International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 17 (2025) pp. 628-645

https://doi.org/10.70917/ijcisim-2025-0039
Article

Batch Mode Ontology Instance Updates with Automated Inconsistency Management in the τJOWL Ecosystem

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

[1] MIRACL Laboratory, University of Sfax, Road of the Aerodrome, Km 4.5, P.O. Box 1088, 3018 Sfax, Tunisia; safa.brahmia@gmail.com (S.B.); raf.bouaziz@fsegs.rnu.tn (R.B.)
2 Dipartimento di Informatica – Scienza e Ingegneria, Alma Mater Studiorum – Università di Bologna, Viale Risorgimento 2, I-40136 Bologna, Italy; fabio.grandi@unibo.it
* Corresponding author: zouhaier.brahmia@fsegs.rnu.tn

Received date: 21 February 2025; Accepted date: 25 August 2025; Published online: 31 December 2025

Abstract: τJOWL (Temporal OWL2 from Temporal JSON) is an ecosystem designed to automatically create a time-referenced OWL2 ontology from a JSON data file containing time-referenced Big Data. This ontology acts as a "schema" to manage time-referenced JSON Big Data instances. In our previous research, we explored the effects of time-referenced JSON data updates, which are interactively and incrementally executed by the τJOWL database administrator (DBA), at both instance and schema levels. In this work, we enrich our proposal by addressing time-referenced JSON data updates executed in batch mode. This situation arises when the τJOWL DBA wants to incorporate a new time-referenced JSON file into the τJOWL DB, such as one prepared offline or imported from an external source. Such a data file needs to be integrated with the current ontology instance to create a new version. Additionally, certain instance updates may render them inconsistent with their ontology schema. Therefore, our proposed solution is to automatically create and apply changes to the ontology schema to maintain global consistency. Our new method for handling ontology instance updates within the τJOWL ecosystem, enhances flexibility in knowledge management and ontology evolution. It also supports time-referenced versioning of both big data instances and ontology schema, ensuring the τJOWL DB remains consistent in a transparent way.

Keywords: ontology; OWL2; τJOWL; big data; JSON, JSON schema; batch mode; instance update; JSON schema change; ontology schema change; temporal databases; schema versioning; ontology versioning; consistency; inconsistent Instance

1. Introduction

Context of work. Today, big data [1] are extensively encountered in a lot of emerging applications like internet of medical things, e-marketing, e-learning, intelligent search engines, and mobile audio-visual communications. JSON [2] is the most used lightweight format for the storage and exchange of these big data and their associated metadata. Applications which also require the explicit specification of the structure of JSON data, mainly use JSON schema files (i.e., files containing JSON data definitions written in the IETF JSON Schema language [3]) for the purpose.

Furthermore, big data have been proposed to be used together with ontologies, like in [4], [5], [6], and [7]. This combination is aimed at providing a formal semantics to big data [8], which can be exploited to facilitate high-level processing including big data analytics, integration, governance, and reasoning tasks.

The Semantic Web is based on adding semantics to resources that are accessible on the Web, where ontologies play a major role in the making of this semantic enrichment. The ontology language, which is recommended by the W3C to represent a Semantic Web ontology, is the OWL2 Web Ontology Language

[9], more informally known with the OWL2 shorthand.

Problems. In [10], we have proposed τJOWL (Temporal OWL2 from Temporal JSON), an integrated ecosystem for automatically building a time-referenced OWL2 ontology based on time-referenced JSON-based big data instances. In such an ecosystem, the τJOWL database administrator (DBA) creates a new time-referenced OWL2 ontology by just supplying a time-referenced JSON data file. In fact, the system automatically, and transparently to the τJOWL DBA, first creates a JSON schema file from JSON data, and then derives the OWL2 ontology schema file corresponding to this JSON schema file. After that, in[11], we have extended the τJOWL functionalities with the evolution of a time-referenced OWL2 ontology driven by instance updates. Each time the τJOWL DBA executes a collection of consecutive JSON data update statements on the current JSON data file that stores the current big data instances (or ontology instances), the system (i) checks these update statements, (ii) creates a new time-referenced version of the updated data file, and (iii) also propagates the effects of such instance updates to the ontology schema level through the intermediary JSON schema level, whenever schema changes are needed to be applied before the execution of the instance updates, in order to safeguard the consistency of the final result. For example, when changing, in a JSON data file, a JSON object member name from "city" to "address", a rename statement in the OWL2 ontology schema is required to be executed through the renaming of the "city" component also in the associated JSON schema file. It is worth mentioning that τJOWL is an integrated approach. In fact, everything is "local", and we do not have ontologies remotely managed e.g., by some authority/committee like DC, FOAF, SIOC etc., to which users link their local instance data.

Scientific contributions. In the previous works mentioned above [10,11], the τJOWL DBA interactively and incrementally updates the instances, with the step-by-step execution of update statements using a command prompt. In this paper, we consider a different approach, where updates are submitted in batch mode, via the addition by the τJOWL DBA of a new portion of a time-referenced JSON data file to an existing τJOWL ontology. The new data file could have been prepared offline or could be imported from an external source and must be merged with the current ontology instance to create a new ontology instance version.

In some cases, plain application of batch mode ontology instance updates makes them inconsistent with respect to their ontology schema. Since data consistency is indeed very important in an ontology-based data management framework, countermeasures must be established. To this end, in this article, we also consider the automatic generation and application of ontology schema changes aimed at re-establishing data consistency in a τJOWL ontology.

Our present approach for ontology instance updates in the τJOWL ecosystem provides more flexibility in knowledge management and ontology evolution, and supports time-referenced versioning of both big data instances and ontology schema, transparently to the τJOWL DBA, while preserving the τJOWL DB consistency. In sum, our contributions are twofold:

- an approach for batch mode ontology instance updates;
- an approach for preserving the consistency of time-referenced ontology instances with their ontology schema, threatened by the execution of certain instance updates.

Organization. The organization of the rest of this article is as follows. Our proposal for batch mode ontology instance updates is introduced in Section 2. Section 3 describes our approach for maintaining consistency of time-referenced ontology instances under instance updates that have an impact on such a consistency. The functioning of the approach is exemplified in Section 4. Section 5 gives some information on the implementation of our current approach. In Section 6, the contributions of our present work are discussed in comparison with the related research work. Section 7 presents the limitations of our approach and its possible extensions outside the JSON/OWL2 context. Section 8 concludes the article and briefly describes some possible future extensions of our current proposal.

2. Management of Batch Mode Ontology Instance Updates

In this section, we deal with time-referenced JSON data updates that are run in a batch mode (i.e., non interactively and non incrementally). In fact, this scenario happens when the τJOWL DBA wants to integrate, into the τJOWL database, a new time-referenced JSON data file, which has been prepared offline or which is imported from an external source. The new data file must be merged with the current ontology instance to create a new ontology instance version. Hence, this scenario deals with an ontology instance update that is specified through the creation of a whole ontology instance version, instead of the stepwise execution of a collection of consecutive data update statements on the current version of the ontology instance, each one producing a modified ontology instance version. Obviously, such an addition requires changes at ontology schema level when the final results of the application of instance updates is not conformant to the current version of the ontology schema; consequently, and transparently to the

τJOWL DBA, a collection of necessary consecutive ontology schema change statements is automatically created and applied to the current ontology schema in order to create a new ontology schema version in which the modified ontology instance fits. It is worth mentioning here that only changes of the time-referenced JSON data file, which does not respect the current ontology schema version, would create a new version of the ontology. More precisely, when the new ontology instance version, which is stored in the new JSON data file, is not conformant to the current ontology schema version, a new ontology schema version is automatically generated.

Our overall approach is formalized via the algorithm of Figure 1.

```
ALGORITHM Integrate_New_JSON_Document_Into_τJOWL_DB
 Inputs: JDF, curJDF, curJSch, curOntSch, TempJD, curSquashJD, TempJSch, TempOntSch
 Outputs: JDU, JSC, newJSch, OSC, newOntSch, newSquashJD
BEGIN
 If IsValid(JDF) Then
  If AreDifferent(JDF, curJDF) Then
   If IsConformantToJSONSchema(JDF, curJSch) Then
    UpdateTemporalJSONDocument(TempJD, JDF, curJSch);
    UpdateSquashedJSONDocument(curSquashJD, JDF, curJSch);
    Display("The new JSON data file has been successfully integrated into the τJOWL DB; only
             JSON/ontology instances have been temporally versioned!");
   Else
    DiscoverJSONInstanceUpdates(curJDF, JDF, JDU);
    InferJSONSchemaChanges(curJDF, curJSch, JDF, JDU, JSC);
    CopyJSONSchema(curJSch, newJSch);
    ExecuteJSONSchemaChanges(JSC, newJSch);
    For Each JSON document D that is conformant/linked to curJSch Do
     CopyJSONDocument(D, newD);
     PropagateSchemaChangesToJSONInstances(newJSch, JSC, newD);
    End For
    DeriveOntologySchemaChanges(JSC, curJSch, curOntSch, OSC);
    CopyOntologySchema(curOntSch, newOntSch);
    ExecuteOntologySchemaChanges(OSC, newOntSch);
    UpdateTemporalJSONDocument(TempJD, JDF, newJSch);
    CreateNewSquashedJSONDocument(newSquashJD, JDF, newJSch);
    UpdateTemporalJSONSchema(TempJSch, newJSch);
    UpdateTemporalOntologySchema(TempOntSch, newOntSch);
    Display("The new JSON data file has been successfully integrated into the τJOWL DB; both
             JSON/ontology instances and JSON/ontology schema have been temporally
             versioned!");
   End If
  Else
   Display("The new JSON data file is equal to the current JSON data file version!");
  End If
 Else
  Display("The new JSON data file is not a valid JSON file!");
 End If
END
```

Figure 1. The algorithm for integrating a new JSON data file version in the τJOWL DB, including management of ontology schema changes deriving from such an integration.

In the following, we provide details on the set of input and output variables used by this algorithm, as well as on the set of functions and procedures, which are called by it.

- Inputs:

- JDF: the new JSON data file to be integrated;

- curJDF: the current JSON data file version;
- curJSch: the current JSON schema version;
- curOntSch: the current OWL2 ontology schema version;
- TempJD: the time-referenced JSON data file;
- curSquashJD: the current squashed JSON data file;
- TempJSch: the time-referenced JSON schema file;
- TempOntSch: the time-referenced OWL2 ontology schema file.

- Outputs:

- JDU: the collection of consecutive JSON data update statements;
- JSC: the collection of consecutive JSON schema change statements;
- newJSch: the new JSON schema version;
- OSC: the collection of consecutive OWL2 ontology schema change statements;
- newOntSch: the new OWL2 ontology schema version;
- newSquashJD: the new squashed JSON data file.

- Functions:

- IsValid(JDF): it returns true if JDF is a valid JSON data file, false otherwise;
- AreDifferent(JDF, curJDF): it returns true if the two JSON data files JDF and curJDF are different (i.e., they do not have the same contents), false otherwise;
- IsConformantToJSONSchema(JDF, curJSch): it returns true if the JSON data file JDF is conformant to the JSON schema curJSch, false otherwise.

- Procedures:

- UpdateTemporalJSONDataFile(TempJD, JDF, curJSch): it updates the time-referenced JSON data file TempJD by adding as a new slice the (non-time-referenced) JSON data file JDF, which is conformant to the JSON schema curJSch;
- UpdateSquashedJSONDataFile(curSquashJD, JDF, curJSch): it updates the time-referenced squashed JSON data file curSquashJD by adding as a new slice the (non-time-referenced) JSON data file JDF, which is conformant to the JSON schema curJSch;
- Display(string): it displays on the screen the string passed as argument;
- DiscoverJSONInstanceUpdates(curJDF, JDF, JDU): it discovers the collection of consecutive JSON data update statements (JDU) that could be applied on the current JSON data file curJDF to create the new JSON data file JDF;
- InferJSONSchemaChanges(curJDF, curJSch, JDF, JDU, JSC): it infers the collection of consecutive JSON schema changes that are necessary to be applied on the JSON data file curJDF, which is conformant to the JSON schema version curJSch, to obtain the new JSON data file JDF, which is conformant to the new JSON schema version newJSch;
- CopyJSONSchema(curJSch, newJSch): it creates a new empty JSON schema file newJSch and copies the contents of an existing JSON schema file curJSch into this new file (newJSch);
- ExecuteJSONSchemaChanges(JSC, newJSch): it executes the collection of consecutive JSON schema change statements JSC on the JSON schema file newJSch;
- CopyJSONDataFile(D, newD): it creates a new empty JSON data file newD and copies the contents of an existing JSON data file D into this new data file (newD);
- PropagateSchemaChangesToJSONInstances(newJSch, JSC, newD): it propagates the effects of the collection of consecutive JSON schema change statements JSC to the JSON data file newD in order to make it conformant to the JSON schema newJSch.
- DeriveOntologySchemaChanges(JSC, curJSch, curOntSch, OSC): it creates the collection of consecutive ontology schema change statements OSC, which should be applied on the ontology schema curOntSch, from the collection of consecutive JSON schema change statements JSC, which have been applied on the JSON schema file curJSch that is equivalent to curOntSch;
- CopyOntologySchema(curOntSch, newOntSch): it creates a new empty ontology schema file newOntSch and copies the contents of an existing ontology schema file curOntSch into this new file (newOntSch);
- ExecuteOntologySchemaChanges(OSC, newOntSch): it executes the collection of consecutive ontology schema change statements OSC on the OWL2 ontology schema file newOntSch;
- CreateNewSquashedJSONDataFile(newSquashJD, JDF, newJSch): it creates a new time-referenced squashed JSON data file newSquashJD that is based on the new JSON data file version JDF, added as a first slice of it, which is conformant to the new JSON schema version newJSch;
- UpdateTemporalJSONSchema(TempJSch, newJSch): it updates the time-referenced JSON

schema TempJSch by adding as a new slice the new JSON schema version newJSch;
- UpdateTemporalOntologySchema(TempOntSch, newOntSch): it updates the time-referenced ontology schema TempOntSch by adding as a new slice the new ontology schema version newOntSch.

In this work, we make use of the three following definitions:
- A time-referenced JSON compound data file is a collection of JSON data slices with consecutive transaction-time timestamps.
- A time-referenced JSON compound schema file is a collection of JSON schema slices with consecutive transaction-time timestamps.
- A time-referenced OWL2 ontology compound schema file is a collection of OWL2 ontology schema slices with consecutive transaction-time timestamps.

Notice that any version of the above mentioned files (i.e., a JSON data file, a JSON schema file, or an OWL2 ontology schema file) could itself include time-referenced components.

3. Management of Inconsistent Instances Resulting from Instance Updates

In the temporal ontology world, ontology instances are time-varying to incorporate the evolution of the real world. In some cases, updates to ontology instances make them inconsistent, that is non conformant with respect to their ontology schema.

Moreover, since data consistency is an important requirement in any ontology-based environment, if an instance update (either retroactive, proactive, or on-time) changes a (past, present or future) ontology instance version such that it is no longer consistent with its current ontology schema version, then necessary ontology schema changes must be automatically created and applied to maintain data consistency. It should be noted here that only the non-conservative updates of the time-varying JSON data file, which does not respect the current ontology schema version, would create a new version of the ontology. In other words, when the new ontology instance version, which is stored in the new temporal JSON data file, is not conformant to the current ontology schema version, a new ontology schema version is generated in an automatic manner. Notice that multiple ontology instance versions could be affected by an ontology instance update, and therefore all the involved versions must be processed to maintain the global consistency of the τJOWL DB.

In this section, we propose an approach that manages updates to time-referenced ontology instances and allows data consistency to be maintained when instance updates would make them inconsistent. The approach is formalized via the algorithm of Figure 2, for which we assume that the user applies a collection of consecutive JSON data updates (JDU) to the current JSON data file version (curJDF).

This algorithm uses a set of input and output variables, and calls a set of functions and a set of procedures. In the following, we present only those that have not been mentioned in the algorithm of Figure 1.

- Functions:
  - Empty(JSC): it returns either true if the collection of consecutive JSON schema changes JSC is empty (i.e., it does not include any statement), or false otherwise.
- Procedures:
  - ExecuteJSONInstanceUpdates(JDU, newJDF, JSC): it executes the collection of consecutive JSON data updates JDU on the JSON data file newJDF and creates the collection of consecutive JSON schema change statements (JSC) that must be executed;
  - DeleteJSONDataFile(newJDF): it physically removes the JSON data file newJDF from the τJOWL DB.

4. Illustrative Example

To illustrate the functioning of our approach, let us consider the example of a JSON NoSQL DB exploited by a scientific research centre for the management of scientists' data. Let us assume that, in order to make such a management more "intelligent" with regard to query, update, mining and analysis of data concerning scientists, the data engineer of the centre deploys an ontology of data (more specifically, via an OWL2 ontology schema file), in the τJOWL ecosystem.

ALGORITHM Managing_Inconsistent_Instances_Resulting_From_Updates
Inputs: curJDF, JDU, curJSch, TempJD, curSquashJD, TempJSch, TempOntSch
Outputs: newJDF, newJSch, JSC, newSquashJD, OSC, newOntSch
BEGIN

```
CopyJSONDataFile(curJDF, newJDF);
ExecuteJSONInstanceUpdates(JDU, newJDF, JSC);
If !AreDifferent(newJDF, curJDF) Then
  DeleteJSONDocument(newJDF);
  Display("No changes have been actually applied to curJDF");
Else
  If Empty(JSC) Then
    UpdateTemporalJSONDataFile(TempJD, newJDF, curJSch);
    UpdateSquashedJSONDataFile(curSquashJD, newJDF, curJSch);
  Else
    CopyJSONSchema(curJSch, newJSch);
    ExecuteJSONSchemaChanges(JSC, newJSch);
    PropagateSchemaChangesToJSONInstances(newJSch, JSC, newJDF);
    For Each JSON data file D that is conformant/linked to curJSch Do
      CopyJSONDataFile(D, newD);
      PropagateSchemaChangesToJSONInstances(newJSch, JSC, newD);
    End For
    DeriveOntologySchemaChanges(JSC, curJSch, curOntSch, OSC);
    CopyOntologySchema(curOntSch, newOntSch);
    ExecuteOntologySchemaChanges(OSC, newOntSch);
    UpdateTemporalJSONDataFile(TempJD, newJDF, newJSch);
    CreateNewSquashedJSONDataFile(newSquashJD, newJDF, newJSch);
    UpdateTemporalJSONSchema(TempJSch, newJSch);
    UpdateTemporalOntologySchema(TempOntSch, newOntSch);
  End If
End If
END
```

Figure 2. The algorithm for updating a JSON data file version in the τJOWL ecosystem, dealing with inconsistent ontology instances that may result from such an update.

Assume that, on May 01, 2024, the τJOWL DBA creates (or imports) the JSON data file "ScientistsInstances_V1.json" exposed in Figure 3, which stores data on scientists: the researcher identifier (ResearcherID), the name, the specialty, characterized by an id and a label, and the h-index. Due to space limitations, this example presents only one scientist named "Layla Ahmad", her ResearcherID is "4444-5555-6666-7777", her specialty is "Data Science" (label) with an id equal to 5, and her h-index is 8. After that, the τJOWL DBA activates the "Ontology Building" module of τJOWL to automatically construct an OWL2 ontology for such JSON data. Indeed, this module starts by extracting the JSON schema file "ScientistsJSONSchema_V1.json" exposed in Figure 4 from the provided JSON data file of Figure 3, by means of the "Schema Extraction" sub-module, and then creates the OWL2 ontology file "ScientistsOntologySchema_V1.owl" exposed in Figure 5, corresponding to the extracted JSON schema, by means of the "Ontology Generation" sub-module.

```
{ "scientist":{
   "ResearcherID": "4444-5555-6666-7777",
   "name": "Layla Ahmad",
   "specialty": { "id": 5,
                  "label": "Data Science"},
   "h-index": 8 } }
```

Figure 3. The JSON data file of scientists ("ScientistsInstances_V1.json") on May 01, 2024.

```
{ "type": "object",
 "properties":
   { "scientist":
      { "type": "object",
        "properties":
```

```
        { "ResearcherID": { "type": "string" },
          "name": { "type": "string" },
          "specialty":
            { "type": "object",
              "properties":
                { "id": { "type": "integer" },
                  "label": { "type": "string" } } },
          "h-index": { "type": "integer" } } } } }
```

Figure 4. The produced JSON schema file of scientists ("ScientistsJSONSchema_V1.json") on May 01, 2024.

```
<rdf:RDF>
  <owl:Ontology rdf:about="http://my_onto/scientist_ontology#">
    <owl:Class rdf:about="scientist"/>
    <owl:DatatypeProperty rdf:about="ResearcherID">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="name">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:about="specialty"/>
    <owl:DatatypeProperty rdf:about="id">
      <rdfs:domain rdf:resource="specialty"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="label">
      <rdfs:domain rdf:resource="specialty"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="scientist_Has_specialty">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="specialty"/>
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="h-index">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    </owl:DatatypeProperty>
  </owl:Ontology>
</rdf:RDF>
```

Figure 5. The produced OWL2 ontology schema file of scientists ("ScientistsOntologySchema_V1.owl") on May 01, 2024.

Furthermore, assume that on June 01, 2024, the τJOWL DBA integrates, in the τJOWL DB, a new JSON data file version, as exposed in Figure 6, in order to upgrade the current JSON data file version already exposed in Figure 3.

```
{ "scientist":{
   "ResearcherID": "4444-5555-6666-7777",
   "firstLastName": "Layla Ahmad Hamid",
   "specialty": "Data Science",
   "WoS_H-index": 8,
   "citations": 280 } }
```

Figure 6. The JSON data file of scientists ("ScientistsInstances_V2.json") on June 01, 2024.

Hence, the system takes this new data file version as input and applies the algorithm of Figure 1, which creates and executes, in a transparent manner to the τJOWL DBA, a collection of consecutive JSON schema change statements that are necessary to consistently integrate the new JSON data file version. In fact, this new JSON data file version is not conformant to the current JSON schema version ScientistsJSONSchema_V1.json exposed in Figure 4. In order to preserve the consistency of the τJOWL DB, a new JSON schema version and consequently a new ontology schema version must be created for the new data file version. More precisely, after the collection of consecutive schema change statements has been created, the system automatically applies it to the first JSON schema version "ScientistsJSONSchema_V1.json" exposed in Figure 4 to obtain the new JSON schema version "ScientistsJSONSchema_V2.json" exposed in Figure 7, for the new JSON data file version exposed in Figure 6. Then, within the same update transaction, the system also derives, from the created collection of consecutive JSON schema change statements, an equivalent collection of consecutive OWL2 ontology schema change statements, and applies it on the first OWL2 ontology schema version "ScientistsOntologySchema_V1.owl" exposed in Figure 5 to obtain the new OWL2 ontology schema version "ScientistsOntologySchema_V2.owl" exposed in Figure 8. Notice that "ScientistsOntologySchema_V2.owl" is the OWL2 ontology schema version corresponding to the updated JSON Big Data instances stored in "ScientistsInstances_V2.json" of Figure 6. In Figures 6, 7, and 8, changes with respect to the previous corresponding version are evidenced in red bold typeface.

```
{ "type": "object",
 "properties":
  { "scientist":
     { "type": "object",
      "properties":
       { "ResearcherID": { "type": "string" },
         "firstLastName": { "type": "string" },
         "specialty": { "type": "string" },
         "WoS_H-index": { "type": "integer" },
         "citations": { "type": "integer" } } } } }
```

Figure 7. The produced JSON schema file of scientists ("ScientistsJSONSchema_V2.json") on June 01, 2024.

```
<rdf:RDF>
  <owl:Ontology rdf:about="http://my_onto/scientist_ontology#">
    <owl:Class rdf:about="scientist"/>
    <owl:DatatypeProperty rdf:about="ResearcherID">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="firstLastName">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
```

```
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="specialty">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="WoS_H-index">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="citations">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    </owl:DatatypeProperty>
  </owl:Ontology>
</rdf:RDF>
```

Figure 8. The derived OWL2 ontology schema file of scientists ("ScientistsOntologySchema_V2.owl") on June 01, 2024.

All the necessary JSON schema change statements have been defined in our previous work on schema versioning in a time-referenced JSON-based NoSQL setting [12,13], and the necessary OWL2 ontology schema change statements have been defined in our previous work on schema versioning in a time-referenced OWL2 Semantic Web context [14,15].

In the following, we show the sequencing of JSON data update statements, JSON schema change statements, and OWL2 ontology schema change statements occurring in the simple case of a renaming statement. For example, we can consider a JSON data update statement expressed by an end user within the JUpdate language [16] as exposed in Listing 1.

```
ALTER DOCUMENT ScientistsInstances_V1.json
OBJECT $
RENAME MEMBER $.scientist.name TO firstLastName;
```

Listing 1 An example of a JSON data update statement specified in the JUpdate language

Its execution causes the automatic generation of the JSON schema change statement:

RenameProperty(ScientistsJSONSchema_V1.json,
$.properties.scientist.properties.name, firstLastName)

which is followed by the automatic generation and of the OWL2 ontology schema change statement:

RenameDataTypeProperty(ScientistsOntologySchema_V1.owl,
name, firstLastName)

The collection of consecutive JSON data update statements produced by the algorithm in Figure 1 that have been applied on ScientistsInstances_V1.json to create ScientistsInstances_V2.json, is as exposed in Listing 2.

```
UPDATE ScientistsInstances_V1.json
OBJECT $.scientist[?(@.ResearcherID=='4444-5555-6666-7777')]
SET name="Layla Ahmad Hamid", specialty="Data Science";
ALTER DOCUMENT ScientistsInstances_V1.json
OBJECT $.scientist[?(@.ResearcherID=='4444-5555-6666-7777')]
ADD MEMBER citations VALUE 280;
ALTER DOCUMENT ScientistsInstances_V1.json
OBJECT $
RENAME MEMBER $.scientist.name TO firstLastName;
```

```
ALTER DOCUMENT ScientistsInstances_V1.json
OBJECT $
RENAME MEMBER $.scientist.h-index TO WoS_H-index;
```

Listing 2. The first collection of consecutive JSON data update statements (defined in JUpdate).

The collection of consecutive JSON schema change statements that result from the execution of the above collection of consecutive JSON data update statements is as follows:

```
ChangePropertyTypeInJSONSchema(ScientistsJSONSchema_V1.json,
    $.properties.scientist.properties.specialty, string);

AddPropertyToJSONSchema(ScientistsJSONSchema_V1.json, $.properties.scientist.properties, last,
    citations);

ChangePropertyTypeInJSONSchema(ScientistsJSONSchema_V1.json,
    $.properties.scientist.properties.citations, integer);

RenamePropertyInJSONSchema(ScientistsJSONSchema_V1.json,
    $.properties.scientist.properties.name, firstLastName);
RenamePropertyInJSONSchema(ScientistsJSONSchema_V1.json,
    $.properties.scientist.properties.h-index, WoS_H-index);
```

The collection of consecutive ontology schema change statements that is derived from the execution of the above collection of consecutive JSON schema change statements is as follows:

```
DropClass(ScientistsOntologySchema_V1.owl, specialty);

AddDataProperty(ScientistsOntologySchema_V1.owl, scientist, specialty,
    http://www.w3.org/2001/XMLSchema#string);

AddDataProperty(ScientistsOntologySchema_V1.owl, scientist, citations,
    http://www.w3.org/2001/XMLSchema#int);

RenameDataProperty(ScientistsOntologySchema_V1.owl, scientist, name, firstLastName);
RenameDataProperty(ScientistsOntologySchema_V1.owl, scientist, h-index, WoS_H-index);
```

Furthermore, in order to illustrate the management of inconsistent instances that result from instance updates, assume that on July 01, 2024, the τJOWL DBA updated the current JSON data file version (ScientistsInstances_V2.json), exposed in Figure 6, by adding the three following object members to the "scientist" object whose ResearcherID member is equal to "4444-5555-6666-7777":

- "researchItems" with the integer value 35;
- "ResearchInterestScore" with the number value 72.6;
- "lastPublicationDate" with the date value "2024-06-18".

Such updates can be formulated using the JUpdate language as exposed in Listing 3.

```
ALTER DOCUMENT ScientistsInstances_V2.json
OBJECT $.scientist[?(@.ResearcherID=='4444-5555-6666-7777')]
ADD MEMBER researchItems VALUE 35;
ALTER DOCUMENT ScientistsInstances_V2.json
OBJECT $.scientist[?(@.ResearcherID=='4444-5555-6666-7777')]
ADD MEMBER ResearchInterestScore VALUE 72.6;
ALTER DOCUMENT ScientistsJSONInstances_V2.json
OBJECT $.scientist[?(@.ResearcherID=='4444-5555-6666-7777')]
ADD MEMBER lastPublicationDate VALUE "2024-06-18";
```

Listing 3. The second collection of consecutive JSON data update statements (defined in JUpdate).

All such instance update statement are non-conservative, since their results do not respect the current JSON schema version ScientistsJSONSchema_V2.json exposed in Figure 7, associated to the current JSON data file version ScientistsInstances_V2.json which is being updated.

The execution of the instance update statements above on the second JSON data file version of scientists "ScientistsInstances_V2.json" exposed in Figure 6 creates the new version

"ScientistsInstances_V3.json" exposed in Figure 9. Moreover, since such statements are non-conservative, their execution also creates, transparently to the τJOWL DBA, a collection of consecutive JSON schema change statements. Such a succession is automatically applied to the second JSON schema version "ScientistsJSONSchema_V2.json" exposed in Figure 7 in order to obtain the new JSON schema version "ScientistsJSONSchema_V3.json" exposed in Figure 10, for the new JSON data file version exposed in Figure 9. Then, and within the same update transaction, the system also derives, from the created collection of consecutive JSON schema change statements, the corresponding collection of consecutive OWL2 ontology schema change statements, and applies it to the second OWL2 ontology schema version "ScientistsOntologySchema_V2.owl" exposed in Figure 8 in order to create the new OWL2 ontology schema version "ScientistsOntologySchema_V3.owl" exposed in Figure 11. Notice that the third OWL2 ontology schema version "ScientistsOntologySchema_V3.owl" corresponds to the updated JSON Big Data instances stored in "ScientistsInstances_V3.json" of Figure 9. Changes are evidenced in red bold typeface.

```
{ "scientist":{
   "ResearcherID": "4444-5555-6666-7777",
   "firstLastName": "Layla Ahmad Hamid",
   "specialty": "Data Science",
   "WoS_H-index": 8,
   "citations": 280,
   "researchItems": 35,
   "ResearchInterestScore": 72.6,
   "lastPublicationDate": "2024-06-18" } }
```

Figure 9. The JSON data file of scientists ("ScientistsInstances_V3.json") on July 01, 2024.

```
{ "type": "object",
 "properties":
  { "scientist":
     { "type": "object",
       "properties":
         { "ResearcherID": { "type": "string" },
           "firstLastName": { "type": "string" },
           "specialty": { "type": "string" },
           "WoS_H-index": { "type": "integer" },
           "citations": { "type": "integer" },
           "researchItems": { "type": "integer" },
           "ResearchInterestScore": { "type": "number" },
           "lastPublicationDate": { "type": "string", "format":"date"} } } } }
```

Figure 10. The produced JSON schema file of scientists ("ScientistsJSONSchema_V3.json") on July 01, 2024.

```
<rdf:RDF>
  <owl:Ontology rdf:about="http://my_onto/scientist_ontology#">
    <owl:Class rdf:about="scientist"/>
    <owl:DatatypeProperty rdf:about="ResearcherID">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="firstLastName">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
```

```
    <owl:DatatypeProperty rdf:about="specialty">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="WoS_H-index">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="citations">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="researchItems">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="ResearchInterestScore">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#decimal"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="lastPublicationDate">
      <rdfs:domain rdf:resource="scientist"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
    </owl:DatatypeProperty>
  </owl:Ontology>
</rdf:RDF>
```

Figure 11. The derived OWL2 ontology schema file of scientists (“ScientistsOntologySchema_V3.owl”) on July 01, 2024.

The collection of consecutive JSON schema change statements resulting from the execution of such collection of consecutive JSON data update statements is:

AddPropertyToJSONSchema(ScientistsJSONSchema_V2.json, $.properties.scientist.properties, last, researchItems);

ChangePropertyTypeInJSONSchema(ScientistsJSONSchema_V2.json, $.properties.scientist.properties.researchItems, integer);

AddPropertyToJSONSchema(ScientistsJSONSchema_V2.json, $.properties.scientist.properties, last, ResearchInterestScore);

ChangePropertyTypeInJSONSchema(ScientistsJSONSchema_V2.json, $.properties.scientist.properties.ResearchInterestScore, number);

AddPropertyToJSONSchema(ScientistsJSONSchema_V2.json, $.properties.scientist.properties, last, lastPublicationDate);

ChangePropertyTypeInJSONSchema(ScientistsJSONSchema_V2.json, $.properties.scientist.properties.lastPublicationDate, date);

Furthermore, the collection of consecutive ontology schema change statements derived from the execution of the collection of consecutive JSON schema change statements above, is as follows:

AddDataProperty(ScientistsOntologySchema_V2.owl, scientist, researchItems,
http://www.w3.org/2001/XMLSchema#int);

AddDataProperty(ScientistsOntologySchema_V2.owl, scientist, ResearchInterestScore,
http://www.w3.org/2001/XMLSchema#decimal);

AddDataProperty(ScientistsOntologySchema_V2.owl, scientist, lastPublicationDate,
http://www.w3.org/2001/XMLSchema#date);

5. Implementation

In order to show its feasibility, a tool that supports the proposed approach, named BatchOntologyInstanceUpdate-Processor, is under development. The BatchOntologyInstanceUpdate-Processor will be integrated in the τJOWL-Manager system that implements the whole τJOWL framework[10].

The BatchOntologyInstanceUpdate-Processor has been developed as a stratum, as shown in Figure 12. The stratum, written in Java, is composed of four modules: "New Ontology Instance File Uploader", "Ontology Instance Conformance Checker", "Ontology Schema Change Statement Generator", and "Schema Change Statement Sequence Optimizer".

As far as the functioning of our tool is concerned, the τJOWL DBA first uploads, via the use of the "New Ontology Instance File Uploader" module, a new time-referenced JSON data file that stores a new ontology instance version that has to be merged with the current ontology instance version to produce a new one.

Then, the "Ontology Instance Conformance Checker" checks the conformity of the new ontology instance version that has been uploaded. If it is not conformant to the current ontology schema version, the "Ontology Schema Change Statement Generator" module is called to automatically generate a sequence of ontology schema change statements.

Next, such a sequence is submitted to "Schema Change Statement Sequence Optimizer" that generates a change execution plan for the sequence of statements produced by the "Ontology Schema Change Statement Generator".

Finally, this change execution plan is submitted to the "Extraction Schema" sub-module of the "Ontology Building" module that completes the process of generating a new ontology schema version, in a transparent manner to the τJOWL DBA, as explained in Sec. 2.2 of [10].

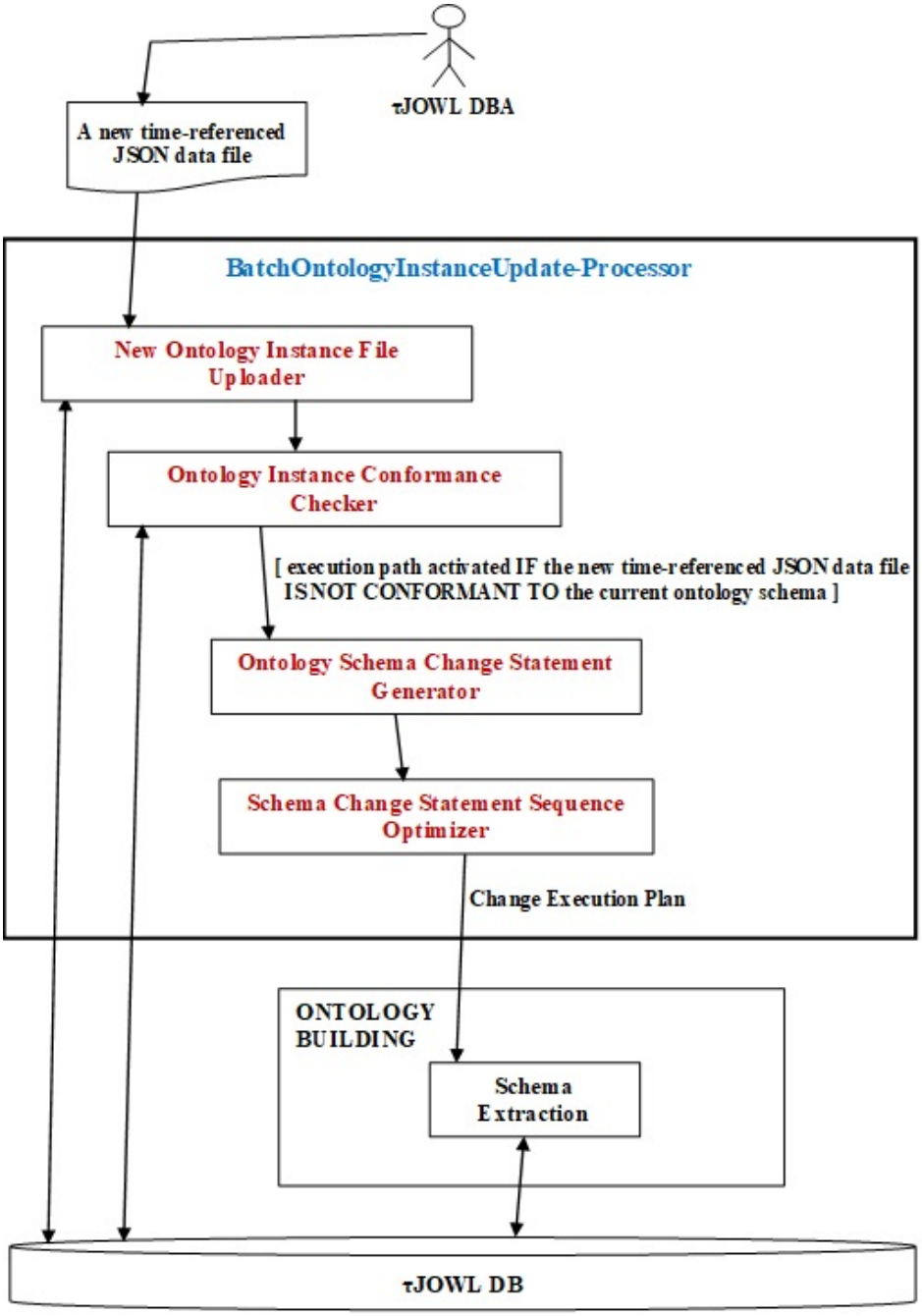

Figure 12. Architecture of the BatchOntologyInstanceUpdate-Processor tool.

6. Related Work Discussion

Updates to ontology instances and their effects have been studied in the literature concerning ontologies of data. Two excellent surveys on ontology evolution have been provided in [17] and [18]. Notice that all papers reviewed in such two works are considered as related to our present paper.

As for the ontology instance update mode, to the best of our knowledge, all existing research work have dealt with interactive and incremental updates of ontology instances (i.e., updates that are done while interacting with the system), like [19], [20], [21], [22], and [23]. We are the first to propose handling ontology instance updates in a batch mode, i.e., instead of interactively/incrementally specifying a collection of consecutive instance update statements on some data file version, the user provides a new entire JSON data file version and asks the system to take it into account.

As for the impact on the ontology schema, although there are a lot of research work that have studied ontology instance updates which have no effect at schema level, like [24] and [25], several other research work have investigated the effects of ontology instance updates on the corresponding ontology schema, like [26] and [11].

Furthermore, multiple research works have dealt with instance updates (or adaptation) that result from ontology schema changes, like [14] and [27].

As far as existing ontology evolution tools are concerned, neither the seven tools reviewed in [28] (PromptDiff, WebProtégé, CODEX, KAON, OWL Diff, OntoDiffGraph, and OIM), nor the τOWL-Manager tool presented in [14,26] do support managing ontology instance updates in a batch mode; only the interactive/incremental one is supported.

It is worth mentioning also that SPARQL 1.1 Update [29], which is the W3C recommendation for updating RDF files (frequently used to store ontology instances), do not support batch mode ontology instance updates.

With regard to our previous work [10,14,26], the major contribution of this article is that it deals with ontology instance updates that, in the τJOWL environment, are performed in batch mode. Indeed, our present approach considers an ontology instance update that consists of the creation of an entire new ontology instance version, instead of the application of isolated ontology instance update statements specified in some instance update language (e.g., JUpdate [16], τJUpdate [30], SPARQL 1.1 Update [29], XQuery Update Facility [31]). Besides, it is also worth mentioning that JUpdate [16] and τJUpdate [30], which are among our previous work and are used to update conventional (i.e., non-temporal) JSON data and time-varying JSON data, respectively, do not provide any support for batch mode JSON instance updates.

Table 1 compares the ontology instance update approaches mentioned above.

Table 1. A comparative table of ontology instance update approaches.

Approach	**Ontology instance update mode**	**Ontology instance updates could have an impact on the ontology schema**	**Ontology instance updates (adaptations) could result from ontology schema changes**	**Implementation**
[19]	Interactive/Incremental	No	Yes	Yes
[20]	Interactive/Incremental	No	Yes	No
[21]	Interactive/Incremental	No	Yes	No
[22]	Interactive/Incremental	No	Yes	No
[23]	Interactive/Incremental	No	Yes	No
[24]	Interactive/Incremental	No	Yes	No
[25]	Interactive/Incremental	No	Yes	No
[10]	Interactive/Incremental	Yes	Yes	No
[26]	Interactive/Incremental	Yes	Yes	Yes
[11]	Interactive/Incremental	Yes	Yes	No
[14]	Interactive/Incremental	No	Yes	Yes
[27]	Interactive/Incremental	No	Yes	No
[28]	Interactive/Incremental	No	Yes	Yes
[29]	Interactive/Incremental	No	Yes	No
Our present approach	**Batch**	**Yes**	**Yes**	**Yes**

7. Limitations and Extensions

As for the limitations of our approach, we think that it cannot be applied in an Open World Assumption (OWA) [32] context. Indeed, our approach is a database-like approach and is based on the usual conformity of ontology instances to their ontology schema; hence, it can be applied only in a Closed World Assumption (CWA) [33], by means of the Data Box (DBox) notion [34]. Recall that in the ontology world, the traditional viewpoint is the OWA 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 CWA, which means that "what is not known to be true must be false" and which is used 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. Notice that, with the CWA, an ontology definition (concepts, relationships between concepts, axioms…) behaves like a database schema and, thus, can be called ontology schema; the ontology individuals are

instances of such an ontology schema. Besides, it should be noted that the addition of a new ontology instance version has to be always under the control of the τJOWL DBA and approved by him/her, otherwise a malicious instance could be submitted to completely destroy the ontology.

As far as possible extensions of our proposal are concerned, it is worth mentioning that although our approach has been defined based on the couple JSON (for ontology instances) and OWL2 (for ontology schema), it is not tightly coupled with these two formalisms. In fact, our approach could easily be generalized to other data models and ontology languages, and be applied outside the JSON/OWL2 environment. We think that whatever the data format (e.g., RDF, XML, JSON) used for the storage of ontology instances and whatever the schema language (e.g., OWL2, RDFS, XML Schema, JSON Schema, Turtle) used for the specification of ontology schemas, our approach will always behave as described in the present work.

8. Conclusion

In this paper, we extend our τJOWL ecosystem by dealing with time-referenced JSON data updates that are executed in a batch mode. This scenario happens when the τJOWL DBA would like to integrate into the τJOWL DB, a new time-referenced JSON data file that is prepared offline or imported from an external source. Such a data file requires to be incorporated with the current OWL2 ontology instance to create a new version. Unlike existing approaches for automatic ontology maintenance, which are focused on interactive and incremental ontology instance updates —where updates are made one by one during user update sessions— the present work (and to the best of our knowledge) is the first to address ontology instance updates executed in a batch mode. In this mode, the user provides (as a JSON file) a new portion of the instance data to be incorporated into the ontology, and the management system processes this entire batch update at once producing a new consistent ontology version also containing the new instance data. Building on our prior research, the main contribution of this paper lies in enabling batch processing of ontology instance updates within the τJOWL ecosystem. Rather than applying isolated update statements stepwise (as usually made by means of update languages such as JUpdate, τJUpdate, SPARQL 1.1 Update, or XQuery Update Facility), our approach treats the update as the creation of a whole new version of the ontology instance. In fact, previous solutions, including our own for τJOWL based on the update languages JUpdate and τJUpdate, did not support batch mode updates. Thus, this work introduces a significant advancement in automated ontology instance management by extending the update paradigm from interactive and incremental, statement-based changes to comprehensive batch revisions.

In addition, some instance updates may render them inconsistent with respect to their ontology schema. Hence, our solution in this work is to automatically create change statements to the OWL2 ontology schema and execute them on this latter to maintain global consistency. We think that our current proposal for managing ontology instance updates within τJOWL, improves flexibility in knowledge management and ontology evolution, and guarantees that the τJOWL DB remains consistent in a transparent way.

Currently, we are developing a tool that shows the feasibility of our approach. We will use it for experimental evaluation of our proposal.

In our future work, we plan to extend our present work by dealing with complex instance updates, i.e., updates involving complex JSON components (JSON objects, and JSON arrays), and also those involving an entire JSON data file (e.g., splitting a JSON data file into several ones) or multiple data files (e.g., merging two JSON data files into one file).

References

1. Davoudian, A., and Liu, M. (2020). Big Data Systems: A Software Engineering Perspective. *ACM Computing Surveys (CSUR)*, 53(5), Article 110. https://doi.org/10.1145/3408314
2. IETF. (2017). The JavaScript Object Notation (JSON) Data Interchange Format. Internet Standards Track document, December 2017. https://tools.ietf.org/html/rfc8259 (accessed: 2025-08-26)
3. IETF. (2022). JSON Schema: A Media Type for Describing JSON Documents. Internet-Draft, 16 June 2022. https://json-schema.org/draft/2020-12/json-schema-core (accessed: 2025-08-26)
4. Eine, B., Jurisch, M., and Quint, W. (2017). Ontology-based big data management. *Systems*, Vol. 5, No. 3, Article 45. https://doi.org/10.3390/systems5030045
5. Konys, A. (2017). Ontology-based approaches to big data analytics. In *Proceedings of the 20th International Conference on Advanced Computer Systems (ACS 2016)*, Miedzyzdroje, Poland, 19-21 October 2016. Advances in Intelligent Systems and Computing (AISC), vol. 534, 2017, pp. 355-365. Springer, Cham.
6. 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*, Vol. 79, pp. 3-19. https://doi.org/10.1016/j.is.2018.01.006
7. Sayah, Z., Kazar, O., Lejdel, B., Laouid, A., and Ghenabzia, A. (2021). An intelligent system for energy

management in smart cities based on big data and ontology. *Smart and Sustainable Built Environment*, Vol. 10, No. 2, pp. 169-192. https://doi.org/10.1108/SASBE-07-2019-0087
8. 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*, Vol. 7, pp. 65-85. https://doi.org/10.1007/s13740-018-0086-2
9. W3C. (2012). OWL2 Web Ontology Language – Primer (Second Edition). W3C Recommendation, 11 December 2012. http://www.w3.org/TR/owl2-primer/ (accessed: 2025-08-26)
10. 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*, Vol. 5, No. 4, pp. 271-281. DOI: 10.26599/BDMA.2021.9020019
11. Brahmia, S., Brahmia, Z., Grandi, F., and Bouaziz, R. (2023). Management of Implicit Ontology Changes Generated by Non-conservative JSON Instance Updates in the τJOWL Environment. In *Proceedings of the 6th International Conference on Information and Knowledge Systems (ICIKS'2023)*, Portsmouth, UK, 22-23 June 2023, LNBIP Vol. 486, Springer Nature Switzerland AG, 2024, pp. 213-226.
12. Brahmia, S., Brahmia, Z., Grandi, F., and Bouaziz, R. (2017). Temporal JSON Schema Versioning in the τJSchema Framework. Journal of Digital Information Management, Vol. 15, No. 4, pp. 179-202. DOI: 10.6025/jdim/2017/15/4/179-202
13. 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, Vol. 10, No. 5/6, pp. 442-479. DOI: 10.1504/IJCC.2021.10030585
14. 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*, Vol. 5, No. 3, pp. 141-163. https://doi.org/10.1007/s13740-016-0066-3
15. 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*, Vol. 26, No. 2, pp. 113-137. https://doi.org/10.1080/12460125.2017.1252230
16. Brahmia, Z., Brahmia, S., Grandi, F., and Bouaziz, R. (2022). JUpdate: a JSON update language. *Electronics*, Vol. 11, No. 4, Article 508. https://doi.org/10.3390/electronics11040508
17. Zablith, F., Antoniou, G., d'Aquin, M., Flouris, G., Kondylakis, H., Motta, E., Plexousakis, D., and Sabou, M. (2015). Ontology evolution: a process-centric survey. *The Knowledge Engineering Review*, Vol. 30, No. 1, pp. 45-75. https://doi.org/10.1017/S0269888913000349
18. Da Silva, V. T., Dos Santos, J. S., Thiago, R., Soares, E., and Azevedo, L. G. (2022). OWL ontology evolution: understanding and unifying the complex changes. *The Knowledge Engineering Review*, Vol. 37, Article e10. https://doi.org/10.1017/S0269888922000066
19. Ahmeti, A. (2020). *Updates in the Context of Ontology-Based Data Management*. PhD thesis dissertation, Faculty of Informatics at the Vienna University of Technology, Vienna, Austria.
20. Kotis, K. I., Vouros, G. A., and Spiliotopoulos, D. (2020). Ontology engineering methodologies for the evolution of living and reused ontologies: status, trends, findings and recommendations. *The Knowledge Engineering Review*, Vol. 35, Article e4. https://doi.org/10.1017/S0269888920000065
21. Slota, M., Leite, J., and Swift, T. (2015). On updates of hybrid knowledge bases composed of ontologies and rules. *Artificial Intelligence*, Vol. 229, pp. 33-104. https://doi.org/10.1016/j.artint.2015.07.008
22. Solimando, A., and Guerrini, G. (2013). Ontology adaptation upon updates. In *Proceedings of the Extended Semantic Web Conference (ESWC) 2013 Satellite Events*, Montpellier, France, 26-30 May 2013, LNCS 7955, pp. 34-45.
23. Sangers, J., Hogenboom, F., and Frasincar, F. (2012). Event-driven ontology updating. In *Proceedings of the 13th International Conference on Web Information Systems Engineering (WISE 2012)*, Paphos, Cyprus, 28-30 November 2012, LNCS 7651, pp. 44-57.
24. Flahive, A., Taniar, D., Rahayu, J. W., and Apduhan, B. O. (2015). A methodology for ontology update in the semantic grid environment. *Concurrency and Computation: Practice and Experience*, Vol. 27, No. 4, pp. 782-808. https://doi.org/10.1002/cpe.2841
25. Bayoudhi, L., Sassi, N., and Jaziri, W. (2019). Efficient management and storage of a multiversion OWL2 DL domain ontology. *Expert Systems*, Vol. 36, No. 2, Article e12355. https://doi.org/10.1111/exsy.12355
26. Brahmia, Z., Grandi, F., Zekri, A., and Bouaziz, R. (2022). Ontology Versioning Driven By Instance Evolution in the τOWL Framework. *Journal of Information and Knowledge Management*, Vol. 21, No. 1, Article 2250002. https://doi.org/10.1142/S0219649222500022
27. Kondylakis, H., and Papadakis, N. (2018). EvoRDF: evolving the exploration of ontology evolution. *The Knowledge Engineering Review*, Vol. 33, Article e12. DOI: 10.1017/S0269888918000140
28. Santos, J. S., Silva, V. T., Azevedo, L. G., Soares, E. F., and Thiago, R. M. (2020). An Experimental Analysis of Tools for Ontology Evolution Management. In *Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS'2020)*, Online streaming, 5-7 May 2020, Volume 2, pp. 111-121.
29. W3C. (2013). SPARQL 1.1 Update. W3C Recommendation, 21 March 2013. https://www.w3.org/TR/2013/REC-sparql11-update-20130321/ (accessed: 2025-08-26)
30. Brahmia, Z., Grandi, F., Brahmia, S., and Bouaziz, R. (2024). τJUpdate: An update language for time-varying JSON data. *Journal of Computer Languages*, Vol. 79, Article no. 101258. https://doi.org/10.1016/j.cola.2024.101258
31. W3C. (2017). XQuery Update Facility 3.0, W3C Working Group Note, 24 January 2017.

https://www.w3.org/TR/2017/NOTE-xquery-update-30-20170124/ (accessed: 2025-08-26)
32. Patel-Schneider, P. F., & Horrocks I. (2007). A comparison of two modelling paradigms in the Semantic Web. *Journal of Web Semantics, 5*(4), pp. 240-250.
33. Etzioni, O., Golden, K., & Weld, D. S. (1997). Sound and efficient closed-world reasoning for planning. *Artificial Intelligence, 89*(1-2), pp. 113-148.
34. Seylan, İ., Franconi, E., & De Bruijn, J. (2009). Effective query rewriting with ontologies over DBoxes. In *Proceedings of the 21st International Joint Conf. on Artificial Intelligence (IJCAI 2009)*, pp. 923-929.