete like Semantic Web repositories and Knowledge Bases. On the other hand, the Closed World Assumption (CWA)[4], which means that "what is not known to be true must be false" and which is usual in contexts where data are assumed to be complete like classical databases, can also be adopted in the ontology world by means of the Data Box (DBox) notion[5]. Notice that, with the CWA, an ontology definition (concepts, relationships between concepts, axioms $\cdots$) behaves like a database schema and, thus, can be called ontology schema; the ontology individuals

• Zouhaier Brahmia and Rafik Bouaziz are with the Faculty of Economics and Management, University of Sfax, Sfax 3029, Tunisia. E-mail: zouhaier.brahmia@fsegs.rnu.tn; rafik.bouaziz@usf.tn.
• Fabio Grandi is with the Department of Computer Science and Engineering, University of Bologna, Bologna 40136, Italy. E-mail: fabio.grandi@unibo.it.
* To whom correspondence should be addressed.
Manuscript received: 2021-08-27; revised: 2021-10-27; accepted: 2021-11-01

are instances of such an ontology schema.

Big data[6,7], that is data so large, fast-growing, or complex that cannot be processed with traditional methods, are being generated and exploited by several Web applications (e.g., online social networks, Internet of Things, cloud computing applications); they are stored and exchanged in semi-structured and lightweight data formats like JavaScript Object Notation (JSON)[8]. Such data lack explicit semantics as they are in general schema-less[9], from one hand, and usually created without an associated ontology, from the other hand. As a consequence, the absence of semantics does not allow to efficiently leverage the benefits of big data, as the *meaning* of data cannot be exploited, making difficult the application of intelligent analytics methods. In addition, due to the *velocity* characteristic of big data, they are evolving over time at a high speed and since many JSON-based Not only SQL (NoSQL) database applications require bookkeeping of the entire history of big data changes, these data could also be temporal (i.e., their values are time-referenced) and multi-version (i.e., the same data have several successive temporal versions). Therefore, these characteristics make more difficult not only the update, querying, and analytics of temporal and multi-version JSON-based big data, but also the formalization of their semantics.

For such reason, it is useful to have appropriate tools that help constructing temporal ontologies for temporal and multi-version big data. However, by studying the state of the art of ontologies for big data[10–24], we have noticed that there is no proposal for automatically creating a temporal ontology from temporal and multi-version big data. To fill this gap, we propose in this paper an approach, named τJOWL (standing for Temporal OWL 2 from Temporal JSON), which allows: (i) to automatically build a temporal OWL 2 ontology of data, with the CWA, from temporal big data in JSON format, and (ii) to manage the incremental maintenance of this ontology in response to the evolution of the underlying time-varying JSON big data. The rest of this paper is structured as follows. Section 2 proposes our τJOWL approach. Section 3 illustrates the functioning of τJOWL through an application example. Section 4 discusses related work. Section 5 summarizes the paper and gives some remarks about our future work.

2 Overview of Our τJOWL Approach

As mentioned above, the τJOWL approach allows to automatically build and manage a temporal OWL 2 ontology from temporal big data in JSON format. Hence, our approach could be considered as composed of two parts: (i) temporal ontology building based on temporal JSON big data, and (ii) temporal ontology maintenance resulting from the evolution of temporal JSON big data. These parts are detailed below in subsections 2.1 and 2.2, respectively. In subsection 2.3 we briefly deal with the evaluation of the scalability of our approach.

2.1 Temporal ontology building based on temporal JSON big data

The goal of this part is to automatically build an ontology of data with the CWA viewpoint. More precisely, we present a solution to define a temporal OWL 2 ontology from given temporal JSON-based big data, in an automatic manner. When a new big data project has to be started in the τJOWL environment, the "τJOWL Base Administrator", interacting with the "Ontology Building" module as shown in Fig. 1, firstly calls for the building of an ontology for some temporal JSON data that are stored in a temporal JSON document (to be precise, in the first version of a temporal JSON document). To satisfy the request of the "τJOWL Base Administrator", the behaviour of this module is composed of two successive steps: "Schema Extraction" and "Ontology Generation". In the first step, the "Schema Extraction" sub-module produces a JSON Schema[25–27] file that defines the schema/structure of the JSON instance document[8] supplied as input. To implement this step, several available JSON Schema inference/extraction tool can be used. For example, one of the three first tools proposed on The Home of JSON Schema[28] for "Schema Generation from Data" could be adopted, being their source code freely available on GitHub: json-schema-inferrer[29], Schema Guru[30] and Clojure JSON Schema Validator & Generator[31].

In the second step, the JSON Schema file resulting

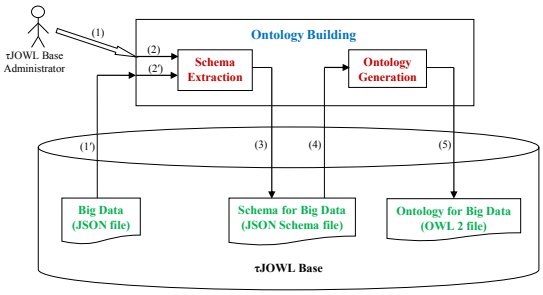

Fig. 1 The first part of the τJOWL approach.

from the previous step is fed to the "Ontology Generation" sub-module, which generates an OWL 2[2] file. The output of the sub-module represents the ontology, specified using the OWL 2 language, of the JSON data initially considered. The algorithm of the "Ontology Generation" module is provided in Fig. 2. This algorithm calls three procedures: GenerateClasses( ), GenerateDataProperties( ), and GenerateObjectProperties( ), which are detailed in Figs. 3–5, respectively.

Notice that, for the sake of simplicity, in this work we only consider the following components of an OWL 2 ontology: classes (i.e., the set of concepts of the modelled reality), data properties (i.e., the set of properties of the concepts/classes), object properties (i.e., the set of the semantic relationships between the concepts/classes), and axioms (i.e., the set of axioms that concern the classes, data properties, object properties, and the set of all identifiers associated to the classes). The complete set of all components of an OWL 2 ontology could be found in Ref. [2].

Notice also that our τJOWL approach deals with OWL 2 ontologies with an RDF/XML (eXtensible Markup Language) syntax[32], which is, according to the OWL 2 specification document[33], the only syntax that must mandatorily be supported by OWL 2 tools.

Recall that, in JSON Schema, a simple type property (mentioned in the algorithm of Fig. 4) is a property with a "string", "number", "integer", "boolean", or "null" type, and a complex type property (mentioned in the algorithm

```
Algorithm Ontology_Generation
// It converts a JSON Schema file to an OWL2 file
Input: JS.json   // A valid JSON Schema file
Output: Ont.owl  // A valid OWL 2 file
Begin
  CreateEmptyOWL2File(Ont.owl);
  // It creates an empty OWL 2 ontology file
  GenerateClasses(JS.json, Ont.owl);
  // It generates initially empty classes
  For Each generated class C Do:
    GenerateDataPropertiesOf(C, Ont.owl);
    // It generates data properties of C
    GenerateObjectPropertiesOf(C, Ont.owl);
    // It generates object properties of C
  End For
End
```

Fig. 2 Algorithm of the "Ontology Generation" module.

```
Procedure GenerateClasses
Inputs: JS.json,  // A valid JSON Schema file
        Ont.owl   // A valid OWL 2 file
Output: CL        // A set of empty classes
                     (only having a name)
Begin
  CL := ∅;
  // To determine all the classes that result
     from JSON Schema objects:
  For Each named JSON Schema object O
             (at any level) in JS.json Do:
    CL := CL ∪ {O};
  End For
  // To determine all the classes that result
     from JSON Schema arrays:
  For Each named JSON Schema array A
             (at any level) in JS.json Do:
    CL := CL ∪ {A};
  End For
  AddClassesToOntology(Ont.owl, CL);
  // It adds all the classes to the ontology
End
```

Fig. 3 Algorithm of the "GenerateClasses" procedure.

```
Procedure GenerateDataProperties
Inputs: C,       // A class named C
        Ont.owl // A valid OWL 2 file
Output: DP       // The set of data properties
                    of the class C
Begin
  DP := ∅;
  If ( the class C results from
       a named JSON Schema object O ) Then
    For Each simple type property P
                          of the object O Do:
      DP := DP ∪ {(name(P),type(P))};
    End For
  Else // The class C results from
          a named JSON Schema array A, in JS.json
    If ( all the elements of the array A
         are of the same data type ) Then
      If (the data type of the elements
          of A is a simple type) Then
        dataPropName := name(A)+"Element";
        dataPropType := type(A[0]);
        DP := DP ∪ {(dataPropName,dataPropType)};
      End If
    Else // The elements of A are not
            of the same data type
      i := 0; // Index for all elements
                 of the array A
      j := 0; // Index for only simple
                 type elements of A
      While ( ¬ endOfArray(A) ) Do:
        If ( A[i] is a simple type element ) Then
          j++;
          dataPropName := name(A)+"Element"+"j";
          dataPropType := type(A[i]);
          DP := DP ∪ {(dataPropName,dataPropType)};
        End If
        i++;
      End While
    End If
  End If
  ClassDataPropertiesToOntology(Ont.owl, C, DP);
  // It adds, to the ontology, all
     the data properties of the class C
End
```

Fig. 4 Algorithm of the "GenerateDataProperties" procedure.

```
Procedure GenerateObjectProperties
Inputs: C,       // A class named C
        Ont.owl  // A valid OWL 2 file
Output: OP       // The set of object properties
         of the class C
Begin
  OP := ∅;
  For Each class R generated from a complex type
           property P of a JSON Schema object O or
           from a complex type element E of a JSON
           Schema array A, which is associated to
           the class C in JS.json Do:
     objPropName := name(C)+"_Has_"+name(R);
     // The name of the new object property
     objPropDomain := C;
     // The domain of the new object property
     objPropRange := R;
     // The range of the new object property
     OP := OP ∪
        {(objPropName,objPropDomain,objPropRange)};
  End For
  AddClassObjectPropertiesToOntology(Ont.owl,C,OP);
  // It adds, to the ontology, all the object
     properties of the class C
End
```

Fig. 5 Algorithm of the "GenerateObjectProperties" procedure.

of Fig. 5) is a property with an "object" or "array" type.

In sum, this first part is a sort of *bulk loading* of the new τJOWL project by creating a first temporal version of an ontology from big data.

Notice that extant JSON data are stored and managed in τJOWL as in our previous temporal JSON τJSchema framework[34, 35]. Moreover, the OWL 2 ontologies are stored and managed in τJOWL as in our previous temporal ontology τOWL framework[36, 37]. Notice that (temporal) reasoning[38–42], which can be part of some AI-based data analytics or data valorization task, can be easily done on temporal OWL 2 ontologies managed in our framework, through the use of suitable tools. This issue is beyond the scope of this paper.

Notice also that the intermediate JSON Schema file resulting from the "Schema Extraction" step is kept (and will be versioned) in τJOWL as in our τJSchema framework.

2.2 Temporal ontology maintenance resulting from the evolution of temporal JSON big data

As big data changes are unavoidable due to their highly dynamic nature, the τJOWL Base Administrator can use the "Big Data Instance Update" module, as shown in Fig. 6, to apply a sequence of JSON update operations, each time some data update is needed, that is to insert new big data or to modify or delete some already

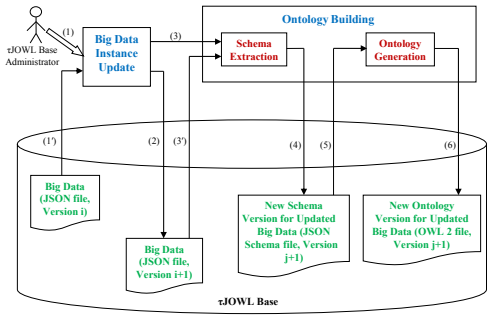

Fig. 6 The second part of the τJOWL approach.

stored big data. A JSON data update could be either conservative[34, 35] or non-conservative[43–45].

- A *conservative update* is an update that produces modified JSON data conformant to their JSON Schema; it is an update that respects the schema of the involved data, like the modification of the value of a string property with another string value. Since τJOWL is a temporal environment, only the updated JSON-based big data are temporally versioned after such an update, through the generation of a new temporal version of them (see Fig. 6), which is still valid with respect to the current JSON Schema version.
- A *non-conservative update* of some JSON data produces a new version of such data that is no longer conformant to their initial JSON Schema; it is an update that does not respect the schema of the involved data, like the renaming of a property, the replacement of the value of a number property with a string value or the addition of the value for a property that does not belong to the schema. Therefore, such an update requires that some schema change(s) must be applied in order to produce a new JSON Schema version that defines the structure of the new JSON data version, before the update can be done. Since τJOWL is a temporal environment and in order to keep all versions of changed JSON-based big data (i.e., from the first version to the last one), not only the updated JSON data but also their JSON Schema, in response to non-conservative updates, come out temporally versioned. In fact, after the generation of the new temporal version of the updated JSON-based big data, the "Big Data Instance Update" module (see Fig. 6) calls the "Ontology Building" module, doing the same work as described in subsection 2.1, but starting from this new big data version. More precisely, the "Ontology Building" module first extracts the schema of this new data version and saves it as a new temporal version of the JSON Schema of the involved JSON-based big data (via the "Schema Extraction" sub-module), and then

generates the OWL 2 file associated to the new JSON Schema version file, and saves it as a new temporal version of the OWL 2 ontology of the same JSON-based big data (via the "Ontology Generation" sub-module).

Hence, the second part of our approach could be considered as a solution for incremental maintenance of a τJOWL big data project, with the creation of new temporal data versions when extant JSON data are updated, and also of new ontology versions when applied updates are non-conservative.

It is worth mentioning that, for the implementation of the "Big Data Instance Update" module, the algorithm named "ApplyUpdatesWithImplicitSchemaVersioning", previously presented in Ref. [43] (Fig. 1), can be used.

2.3 Complexity evaluation

From a theoretical point of view, all the procedures used for ontology bulk loading presented in Section 2.1 require a scan of the JSON data file and, thus, the whole process scales linearly with its dimension. The incremental ontology maintenance described in Section 2.2 requires three steps: (i) non-conservative update detection, (ii) execution of schema changes, and (iii) propagation of changes to data instances. As far as the complexity of these steps is concerned, we can say that: (i) is linear with respect to the size of the new data to be accommodated; (ii) is linear with respect to the ontology size (which is much smaller than the instance data size; it also has a much smaller growth rate as most updates are expected to be conservative); (iii) is expected to be linear with respect to the instance data size, giving rise to a sublinear scaling for the whole maintenance process (it would be linear if all updates were non-conservative).

3 Application Example

In order to illustrate the functioning of our approach, let us consider an example of a JSON-based NoSQL data store used by a scholarly publisher for the management of scientific journals' data. Let us assume that, with the aim of making such a management more intelligent and to improve access and querying of journal data, the Information Technology (IT) manager of the publisher decides to use ontologies of data with the CWA, within a τJOWL environment.

Assume also that on June 01, 2021, the τJOWL Base Administrator created (or imported from an already existing project) a JSON instance document, named "journalsInstances_V1.json" (as shown in Fig. 7),

```
{ "journal":{
     "title": "Big Data and Ontologies",
     "editor": { "firstName":"Abdullah",
                 "lastName":"Farouk" },
     "periodicity": 4,
     "SJR": 0.15 } }
```

Fig. 7 JSON instance document of journals ("JournalsInstances_V1.json") on June 01, 2021.

which stores information on journals (the title, the first name and the last name of the editor, the periodicity, and the SJR ranking of each journal). Due to space limitations, in our example we only consider one journal named "Big Data and Ontologies", whose editor is Abdullah (first name) Farouk (last name), its periodicity is 4 (issues per year), and its SJR is 0.15. Then, he/she invoked the "Ontology Building" module to automatically build an ontology for these JSON-based big data. Indeed, this module uses the provided big data to extract their JSON Schema (by calling the "Schema Extraction" sub-module), named "journalsSchema_V1.json" (as shown in Fig. 8), before generating the OWL 2 file corresponding to the extracted schema (by calling the "Ontology Generation" sub-module), named "journalsOntology_V1.owl" (as shown in Fig. 9).

After that, assume that on June 13, 2021, the τJOWL Base Administrator used the "Big Data Instance Update" module to update the first/current version of the JSON instance document that stores journal details (as shown in Fig. 7), by applying the following changes:

- Add to the "journal" object a new object member, named "scopus-indexed", with type "boolean" and value true; this member provides information on the fact that the corresponding journal is indexed by the Scopus database or not;
- Change the value of the object member "periodicity"

```
{"type":"object",
 "properties":
   {"journal":
      {"type":"object",
       "properties":
         {"title":{"type":"string"},
          "editor":
            {"type":"object",
             "properties":
               {"firstName":{"type":"string"},
                "lastName":{"type":"string"}}},
          "periodicity":{"type":"integer"},
          "SJR":{"type":"number"}}}}}
```

Fig. 8 The generated JSON Schema of journals ("JournalsSchema_V1.json") on June 01, 2021.

```
<rdf:RDF>
  <owl:Ontology rdf:about="http://
       my_ontologies/journal_ontology#">
    <owl:Class rdf:about="journal"/>
    <owl:DatatypeProperty rdf:about="title">
      <rdfs:domain rdf:resource="journal"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="periodicity">
      <rdfs:domain rdf:resource="journal"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="SJR">
      <rdfs:domain rdf:resource="journal"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#float"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:about="editor"/>
    <owl:DatatypeProperty rdf:about="firstName">
      <rdfs:domain rdf:resource="editor"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="lastName">
      <rdfs:domain rdf:resource="editor"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:ObjectProperty
        rdf:about="journal_Has_editor">
      <rdfs:domain rdf:resource="journal"/>
      <rdfs:range rdf:resource="editor"/>
    </owl:ObjectProperty>
  </owl:Ontology>
</rdf:RDF>
```

Fig. 9 The generated OWL 2 ontology of journals ("JournalsOntology_V1.owl") on June 01, 2021.

from the integer 4 to the string "quarterly";

- Rename the object member "SJR" to "currentSJR".

Since τJOWL is temporal and multi-version, the execution of the above sequence of operations on the first JSON instance document version of journals produces a new version named "journalsInstances_V2.json" (as shown in Fig. 10). Moreover, since each one of these operations is a non-conservative update (i.e., it does not respect the JSON Schema version, as shown in Fig. 8, corresponding to the JSON instance document

```
{ "journal":
  { "title": "Big Data and Ontologies",
    "editor": { "firstName": "Abdullah",
                "lastName": "Farouk" },
    "scopus-indexed":  true,
    "periodicity": "quarterly",
    "currentSJR": 0.15 } }
```

Fig. 10 JSON instance document of journals ("JournalsInstances_V2.json") on June 13, 2021.

version being updated), the "Big Data Instance Update" module also calls the "Ontology Building" module that, first, generates a new JSON Schema version, named "journalsSchema_V2.json" (as shown in Fig. 11), for this new JSON instance document version and, second, converts the new JSON Schema version to an OWL 2 file named "journalsOntology_V2.owl" (as shown in Fig. 12), which represents the second OWL 2 ontology version associated to the updated JSON-based big data mentioned above (i.e., those stored in the new JSON instance document version of Fig. 10). Changes are presented in red bold typeface.

4 Related Work Discussion

In the literature, a lot of works have dealt with the use of ontologies for big data management, as summarized in the following.

Djebouri and Keskes[21] have studied (among others) a set of research papers that combine ontologies and big data; these works introduce ontologies to deal with the challenges posed by big data, including various formats, various sources, and huge volumes of data.

To deal with the issue of big data integration, several research works, like Refs. [16, 19, 20, 24], have widely used ontologies not only since they allow to formalize the knowledge of any domain but also since they allow to have a unified view of big data, to extract reliable and consistent knowledge and facilitate reasoning and analytics on large amounts of data.

Some other works, like Refs. [13–15, 46], have used ontologies for (intelligent) big data analysis and answering queries on big data[11, 19].

In addition, there are many research proposals, like Refs. [15, 18, 46–48], which have used ontologies for the management of social big data (i.e., data stored in the

```
{"type":"object",
 "properties":
   {"journal":
      {"type":"object",
       "properties":
         {"title":{"type":"string"},
          "editor":
             {"type":"object",
              "properties":
                 {"firstName":{"type":"string"},
                  "lastName":{"type":"string"}}},
          "scopus-indexed":{"type":"boolean"},
          "periodicity":{"type":"string"},
          "currentSJR":{"type":"number"}}}}}
```

Fig. 11 The generated JSON Schema of journals ("JournalsSchema_V2.json") on June 13, 2021.

```
<rdf:RDF>
  <owl:Ontology rdf:about="http://
       my_ontologies/journal_ontology#">
    <owl:Class rdf:about="journal"/>
    <owl:DatatypeProperty rdf:about="title">
      <rdfs:domain rdf:resource="journal"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty
         rdf:about="scopus-indexed">
      <rdfs:domain rdf:resource="journal"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#boolean"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="periodicity">
      <rdfs:domain rdf:resource="journal"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="currentSJR">
      <rdfs:domain rdf:resource="journal"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#float"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:about="editor"/>
    <owl:DatatypeProperty rdf:about="firstName">
      <rdfs:domain rdf:resource="editor"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="lastName">
      <rdfs:domain rdf:resource="editor"/>
      <rdfs:range rdf:resource="http://
            www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:ObjectProperty
         rdf:about="journal_Has_editor">
      <rdfs:domain rdf:resource="journal"/>
      <rdfs:range rdf:resource="editor"/>
    </owl:ObjectProperty>
  </owl:Ontology>
</rdf:RDF>
```

Fig. 12 The generated OWL 2 ontology of journals ("JournalsOntology_V2.owl") on June 13, 2021.

databases of the online social networks, like Facebook, Google+, YouTube, and LinkedIn).

The works that are more strictly related with our approach are Refs. [16, 23, 48–53].

Abbes and Gargouri[16] and Abbes et al.[49] have proposed an approach to learn an OWL ontology from big data in a MongoDB NoSQL database, by means of the application of eleven transformation rules. The approach is supported by a tool, named M2Onto, which also has been described by the authors. Similarly to this proposal, Mhammedi et al.[23] have provided an approach to generate an OWL ontology from data that are stored in a Couchbase NoSQL database, through the application of six mapping rules; a tool, named Cb2Onto, supporting the approach has been also presented. With regard to Refs. [16, 23, 49], our approach also allows learning an OWL 2 ontology but from a JSON Schema file which can much more complex than the data models supported by the two NoSQL database systems MongoDB and CouchBase. Moreover, our work is generic since it is independent from any NoSQL database system.

Some researchers have worked on automatic generation of an OWL ontology from a JSON instance document, like Yao et al.[50] and Moreira et al.[51]. Some others have proposed to produce an OWL ontology directly from a JSON Schema file, like Wischenbart et al.[48] and Cheong[52]. The idea of Ganzha et al.[53] was to make the conversion of an input JSON Schema file into an output OWL ontology easier but possibly longer, using an XML Schema file as an intermediate result. In fact, first the JSON Schema file is converted into an XML Schema file (by using one of the several available tools), then this latter is transformed into an OWL file. Similarly to Refs. [48,52,53], our approach uses a JSON Schema file in order to build an ontology. However, differently from these proposals, τJOWL starts from a JSON instance file to automatically construct an OWL 2 ontology, and implicitly propagates changes on JSON instance documents to OWL 2 ontology files, within a temporal and multi-schema-version environment.

Notice that the problem of "extracting a schema from JSON data" has been widely studied by the JSON community, for example in Refs. [27, 54–59].

Since the XML[60] format could be also used to store and exchange irregular data on the (Semantic) Web, several works have dealt with transformation of XML documents to OWL ontologies. Hacherouf et al.[61] have presented a survey on such works. Some other researchers have focused on converting to OWL ontologies the schemas of XML documents, specified using either the XML Schema language[62] or the Document Type Definition (DTD) language[60]. In particular, Bedini et al.[63] have proposed a set of patterns for an automatic transformation of XML Schema files into OWL files, while Hacherouf and Bahloul[64] have studied transformation of DTD files into OWL ontologies. However, XML documents are usually complicated and verbose, whereas JSON has been designed as a lightweight alternative to XML, more suitable to big data storage and exchange.

5 Conclusion

In this paper, we have proposed an approach, named τJOWL (Temporal OWL 2 from Temporal JSON), with a twofold purpose, that is allowing users of a big data project: (i) to automatically construct a temporal OWL 2 ontology of data, with the CWA, from temporal JSON-based big data; (ii) to manage as incremental maintenance the evolution of this ontology guided by the evolution of the underlying temporal JSON big data, in a temporal environment that supports versioning at both instance and schema levels.

In the near future, we plan to develop a tool that supports our approach and shows its usefulness. We will use the new tool to experimentally evaluate the effectiveness, usability, and scalability of our approach, and to compare its performance with similar approaches. Its design is currently in progress and is based on the extension and integration of the previously developed prototypes τJSchema-Manager[43, 44] and τOWL-Manager[37, 45, 65]. In order to evaluate the effectiveness of our approach, we will test the accuracy of the results produced by our tool starting from real examples. As for the assessment of the usability of the approach, first we will recruit a panel of users to test our tool (working on some illustrative cases and real examples), and after that we will collect their feedbacks concerning the usability of the tool; such feedbacks will help us to improve the tool and possibly the approach. In order to evaluate the scalability of our approach, we will look for an existing synthetic benchmark to be used in experiments. Otherwise, we will construct such a benchmark. To this purpose, a JSON generator will be exploited to produce random JSON files, according to some chosen characteristics (e.g., number of JSON objects, average number of JSON properties, and levels of nesting of JSON objects and JSON properties); obviously, a custom JSON generator could be written from scratch or some available JSON generator (e.g., Refs. [66, 67]) could be used as it is or adapted to our situation. In this way, the scaling behaviour of the approach sketched in Section 2.3, including its dependence on the fraction of non-conservative updates, will be matched against practical management of real JSON data.

References

[1] N. Guarino, *Formal Ontology in Information Systems*. Amsterdam, The Netherlands: IOS Press, 1998.

[2] W3C, OWL 2 web ontology language primer (second edition), W3C recommendation 11 December 2012, http://www.w3.org/TR/owl2-primer/, 2021.

[3] P. F. Patel-Schneider and I. Horrocks, A comparison of two modelling paradigms in the Semantic Web, *J. Web Semant.*, vol. 5, no. 4, pp. 240–250, 2007.

[4] O. Etzioni, K. Golden, and D. S. Weld, Sound and efficient closed-world reasoning for planning, *Artif. Intell.*, vol. 89, no. 1&2, pp. 113–148, 1997.

[5] I. Seylan, E. Franconi, and J. De Bruijn, Effective query rewriting with ontologies over DBoxes, in *Proc. 21st Int. Joint Conf. on Artificial Intelligence*, Pasadena, CA, USA, 2009, pp. 923–929.

[6] T. R. Rao, P. Mitra, R. Bhatt, and A. Goswami, The big data system, components, tools, and technologies: A survey, *Knowl. Inf. Syst.*, vol. 60, no. 3, pp. 1165–1245, 2019.

[7] A. Davoudian and M. C. Liu, Big data systems: A software engineering perspective, *ACM Comput. Surv.*, vol. 53, no. 5, p. 110, 2020.

[8] IETF, The JavaScript Object Notation (JSON) data interchange format, https://tools.ietf.org/html/rfc8259, 2021.

[9] S. Banerjee, R. Shaw, A. Sarkar, and N. C. Debnath, Towards logical level design of big data, in *Proc. of 2015 IEEE 13th Int. Conf. on Industrial Informatics*, Cambridge, UK, 2015, pp. 1665–1671.

[10] A. Hoppe, C. Nicolle, and A. Roxin, Automatic ontology-based user profile learning from heterogeneous web resources in a big data context, *Proc. VLDB Endow.*, vol. 6, no. 12, pp. 1428–1433, 2013.

[11] A. Soylu, M. Giese, E. Jimenez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks, OptiqueVQS: Towards an ontology-based visual query system for big data, in *Proc. 5th Int. Conf. on Management of Emergent Digital EcoSystems*, Neumünster Abbey, Luxembourg, 2013, pp. 119–126.

[12] C. Jayapandian, C. H. Chen, A. Dabir, S. Lhatoo, G. Q. Zhang, and S. S. Sahoo, Domain ontology as conceptual model for big data management: Application in biomedical informatics, in *Proc. of the 33rd Int. Conf. on Conceptual Modeling*, Atlanta, GA, USA, 2014, pp. 144–157.

[13] T. Shah, F. Rabhi, and P. Ray, Investigating an ontology-based approach for Big Data analysis of inter-dependent medical and oral health conditions, *Cluster Comput.*, vol. 18, no. 1, pp. 351–367, 2015.

[14] J. P. C. Verhoosel and J. Spek, Applying ontologies in the dairy farming domain for big data analysis, in *Proc. 3rd Stream Reasoning* (*SR 2016*) *and the 1st Semantic Web Technologies for the Internet of Things* (*SWIT 2016*) *Workshops Co-located with 15th Int. Semantic Web Conf.* (*ISWC 2016*), Kobe, Japan, 2016, pp. 91–100.

[15] A. R. Kim, H. A. Park, and T. M. Song, Development and evaluation of an obesity ontology for social big data analysis, *Healthc. Inform. Res.*, vol. 23, no. 3, pp. 159–168, 2017.

[16] H. Abbes and F. Gargouri, MongoDB-based modular ontology building for big data integration, *J. Data Semant.*, vol. 7, no. 1, pp. 1–27, 2018.

[17] L. S. Globa, R. L. Novogrudska, and A. V. Koval, Ontology model of telecom operator big data, in *Proc. of 2018 IEEE*

Int. Black Sea Conf. on Communications and Networking, Batumi, GA, USA, 2018, pp. 1–5.

[18] P. Wongthongtham and B. A. Salih, Ontology-based approach for identifying the credibility domain in social Big Data, *J. Organ. Comput. Electron. Commer*, vol. 28, no. 4, pp. 354–377, 2018.

[19] S. Nadal, O. Romero, A. Abelló, P. Vassiliadis, and S. Vansummeren, An integration-oriented ontology to govern evolution in Big Data ecosystems, *Inform. Syst.*, vol. 79, pp. 3–19, 2019.

[20] P. S. Rani, R. M. Suresh, and R. Sethukarasi, Multi-level semantic annotation and unified data integration using semantic web ontology in big data processing, *Cluster Comput.*, vol. 22, no. 5, pp. 10401–10413, 2019.

[21] D. Djebouri and N. Keskes, Exploitation of ontological approaches in Big Data: A State of the Art, in *Proc. 10^{th} Int. Conf. on Information Systems and Technologies*, Lecce, Italy, 2020, p. 45.

[22] M. Y. Aghdam, S. R. K. Tabbakh, S. J. M. Chabok, and M. Kheyrabadi, Ontology generation for flight safety messages in air traffic management, *J. Big Data*, vol. 8, no. 1, p. 61, 2021.

[23] S. Mhammedi, H. El Massari, and N. Gherabi, Cb2Onto: OWL ontology learning approach from couchbase, in *Intelligent Systems in Big Data, Semantic Web and Machine Learning*, N. Gherabi and J. Kacprzyk, eds. Cham, Germany: Springer, 2021, pp. 95–110.

[24] I. Mountasser, B. Ouhbi, F. Hdioud, and B. Frikh, Semantic-based Big Data integration framework using scalable distributed ontology matching strategy, *Distrib. Parallel Dat.*, vol. 39, no. 4, pp. 891–937, 2021.

[25] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoc, Foundations of JSON Schema, in *Proc. 25^{th} Int. Conf. on World Wide Web*, Montreal, Canada, 2016, pp. 263–273.

[26] IETF, JSON Schema: A media type for describing JSON documents, https://json-schema.org/latest/json-schema-core.html, 2021.

[27] L. Attouche, M. A. Baazizi, D. Colazzo, F. Falleni, G. Ghelli, C. Landi, C. Sartiani, and S. Scherzinger, A tool for JSON schema witness generation, in *Proc. 24^{th} Int. Conf. on Extending Database Technology*, Nicosia, Cyprus, 2021, pp. 694–697.

[28] JSON Schema, Implementations of JSON schema. Schema generators from data, https://json-schema.org/implementations.html#from-data, 2021.

[29] Json-Schema-Inferrer, Java library for inferring JSON schema from sample JSONs, https://github.com/saasquatch/json-schema-inferrer, 2021.

[30] Schema Guru, https://github.com/snowplow/schema-guru, 2021.

[31] Clojure JSON schema validator & generator, https://github.com/luposlip/json-schema, 2021.

[32] W3C, RDF/XML syntax specification (revised), W3C recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/, 2021.

[33] W3C, OWL 2 web ontology language document overview (second edition), W3C recommendation 11 December 2012, http://www.w3.org/TR/owl2-overview/, 2021.

[34] S. Brahmia, Z. Brahmia, F. Grandi, and R. Bouaziz, τJSchema: A framework for managing temporal JSON-Based NoSQL databases, in *Proc. of the 27^{th} Int. Conf. on Database and Expert Systems Applications*, Porto, Portugal, 2016, pp. 167–181.

[35] S. Brahmia, Z. Brahmia, F. Grandi, and R. Bouaziz, A disciplined approach to temporal evolution and versioning support in JSON data stores, in *Emerging Technologies and Applications in Data Processing and Management*, Z. M. Ma and L. Yan, eds. Hershey, PA, USA: IGI Global, 2019, pp. 114–133.

[36] A. Zekri, Z. Brahmia, F. Grandi, and R. Bouaziz, τOWL: A framework for managing temporal semantic web documents, in *Proc. of the 8^{th} Int. Conf. on Advances in Semantic Processing*, Rome, Italy, 2014, pp. 33–41.

[37] A. Zekri, Z. Brahmia, F. Grandi, and R. Bouaziz, τOWL: A systematic approach to temporal versioning of semantic web ontologies, *J. Data Semant.*, vol. 5, no. 3, pp. 141–163, 2016.

[38] M. J. O'Connor and A. K. Das, A lightweight model for representing and reasoning with temporal information in biomedical ontologies, in *Proc. 3^{rd} Int. Conf. on Health Informatics*, Valencia, Spain, 2010, pp. 90–97.

[39] V. Milea, F. Frasincar, and U. Kaymak, tOWL: A temporal web ontology language, *IEEE Trans. Syst. Man Cybern. B Cybern.*, vol. 42, no. 1, pp. 268–281, 2012.

[40] E. Anagnostopoulos, S. Batsakis, and E. G. M. Petrakis, CHRONOS: A reasoning engine for qualitative temporal information in OWL, in *Proc. of the 17^{th} Int. Conf. in Knowledge Based and Intelligent Information and Engineering Systems*, Kitakyushu, Japan, 2013, pp. 70–77.

[41] S. Batsakis, E. G. M. Petrakis, I. Tachmazidis, and G. Antoniou, Temporal representation and reasoning in OWL 2, *Semant. Web*, vol. 8, no. 6, pp. 981–1000, 2017.

[42] F. Ghorbel, F. Hamdi, E. Métais, N. Ellouze, and F. Gargouri, Ontology-based representation and reasoning about precise and imprecise temporal data: A fuzzy-based view, *Data Knowl. Eng.*, vol. 124, p. 101719, 2019.

[43] Z. Brahmia, S. Brahmia, F. Grandi, and R. Bouaziz, Implicit JSON schema versioning driven by big data evolution in the τJSchema framework, in *Proc. of Int. Conf. on Big Data and Networks Technologies*, Leuven, Belgium, 2019, pp. 23–35.

[44] Z. Brahmia, S. Brahmia, F. Grandi, and R. Bouaziz, Implicit JSON schema versioning triggered by temporal updates to JSON-based Big Data in the τJSchema framework, in *Proc. 5^{th} Int. Conf. on Big Data and Internet of Things*, Rabat, Morocco, doi:10.1007/978-3-030-23672-4_3.

[45] Z. Brahmia, F. Grandi, A. Zekri, and R. Bouaziz, Ontology versioning driven by instance evolution in the τOWL framework, *J. Inf. Knowl. Manag.*, doi: 10.1142/S0219649222500022.

[46] Y. Han, H. Kim, J. Song, and T. M. Song, Ontology development of school bullying for social big data collection and analysis, *J. Korea Contents Assoc.*, vol. 19, no. 6, pp. 10–23, 2019.

[47] M. Wischenbart, S. Mitsch, E. Kapsammer, A. Kusel, B. Pröll, W. Retschitzegger, W. Schwinger, J. Schönböck, M.Wimmer, and S. Lechner, User profile integration made easy: Model-driven extraction and transformation of social

network schemas, in *Proc. 21st Int. Conf. on World Wide Web*, Lyon, France, 2012, pp. 939–948.

[48] M. Wischenbart, S. Mitsch, E. Kapsammer, A. Kusel, S. Lechner, B. Pröll, W. Retschitzegger, J. Schönböck, W. Schwinger, and M. Wimmer, Automatic data transformation-breaching the walled gardens of social network platforms, in *Proc. of the 9th Asia-Pacific Conf. on Conceptual Modelling*, Adelaide, Australia, 2013, pp. 89–98.

[49] H. Abbes, S. Boukettaya, and F. Gargouri, Learning ontology from Big Data through MongoDB database, in *Proc. of the 2015 IEEE/ACS 12th Int. Conf. of Computer Systems and Applications*, Marrakech, Morocco, 2015, pp. 1–7.

[50] Y. G. Yao, R. P. Wu, and H. Liu, JTOWL: A JSON to OWL Converto, in *Proc. 5th Int. Workshop on Web-scale Knowledge Representation Retrieval & Reasoning*, Shanghai, China, 2014, pp. 13–14.

[51] G. B. Moreira, V. M. Calegario, J. C. Duarte, and A. F. P. dos Santos, Extending the VERIS framework to an incident handling ontology, in *Proc. of 2018 IEEE/WIC/ACM Int. Conf. on Web Intelligence*, Santiago, Chile, 2018, pp. 440–445.

[52] H. Cheong, Translating JSON Schema logics into OWL axioms for unified data validation on a digital manufacturing platform, *Procedia Manuf.*, vol. 28, pp. 183–188, 2019.

[53] M. Ganzha, M. Paprzycki, W. Pawlowski, P. Szmeja, K. Wasielewska, and C. E. Palau, From implicit semantics towards ontologies—practical considerations from the INTER-IoT perspective, in *Proc. of the 14th IEEE Annual Consumer Communications & Networking Conf.*, Las Vegas, NV, USA, 2017, pp. 59–64.

[54] J. L. Cánovas Izquierdo and J. Cabot, Discovering implicit schemas in JSON data, in *Proc. of the 13th Int. Conf. on Web Engineering*, Aalborg, Denmark, 2013, pp. 68–83.

[55] M. Klettke, U. Störl, and S. Scherzinger, Schema extraction and structural outlier detection for JSON-based NoSQL Data stores, in *Proc. of the Conf. Database Systems for Business, Technology and Web*, Hamburg, Germany, 2015, pp. 425–444.

[56] D. S. Ruiz, S. F. Morales, and J. G. Molina, Inferring versioned schemas from NoSQL databases and its applications, in *Proc. of the 34th Int. Conf. on Conceptual Modeling*, Stockholm, Sweden, 2015, pp. 467–480.

[57] L. J. Wang, S. Zhang, J. W. Shi, L. M. Jiao, O. Hassanzadeh, J. Zou, and C. Wang, Schema management for document stores, *Proc. VLDB Endow.*, vol. 8, no. 9, pp. 922–933, 2015.

[58] M. A. Baazizi, H. B. Lahmar, D. Colazzo, G. Ghelli, and C. Sartiani, Schema inference for massive JSON datasets, in *Proc. 20th Int. Conf. on Extending Database Technology*, Venice, Italy, 2017, pp. 222–233.

[59] I. Comyn-Wattiau and J. Akoka, Model driven reverse engineering of NoSQL property graph databases: The case of Neo4j, in *Proc. of 2017 IEEE Int. Conf. on Big Data*, Boston, MA, USA, 2017, pp. 453–458.

[60] W3C, Extensible markup language (XML) 1.0 (fifth edition) W3C recommendation 26 November 2008, http://www.w3.org/TR/2008/REC-xml-20081126/, 2021.

[61] M. Hacherouf, S. N. Bahloul, and C. Cruz, Transforming XML documents to OWL ontologies: A survey, *J. Inf. Sci.*, vol. 41, no. 2, pp. 242–259, 2015.

[62] W3C, XML schema part 0: Primer second edition W3C recommendation 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/, 2021.

[63] I. Bedini, C. Matheus, P. F. Patel-Schneider, A. Boran, and B. Nguyen, Transforming XML schema to OWL using patterns, in *Proc. of the 2011 IEEE 5th Int. Conf. on Semantic Computing*, Palo Alto, CA, USA, 2011, pp. 102–109.

[64] M. Hacherouf and S. N. Bahloul, DTD2OWL^2: A new approach for the transformation of the DTD to OWL, *Procedia Comput. Sci.*, vol. 62, pp. 457–466, 2015.

[65] A. Zekri, Z. Brahmia, F. Grandi, and R. Bouaziz, τOWL-Manager: A tool for managing temporal semantic web documents in the τOWL framework, in *Proc. of the 9th Int. Conf. on Advances in Semantic Processing*, Nice, France, 2015, pp. 56–64.

[66] S. Jahangiri, Wisconsin benchmark data generator: To JSON and beyond, in *Proc. 2021 Int. Conf. on Management of Data*, Virtual Event, China, 2021, pp. 2887–2889.

[67] R. Betík and I. Holubová, JBD generator: Towards semi-structured JSON big data, in *Proc. of ADBIS 2016 Short Papers and Workshops*, Prague, Czech Republic, 2016, pp. 54–62.

Rafik Bouaziz received the PhD degree in computer science from the University of Tunis El Manar, Tunis, Tunisia in 1991, and the Habilitation in computer science from the University of Sfax, Sfax, Tunisia in 2007; he was the director of the Economy, Management, and Computer Science Doctoral School in the University of Sfax between 2011 and 2014, and the president of the same university between 2014 and 2017; he is currently a full professor of computer science at the Faculty of Economics and Management, University of Sfax; his research interests include temporal databases, real-time databases, information systems engineering, ontologies, and data warehousing and workflows.

Zouhaier Brahmia received the BSc, MSc, and PhD degrees in computer science from the University of Sfax, Sfax, Tunisia in 2003, 2005, and 2011, respectively; he is currently an associate professor of computer science in the Department of Computer Science at the Faculty of Economics and Management, University of Sfax; his research interests include temporal databases, database schema versioning, and temporal, evolution and versioning aspects in emerging (XML, NoSQL, etc.) databases, big data, semantic Web ontologies, knowledge representation, IoT data management, and blockchains.

Fabio Grandi received the Laurea degree cum Laude in electronics engineering from the University of Bologna, Italy in 1988, and the PhD degree in electronics engineering and computer science from the University of Bologna, Italy in 1994; from 1989 to 2012 he had worked at the CSITE center of the Italian National Research Council (CNR) in Bologna in the field of neural networks and temporal databases, initially supported by a CNR fellowship, in 1993 and 1994 he was an adjunct professor at the Universities of Ferrara, Italy, in 1994 he was appointed as research associate at the University of Bologna, since 1998 he has been an associate professor at the University of Bologna, currently in the Department of Computer Science and Engineering. He has published more than 100 papers in scholarly journals and conference proceedings, is a member of the TSQL2 Language Design Committee and the co-author of the "*The TSQL2 Temporal Query Language*" book; his scientific interests include temporal, evolution and versioning aspects in data management, WWW and Semantic Web, knowledge representation, storage structures, and access cost models.