τOWL: A Framework for Managing Temporal Semantic Web Documents Supporting Temporal Schema Versioning

Abir Zekri
University of Sfax
Sfax, Tunisia
abir.zekri@fsegs.rnu.tn

Zouhaier Brahmia
University of Sfax
Sfax, Tunisia
zouhaier.brahmia@fsegs.rnu.tn

Fabio Grandi
University of Bologna
Bologna, Italy
fabio.grandi@unibo.it

Rafik Bouaziz
University of Sfax
Sfax, Tunisia
raf.bouaziz@fsegs.rnu.tn

***Abstract* — The OWL 2 Web Ontology Language allows defining both schema and instances of ontologies for Semantic Web applications, but lacks explicit support for time-varying ontologies. Hence, knowledge engineers or maintainers of Semantic Web documents have to use *ad hoc* techniques in order to specify an OWL 2 schema for time-varying instances and to cope with its temporal evolution. In this paper, for a disciplined and systematic approach to the temporal management of OWL 2 ontologies, we propose the adoption of a framework called Temporal OWL 2 (τOWL), inspired by the Temporal XML Schema (τXSchema) framework defined for XML data. Hence, τOWL allows creating a temporal OWL 2 ontology from a conventional (i.e., non-temporal) OWL 2 ontology and a set of logical and physical annotations. Logical annotations identify which elements of a Semantic Web document can vary over time and physical annotations specify how the time-varying aspects are represented in the document. By using annotations to integrate temporal aspects in the traditional Semantic Web, our framework (i) guarantees logical and physical data independence for temporal schemas and (ii) provides a low-impact solution since it requires neither modifications of existing Semantic Web documents, nor extensions to the OWL 2 recommendation and Semantic Web standards. Furthermore, temporal versioning of the schema itself is supported in τOWL by means of a temporal schema, which is a document that binds the three components of a τOWL schema to the temporal versions they belong to. In τOWL, either the conventional schema and the temporal schema can be versioned, by means of two dedicated complete sets of schema change primitives, which are defined in this work. We also illustrate their use and show their impact on OWL 2 instances through an example.**

Keywords – Semantic Web; Ontology; OWL 2; τXSchema; Logical annotations; Physical annotations; Temporal database; XML Schema; XML; τOWL; Conventional schema; Temporal schema; Schema versioning; Temporal ontology; Ontology versioning

I. INTRODUCTION

Time is an omnipresent dimension in both modern [1] and classical [2] applications; it is used to timestamp data values to keep track of changes in the real world and model their history. Hence, studying time has been, and continues to be, one of the main research interests in different scientific fields, such as databases and knowledge representation.

Since the second half of the 1980s, a great deal of work has been done in the field of temporal databases [3][4][5]. Several data models and query languages have been proposed for the management of time-varying data. Temporal databases usually adopt one or two time dimensions to timestamp data: (a) transaction time, which indicates when an event is recorded in the database, and (b) valid time, which represents the time when an event occurred, occurs or is expected to occur in the real world. Bi-temporal data are timestamped by both transaction time and valid time dimensions. Snapshot data are traditional data, without time support.

On the other hand, the World Wide Web (WWW or Web) [6] was shifted from the semi-structured internet to a more structured Web called the Semantic Web [7][8]. The new generation of Web aims at providing languages and tools that specify explicit semantics for data and enable knowledge sharing among knowledge-based applications. In this vision, ontologies [9] are used for defining and relating concepts that describe Web resources, in a formal way. The new emerging standard for describing ontologies, which has been recommended by the W3C since 2009, is OWL 2 [10][11][12]. It allows defining both schema (in terms of entities, axioms, and expressions) and instances (i.e., individuals) of ontologies; OWL 2 ontologies are stored as Semantic Web documents.

Due to the dynamic nature of the Web, ontologies that are used on the Web (like other Web application components such as Web databases, Web pages and Web scripts) evolve over time to reflect and model changes occurring in the real-world. Furthermore, several Semantic Web-based applications (like e-commerce, e-government and e-health applications) require keeping track of ontology evolution and versioning with respect to time, in order to represent, store and retrieve time-varying ontologies.

Unfortunately, while there is a sustained interest for temporal and evolution aspects in the research community [13], existing Semantic Web standards but also state-of-the-art ontology editors and knowledge representation tools do not provide any built-in support for managing temporal ontologies. In particular, the W3C OWL 2 recommendation lacks explicit support for time-varying ontologies, at both schema and instance levels. Thus, knowledge engineers or maintainers of semantics-based Web resources must use *ad* *hoc* techniques when there is a need, for example, to specify an OWL 2 ontology schema for time-varying ontology instances or to deal with temporal evolution of the ontology schema itself. In the rest of the paper, we define as Knowledge Base Administrator (KBA) a knowledge engineer or, more in general, the person in charge of the maintenance of semantics-based Web resources.

According to what precedes, we think that if we would like to handle ontology evolution over time in an efficient manner and to allow historical queries to be executed on time-varying ontologies, a built-in temporal ontology management system is needed. For that purpose, we propose in this paper a framework, called τOWL, for managing temporal Semantic Web documents, through the use of a temporal OWL 2 extension. In fact, we want to introduce with τOWL a principled and systematic approach to the temporal extension of OWL 2, similar to that Snodgrass and colleagues did to the XML language with τXSchema [14][15][16]. τXSchema is a framework (i.e., a data model equipped with a suite of tools) for managing temporal XML documents, well known in the database research community and, in particular, in the field of temporal XML [17]. Moreover, in our previous work [18][19][20], with the aim of completing the framework, we augmented τXSchema by defining necessary schema change operations acting on conventional schema, temporal schema, and logical and physical annotations (extensions which we plan to apply to τOWL too).

Being defined as a τXSchema-like framework, τOWL facilitates the creation of a temporal OWL 2 ontology from a conventional (i.e., non-temporal) OWL 2 ontology specification and a set of logical (or temporal) and physical annotations. Logical annotations identify which components of a Semantic Web document can vary over time; physical annotations specify how the time-varying aspects are represented in the document. By using temporal schema and annotations to introduce temporal aspects in the conventional (i.e., non temporal) Semantic Web, our framework (i) guarantees logical and physical data independence [21] for temporal schemas and (ii) provides a low-impact solution since it requires neither modifications of existing Semantic Web documents, nor extensions to the OWL 2 recommendation and Semantic Web standards.

Furthermore, with respect to the preliminary version of this work presented at SEMAPRO 2014 [1], in this paper we extend the τOWL framework to also support schema versioning [22][23], which is the most powerful technique for managing the history of schema changes. Since ontology schemata are also evolving over time to reflect changes in real-world applications [24], keeping a fully fledged history of ontology changes (i.e., involving both the ontology instances and the ontology schema) is a very required feature for many Semantic Web applications. More precisely, we present our technique for the versioning of a τOWL schema, and define necessary schema change operations acting on conventional ontology schema and on temporal ontology schema. We do not deal in this paper with changes acting on logical and physical annotations; that will be studied in a future work.

The remainder of the paper is organized as follows. Section II motivates the need for an efficient management of time-varying Semantic Web documents. Section III describes the τOWL framework that we propose for extending the Semantic Web to temporal aspects: the architecture of τOWL is presented and details on all its components and support tools are given. Section IV presents our approach for versioning of a τOWL schema. Section V introduces the schema change primitives that we propose, in the τOWL framework, for changing the conventional schema and for updating the temporal schema. Section VI discusses related work. Section VII provides a summary of the paper and some remarks about our future work.

II. MOTIVATION

In this section, we present a motivating example that shows the limitation of the OWL 2 language for explicitly supporting time-varying instances. Then, we state the desiderata for an OWL 2 extension, which could accommodate time-varying instances in a disciplined and systematic way.

A. *Running Example*

As a motivating and illustrative example for τOWL, we recall and extend the example presented in the preliminary version of this work [1], dealing with the management of the evolution of an ontology based on Friend Of A Friend (FOAF). The FOAF project [25] is creating a Web of machine-readable pages describing people, the links between them and the things they create and do.

Suppose that the Web site “Web-S1” publishes the FOAF definition for his user “Nouredine”. A fragment of the FOAF Resource Description Framework (RDF [26]) document of “Nouredine” is presented in Figure 1. It describes, according to the FOAF ontology, the personal information of “Nouredine” (i.e., name and nickname) and the information about his online accounts on diverse sites (i.e., the home page of the site, and the account name of the user). In this example, we limit to describe user’s information concerning the account on the online Web site “Facebook”.

```
…
<foaf:Person rdf:ID="#Person1">
  <foaf:name>Nouredine Tounsi</foaf:name>
  <foaf:nick>Nor</foaf:nick>
  <foaf:holdsAccount>
    <foaf:OnlineAccount
          rdf:about="https://www.facebook.com/
          Nouredine.Tounsi">
      <foaf:accountName>Nor_Tunsi
      </foaf:accountName>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
…
```

Figure 1. A fragment of Nouredine FOAF RDF document on January 15, 2014.

Assume that information about the user “Nouredine” of the Web site “Web-S1” was added on January 15, 2014. On February 08, 2014, Nouredine modified his nickname from

"Nor" to "Nouri" and his account name of Facebook from "Nor_Tunsi" to "Nouri_Tunsi". Thus, the corresponding fragment of the Nouredine FOAF RDF document was revised to that shown in Figure 2.

```
…
<foaf:Person rdf:ID="#Person1">
 <foaf:name>Nouredine Tounsi</foaf:name>
 <foaf:nick>Nouri</foaf:nick>
 <foaf:holdsAccount>
  <foaf:OnlineAccount
        rdf:about="https://www.facebook.com/
        Nouredine.Tounsi">
   <foaf:accountName>Nouri_Tunsi
   </foaf:accountName>
  </foaf:OnlineAccount>
 </foaf:holdsAccount>
</foaf:Person>
...
```

Figure 2. A fragment of Nouredine FOAF RDF document on February 08, 2014.

In many Semantic Web-based applications, the history of ontology changes is a fundamental requirement, since such a history allows recovering past ontology versions, tracking changes over time, and evaluating temporal queries [27]. A τOWL time-varying Semantic Web document records the evolution of a Semantic Web document over time by storing all versions of the document in a way similar to that originally proposed for τXSchema [14].

Suppose that the webmaster of the Web site "Web-S1" would like to keep track of the changes performed on our FOAF RDF information by storing both versions of Figure 1 and of Figure 2 in a single (temporal) RDF document. As a result, Figure 3 shows a fragment of a time-varying Semantic Web document that captures the history of the specified information concerning "Nouredine".

```
…
<foaf:Person rdf:ID="#Person1">
 <foaf:name>Nouredine Tounsi</foaf:name>
 <versionedNick>
  <NickVersion>
   <nickValidityStartTime>2014-01-15
   </nickValidityStartTime>
   <nickValidityEndTime>2014-02-07
   </nickValidityEndTime>
   <foaf:nick>Nor</foaf:nick>
  </NickVersion>
  <NickVersion>
   <nickValidityStartTime>2014-02-08
   </nickValidityStartTime>
   <nickValidityEndTime>now
   </nickValidityEndTime>
   <foaf:nick>Nouri</foaf:nick>
  </NickVersion>
 </versionedNick>
 <foaf:holdsAccount>
  <foaf:OnlineAccount
        rdf:about="https://www.facebook.com/
        Nouredine.Tounsi">
   <versionedAccountName>
    <AccountNameVersion>
     <accountNameValidityStartTime>
         2014-01-15
     </accountNameValidityStartTime>
     <accountNameValidityEndTime>
         2014-02-07
     </accountNameValidityEndTime>
     <foaf:accountName>Nor_Tunsi
     </foaf:accountName>
    </AccountNameVersion>
    <AccountNameVersion>
     <accountNameValidityStartTime>
         2014-02-08
     </accountNameValidityStartTime>
     <accountNameValidityEndTime>
         now
     </accountNameValidityEndTime>
     <foaf:accountName>Nouri_Tunsi
     </foaf:accountName>
    </AccountNameVersion>
   </versionedAccountName>
  </foaf:OnlineAccount>
 </foaf:holdsAccount>
</foaf:Person>
...
```

Figure 3. A fragment of the time-varying Nouredine FOAF RDF document.

In this example, we use valid-time to capture the history of such information. In order to timestamp the entities which can evolve over time, we use the following optional tags: **nickValidityStartTime** and **nickValidityEndTime**, for recording **nick** name evolution, and **accountNameValidityStartTime** and **accountNameValidityEndTime**, for keeping the **accountName** history. These are optional Data Properties which can be added to a temporal entity. The domain of nickValidityEndTime or accountNameValidityEndTime includes the value "now" [28]; the entity that has "now" as the value of its validity end time property represents the current entity until some change occurs.

Assume that the extract of the FOAF ontology presented in Figure 4 contains the conventional (i.e., non-temporal) schema [14] for the FOAF RDF document presented in both Figure 1 and Figure 2. The conventional schema is the schema for an individual version, which allows updating and querying individual versions.

```
<rdf:RDF>
 <owl:Ontology  rdf:about="http://purl.org/
                           az/foaf#">
  <rdfs:Class  rdf:about="#Person">
   <rdf:type  rdf:resource="http://www.w3.org/
                              2002/07/owl#Class"/>
  </rdfs:Class>
  <rdf:Property  rdf:about="#holdsAccount">
   <rdf:type  rdf:resource="http://www.w3.org/
                     2002/07/owl#ObjectProperty"/>
   <rdfs:domain  rdf:resource="#Person"/>
   <rdfs:range  rdf:resource="#OnlineAccount"/>
  </rdf:Property>
  <rdf:Property  rdf:about="#accountName">
   <rdf:type  rdf:resource="http://www.w3.org/
                   2002/07/owl#DatatypeProperty"/>
   <rdfs:domain  rdf:resource="#OnlineAccount"/>
  </rdf:Property>
  …
</rdf:RDF>
```

Figure 4. An RDF/XML extract from the OWL 2 FOAF ontology.

The problem is that the time-varying ontology document (see Figure 3) does not conform to the conventional ontology schema (see Figure 4). Thus, to resolve this problem, we need a different ontology schema that can describe the structure of the time-varying ontology document. This new schema should specify, for example, timestamps associated to entities, time dimensions involved, and how the entities vary over time. This example will be continued in Section III.A, in order to show how these problems can be solved in our proposed τOWL framework.

Furthermore, we want our framework also allows KBAs to effect and keep track of changes to the conventional schema itself. In Section V.D, we will complete this example by describing some changes made by the KBA on this initial framework and showing their effects both at schema and at instance levels.

B. Desiderata

There are several goals that can be fulfilled when augmenting the OWL 2 language to support time-varying instances. Our approach aims at satisfying the following requirements:

- facilitating the management of time for KBAs;
- supporting both valid time and transaction time;
- supporting (temporal) versioning of OWL 2 ontology instances;
- keeping compatibility with existing OWL 2 W3C recommendations, standards, and editors, without requiring any changes to these recommendations, standards, and tools;
- supporting existing applications that are already using OWL 2 ontologies;
- providing OWL 2 data independence so that changes at the logical level are isolated from those performed at the physical level, and vice versa;
- accommodating a variety of physical representations for time-varying OWL 2 instances;
- supporting (temporal) versioning of OWL 2 ontology schemata.

III. The τOWL Framework

In this section, we present our τOWL framework for handling temporal Semantic Web documents and provide an illustrative example of its use. We describe the overall architecture of τOWL and the tools used for managing both τOWL schema and τOWL instances. Since τOWL is a τXSchema-like framework, we were inspired by the τXSchema architecture and tools while defining the architecture and tools of τOWL.

The τOWL framework allows a KBA to create a temporal OWL 2 schema for temporal OWL 2 instances from a conventional OWL 2 schema, logical annotations, and physical annotations. Since it is a τXSchema-like framework, τOWL use the following principles:

- separation between (i) the conventional (i.e., non-temporal) schema and the temporal schema, and (ii) the conventional instances and the temporal instances;
- use of temporal and physical annotations to specify temporal and physical aspects, respectively, at schema level.

Figure 5 illustrates the architecture of τOWL. Notice that only the components that are presented in the figure as rectangular pink boxes with bold border are specific to an individual time-varying OWL 2 document and need to be supplied by a KBA. The framework is based on the OWL 2 language [10], which is a W3C standard ontology language for the Semantic Web. It allows defining both schema (i.e., entities, axioms, and expressions) and instances (i.e., individuals) of ontologies. Thus, we consider that the signature of an OWL 2 ontology O can be defined as follows: O = {E, A, Exp} such that:

i) E = {C, DP, OP, AP} represents the set of the entities with:
- C: Class, represents the set of concepts;
- DP: Data Property, represents the set of properties of the concepts;
- OP: Object Property, represents the set of the semantic relations between the concepts;
- AP: Annotation Property, represents the set of annotations on the entities and those on the axioms.

ii) A = {EAx, KAx} represents the set of axioms with:
- EAx: Entity Axioms, represents the axioms which concern the entities;
- KAx: Key Axioms, represents all the identifiers associated to the various classes.

iii) Exp = {CE, OPE, DPE} represents the set of the used expressions (an expression is a complex description which results from combinations of entities by using constructors such as enumeration, restriction of cardinality and restriction of properties) with:
- CE: Class Expressions, represents the set of combinations of concepts by using constructors;
- OPE: Object Property Expressions, represents the set of combinations of relations;
- DPE: Data Property Expressions, represents the set of combinations of properties.

The KBA starts by creating the *conventional schema* (box 7), which is an OWL 2 ontology that models the concepts of a particular domain and the relations between these concepts, without any temporal aspect. To each conventional schema corresponds a set of conventional (i.e., non-temporal) OWL 2 instances (box 12). Any change to the conventional schema is propagated to its corresponding instances. Notice that our approach deals with OWL 2 ontologies with an RDF/XML syntax [29], which is, according to the OWL 2 specification document [11], the only syntax that must mandatorily be supported by OWL 2 tools.

After that, the KBA augments the conventional schema with *logical* and *physical annotations*, which allow him/her to express, in an explicit way, all requirements dealing with the representation and the management of temporal aspects associated to the components of the conventional schema, as described in the following.

Logical annotations [16] allow the KBA to specify:

1) whether a conventional schema component varies over valid time and/or transaction time;

2) whether its lifetime is described as a continuous state or a single event;

3) whether the component may appear at certain times (and not at others);

4) whether its content changes.

If no logical annotations are provided, the default logical annotation is that anything can change. However, once the conventional schema is annotated, components that are not described as time-varying are static and, thus, they must have the same value across every instance document (box 12).

Physical annotations [16] allow the KBA to specify the timestamp representation options chosen, such as where the timestamps are placed and their kind (i.e., valid time or transaction time) and the kind of representation adopted. The location of timestamps is largely independent of which components vary over time. Timestamps can be located either on time-varying components (as specified by the logical annotations) or somewhere above such components. Two OWL 2 documents with the same logical information will look very different if we change the location of their physical timestamps. Changing an aspect of even one timestamp can make a big difference in the representation. τOWL supplies a default set of physical annotations, which is to timestamp the root element with valid and transaction times. However, explicitly defining them can lead to more compact representations [16].

In order to improve conceptual clarity and also to enable a more efficient implementation, we adopt a "separation of concerns" principle in our approach: since the entities, the axioms and the expressions of an OWL 2 ontology evolve over time independently, we distinguish between three separate types of annotations to be defined and to be associated to a conventional schema: the *entity annotations* (box 9), the *axiom annotations* (box 10) and the *expression annotations* (box 11).

Entity annotations describe the logical and physical characteristics associated to the components of an OWL 2 ontology: classes, relations, and properties. They indicate for example the temporal formats of these components, which could be valid-time, transaction-time, bi-temporal or snapshot (by default). The schema for the logical and physical entity annotations is given by *EntASchema* (box 4). Axiom annotations and expression annotations describe the logical and physical aspects of axioms and expressions defined on classes or on properties. The schema for the logical and physical axiom annotations is given by *AxiASchema* (box 5) and the schema for the logical and physical expression annotations is given by *ExpASchema* (box 6).

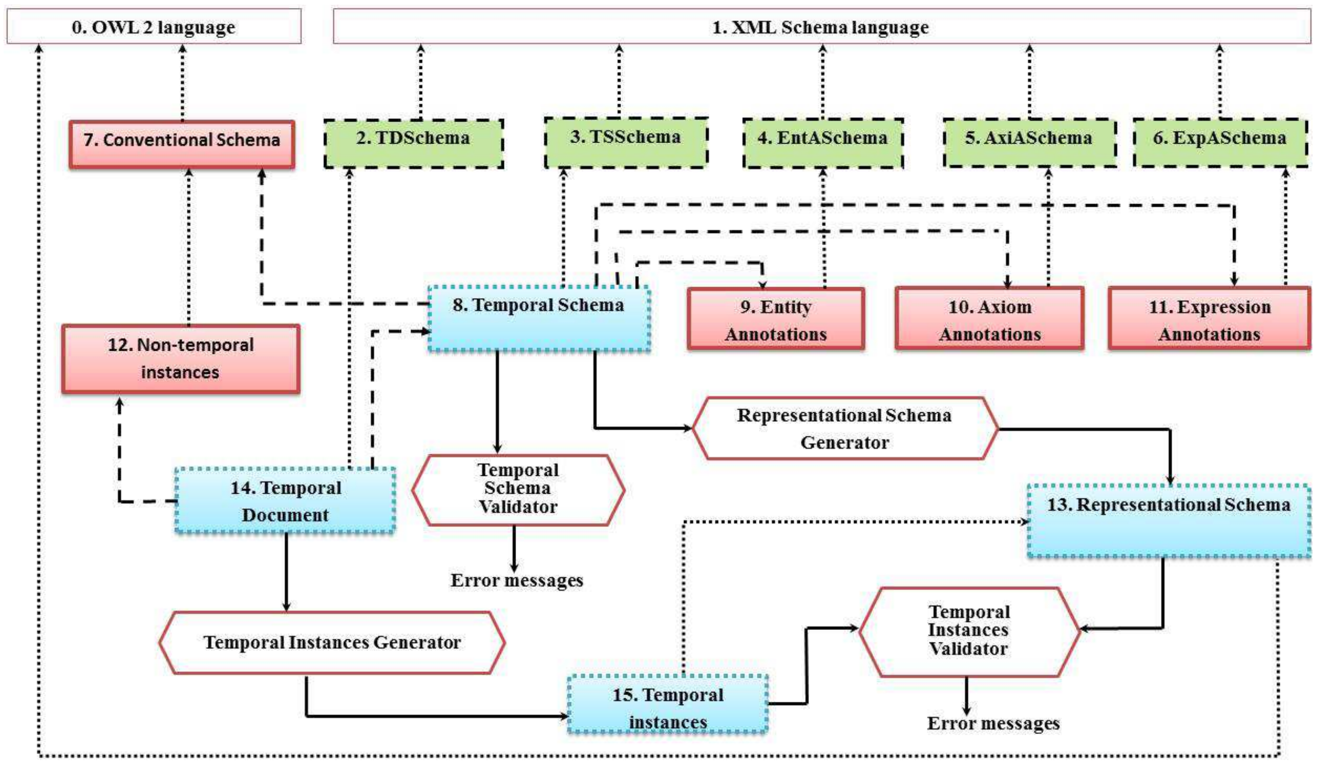

Figure 5. Overall architecture of τOWL. In the picture, rectangular boxes represent documents, hexagonal boxes represent tools, solid arrows denote Input/Output data flows, dotted arrows link documents to namespaces and dashed arrows stand for "references" relationships. Moreover, the meaning of the color and the border pattern of rectangular boxes is as follows: pink box with bold border for documents created/added by the KBA (7, 9, 10, 11 and 12), blue box with dotted border for documents automatically generated by the system (8, 13, 14, and 15), green box with dashed border for predefined documents making part of the framework (2, 3, 4, 5 and 6), and white box with thin border for reference documents created by the W3C (0 and 1).

Notice that EntASchema, AxiASchema, and ExpASchema, which all contain both logical and physical annotations, are XML Schemas [30]. The annotations associated to the same conventional schema can evolve independently. Any change to one of the three sets of annotations does not affect the two other sets.

Finally, when the KBA finishes annotating the conventional schema and asks the system to save his/her work, this latter creates the *temporal schema* (box 8) in order to provide the linking information between the conventional schema and its corresponding logical and physical annotations. The temporal schema is a standard XML document, which ties the conventional schema, the entity annotations, the axiom annotations, and the expression annotations together. In the τOWL framework, the temporal schema is the logical equivalent of the conventional OWL 2 schema in a non-temporal context. This document contains sub-elements that associate a series of conventional schema definitions with entity annotations, axiom annotations, and expression annotations, along with the time span during which the association was in effect. The schema for the temporal schema document is the XML Schema Definition document *TSSchema* (box 3).

To complete the figure in our temporal context, after creating the temporal schema, the system creates a *temporal document* (box 14) in order to link each conventional ontology instance document (box 12), which is valid to a conventional ontology schema (box 7), to its corresponding temporal ontology schema (box 8), and more precisely to its corresponding logical and physical annotations (which are referenced by the temporal schema). A temporal document is a standard XML document that maintains the evolution of a non-temporal ontology instance document over time, by recording all of the versions (or temporal slices) of the document with their corresponding timestamps and by specifying the temporal schema associated to these versions. This document contains sub-elements that associate a series of conventional ontology instance documents with logical and physical annotations (on entities, axioms, and expressions), along with the time span during which the association was in effect. Thus, the temporal document is very important for making easy the support of temporal queries working on past versions or dealing with changes between versions. The schema for the temporal document is the XML Schema Definition document *TDSchema* (box 2).

Notice that, whereas TDSchema (box 2), TSSchema (box 3), EntASchema (box 4), AxiASchema (box 5), and ExpASchema (box 6) have been developed by us, OWL 2 (box 0) and XML Schema (box 1) correspond to the standards endorsed by the W3C.

In a similar way to what happens in the τXSchema framework, the temporal schema document (box 8) is processed by the ***temporal schema validator*** tool in order to ensure that the logical and physical entity annotations, axiom annotations and expression annotations are (i) valid with respect to their corresponding schemas (i.e., EntASchema, AxiASchema, and ExpASchema, respectively), and (ii) consistent with the conventional schema. The temporal schema validator tool reports whether the temporal schema document is valid or invalid.

Once all the annotations are found to be consistent, the ***representational schema generator*** tool generates the *representational schema* (box 13) from the temporal schema (i.e., from the conventional schema and the logical and physical annotations); it is the result of transforming the conventional schema according to the requirements expressed through the different annotations. The representational schema becomes the schema for temporal instances (box 15). Temporal instances could be created in four ways:

i) automatically from the temporal document (box 14) (i.e., from *non-temporal ontology instances* (box 12) and the temporal ontology schema (box 8)), using the ***temporal instances generator*** tool (such an operation is called “squash” in the original τXSchema approach);
ii) automatically from instances stored in a knowledge base, i.e., as the result of a “temporal query” or a “temporal view”;
iii) automatically from a third-party tool, or
iv) manually (i.e., temporal instances are directly inserted by the KBA into the τOWL repository).

Moreover, temporal instances are validated against the representational schema through the ***temporal instances validator*** tool, which reports whether the temporal instances document (box 15) is valid or invalid.

Notice that the four mentioned tools (i.e., Temporal Schema Validator, Temporal Instances Validator, Representational Schema Generator, and Temporal Instances Generator) are under development. For example, the temporal instances validator tool is being implemented as a temporal extension of an existing conventional ontology instance validator.

A. Running example reprise

In order to show the functioning of the proposed approach, we continue in the following our motivating example of Section II.A, in order to show how management of temporal ontology document versions is dealt with in the τOWL approach.

On January 15, 2014, the KBA creates a conventional ontology schema (box 7), named “PersonSchema_V1.owl” (as in Figure 4), and a conventional ontology document (box 12), named “Persons_V1.rdf” (as in Figure 1), which is valid with respect to this schema. We assume that the KBA defines also a set of logical and physical annotations, associated to that conventional schema; they are stored in an ontology annotation document (boxes 9, 10, and 11) titled “PersonAnnotations_V1.xml” as shown in Figure 6.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<ontologyAnnotationSet>
  <logicalAnnotations>
    <item target=”/Person/nick”>
      <validTime kind=”state”
                content=”varying”
                existence=”constant”/>
    </item>
  </logicalAnnotations>
  <physicalAnnotations>
```

```
   <stamp target=”Person/nick”
          dataInclusion=”expandedVersion”>
     <stampkind timeDimension=”validTime”
              stampBounds=”extent”/>
   </stamp>
  </physicalAnnotations>
</ontologyAnnotationSet>
```

Figure 6. The annotation document on January 15, 2014.

After that, the system creates the temporal ontology schema (box 8) in Figure 7, which ties “PersonSchema_V1.owl” and “PersonAnnotations_V1.xml” together; this temporal schema is saved in an XML file titled “PersonTemporalSchema.xml”. Consequently, the system uses the temporal ontology schema of Figure 7 and the conventional ontology document in Figure 1 to create a temporal document (box 14) as in Figure 8, which lists both versions (i.e., temporal “slices”) of the conventional ontology documents with their associated timestamps. The squashed version (box 15) of this temporal document, which could be generated by the Temporal Instances Generator, is provided in Figure 9.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<temporalOntologySchema>
  <conventionalOntologySchema>
    <sliceSequence>
      <slice location=”PersonSchema_V1.owl”
              begin=”2014-01-15” />
    </sliceSequence>
  </conventionalOntologySchema>
  <ontologyAnnotationSet>
    <sliceSequence>
      <slice
        location=”PersonAnnotations_V1.xml”
        begin=”2014-01-15” />
    </sliceSequence>
  </ontologyAnnotationSet>
</temporalOntologySchema>
```

Figure 7. The temporal schema on January 15, 2014.

```
<?xml version=”1.0” encoding=”UTF-8”?>
  <td:temporalRoot
    temporalSchemaLocation=”PersonTemporalSchema.xml
” />
    <td:sliceSequence>
      <td:slice location=”Persons_V1.rdf”
               begin=”2014-01-15” />
    </td:sliceSequence>
  </td:temporalRoot>
```

Figure 8. The temporal document on January 15, 2014.

```
<foaf:Person rdf:ID="#Person1">
  <foaf:name>Nouredine Tounsi</foaf:name>
  <nick_RepItem>
    <nick_Version>
      <timestamp_ValidExtent
              begin=”2014-01-15” end=”now” />
      <foaf:nick>Nor</foaf:nick>
    </nick_Version>
  </nick_RepItem>
  <foaf:holdsAccount>
    <foaf:OnlineAccount
          rdf:about="https://www.facebook.com/
          Nouredine.Tounsi">
      <accountName_RepItem>
        <accountName_Version>
          <timestamp_ValidExtent
                 begin=”2014-01-15” end=”now” />
          <foaf:accountName>Nor_Tunsi
          </foaf:accountName>
        </accountName_Version>
      </accountName_RepItem>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
```

Figure 9. The squashed document correponding to the temporal document on January 15, 2014.

On February 08, 2014, the KBA updates the conventional ontology document “Persons_V1.rdf” as presented in Section II.A to produce a new conventional ontology document named “Persons_V2.rdf” (as in Figure 2). Since the conventional ontology schema (i.e., PersonSchema_V1.owl) and the ontology annotation document (i.e., PersonAnnotations_V1.xml) are not changed, the temporal ontology schema (i.e., PersonTemporalSchema.xml) is consequently not updated. However, the system updates the temporal document, in order to include the new slice of the new conventional ontology document, as shown in Figure 10. The squashed version of the updated temporal document is provided in Figure 11.

```
<?xml version=”1.0” encoding=”UTF-8”?>
  <td:temporalRoot
    temporalSchemaLocation=”PersonTemporalSchema.xml
” />
    <td:sliceSequence>
      <td:slice location=”Persons_V1.rdf”
               begin=”2014-01-15” />
      <td:slice location=”Persons_V2.rdf”
               begin=”2014-02-08” />
    </td:sliceSequence>
  </td:temporalRoot>
```

Figure 10. The temporal document on February 08, 2014.

```
<foaf:Person rdf:ID="#Person1">
  <foaf:name>Nouredine Tounsi</foaf:name>
  <nick_RepItem>
    <nick_Version>
      <timestamp_ValidExtent begin=”2014-01-15”
                             end=”2014-02-07” />
      <foaf:nick>Nor</foaf:nick>
    </nick_Version>
    <nick_Version>
      <timestamp_ValidExtent begin=”2014-02-08”
                             end=”now” />
      <foaf:nick>Nouri</foaf:nick>
    </nick_Version>
  </nick_RepItem>
  <foaf:holdsAccount>
    <foaf:OnlineAccount
          rdf:about="https://www.facebook.com/
          Nouredine.Tounsi">
      <accountName_RepItem>
        <accountName_Version>
          <timestamp_ValidExtent
                  begin=”2014-01-15”
                  end=”2014-02-07”/>
          <foaf:accountName>Nor_Tunsi
          </foaf:accountName>
      </accountName_Version>
      <accountName_Version>
       <timestamp_ValidExtent
                  begin="2014-02-08"
                  end="now" />
       <foaf:accountName>Nouri_Tunsi
       </foaf:accountName>
      </accountName_Version>
     </accountName_RepItem>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
```

Figure 11. The squashed document correponding to the temporal document on February 08, 2014.

Obviously, each one of the squashed documents (see Figure 9 and Figure 11) should conform to a particular schema, which is the representational schema (box 13), which is generated (by the Representational Schema Generator) from the temporal schema shown in Figure 7.

The example will be completed in Section V.D, after that the management of schema changes has been introduced.

IV. Our Approach to Schema Versioning in the τOWL Framework

In this section, we describe how τOWL conventional schema and τOWL logical and physical annotations can be versioned in our approach.

The first step of a schema versioning sequence is the creation of a first schema version: the KBA creates a conventional ontology schema (i.e., an OWL 2 file) and annotates it with some logical and physical annotations in an independent document (which is stored as an XML file), through, for instance, a graphical interface. Consequently, the system creates the temporal ontology schema (also stored as an XML file) that ties together the conventional schema and the annotations.

In further steps of the versioning sequence, applied when necessary, the KBA can independently change the conventional ontology schema, the logical or the physical ontology annotations.

Changing the conventional ontology schema leads to a new version of it. Similarly, changing logical or physical ontology annotations leads to a new version of the whole ontology annotation document. Therefore, the temporal ontology schema is implicitly and automatically updated by the system after each change of the conventional schema or of the annotation document.

Schema change operations performed by the KBA are high-level, since they are usually conceived having in mind high-level real-world specifications. Each of these high-level schema change operations is then mapped onto a sequence of low-level schema change operations (or schema change primitives). The mapping is performed by a schema change processor.

Each high-level change can be expressed as a sequence of change primitives. Thus, the consistency of the resulting conventional ontology schema (respectively, the resulting ontology annotation document or the resulting temporal ontology schema) is always guaranteed, if change primitives preserve the conventional ontology schema (respectively, the ontology annotation document or the temporal ontology schema) consistency.

Notice that in our approach, like in [15], the temporal schema, which ties the conventional schema and the annotations together, is not "explicitly" versioned; for each conventional schema (i.e., all the versions of this schema) and its associated annotation document (i.e., all the versions of this document), there is always one XML document that represents the temporal schema, which is updated when the conventional schema and/or the annotation document are changed. In fact, in the τOWL framework, the temporal schema is instrumental to support versioning of anything can change in the managed Semantic Web repository. As a consequence, by its nature, the temporal schema comes out "implicitly" versioned (i.e., all versions of a temporal schema document are stored within this document; the version of a temporal schema, valid at any given time Tx, could be extracted from that schema by removing all the <slice ... begin=Ty/> elements where Ty>Tx). Thus, we think that other kinds of versioning of the temporal schema are neither necessary nor could be meaningfully put at user's disposal (without getting out of the τOWL framework).

Notice also that neither conventional schema versioning nor annotation versioning lead automatically to proliferation of schema versions. The creation of a new conventional schema version (or of a new annotation document version) is anyway a seldom task during the Semantic Web repository lifetime, which can only be performed by an administrator of this repository. This task may consist of dozens of schema change operations, which are grouped together in the same single transaction.

V. Primitives for Changing Conventional Schema and Temporal Schema in the τOWL Framework

In this section, we first present our design principles and then introduce our proposed change primitives. We start by providing change primitives acting on conventional schema in τOWL and then we provide primitives for changing the temporal ontology schema. We have individuated change primitives (i.e., non-further decomposable in terms of the other ones), which make up a complete set of changes (i.e., such that any possible complex change can be defined via a combination/sequence of them). For each change primitive, we describe its arguments and its operational semantics. Finally, we give an example that illustrates the use of these primitives for versioning of τOWL conventional schema.

A. Design principles

The definition of the primitives will obey the following principles and conventions:

1) all primitives must work on a well-formed and valid Conventional Ontology Schema (COS) (or on the Temporal Ontology Schema (TOS)), that is, primitives must have a well-formed and valid COS (or TOS) as input and produce a well-formed and valid COS (or TOS) as output;

2) all primitives need to work on an OWL 2 file (or an XML file) storing the COS (or TOS), whose name must be supplied as argument;

3) for all primitives, arguments that are used to identify the object on which the primitive works are in the first place of the argument list;

4) primitives adding elements with possibly optional attributes have the values for all the attributes as arguments; empty places in the argument list stand for unspecified optional attributes;

5) for primitives changing elements, values are specified only for attributes that are changed; the value "unchanged" means that the corresponding attribute is not updated; an empty place in the argument list means that the corresponding attribute receives a nil value.

The lists of operations in the subsections that follow are the applications of the design principles presented above.

B. *Primitives for changing conventional schema*

Based on the OWL 2 ontology definition we adopted in Section III (e.g., assuming the signature O = {E, A, Exp}), we define a complete set of primitives for changing a conventional ontology schema, composed of twenty-eight operations. The idea is that each primitive deals with an OWL 2 ontology component (e.g., a class, a data property, an object property), by creating, removing or modifying such a component. For each primitive change, we describe its arguments and its operational semantics. Obviously, each primitive change has an effect on the COS. We do not present in this paper the effects of all primitive changes. We give only the effect of some selected primitive changes.

We have organized the proposed primitives into eight categories: (i) primitives acting on the whole COS (in the sub-section V.B.1), (ii) primitives acting on a class (in the sub-section V.B.2), (iii) primitives acting on a data property (in the sub-section V.B.3), (iv) primitives acting on an object property (in the sub-section V.B.4), (v) primitives acting on an annotation property (in the sub-section V.B.5), (vi) primitives acting on an entity axiom (in the sub-section V.B.6), (vii) primitives acting on a key axiom (in the sub-section V.B.7), and (viii) primitives acting on an entity expression (in the sub-section V.B.8).

1) Primitives acting on the whole COS

We have only three primitives:

- **CreateConventionalOntologySchema(COS.owl)**

It produces a valid empty OWL 2 file. According to the second design principle, the argument is the name of the OWL 2 file where the new COS is stored.

Notice also that the name of this file is the name of the ontology (e.g., Author, Paper, and Conference).

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about="" />
</rdf:RDF>
```

- **RenameConventionalOntologySchema(oldCOS, newCOS)**

It changes the name of a COS from "oldCOS" to "newCOS" (or, it changes the name of an ontology from "oldOntoName" to "newOntoName").

- **DropConventionalOntologySchema(COS.owl)**

It removes the COS.owl file from disk, with the constraint that the argument represents an empty COS (i.e., like the one above initially created by the CreateConventionalOntologySchema primitive). Any other contents must have been removed before.

2) Primitives acting on a class

We have defined three primitives:

- **AddClass(COS.owl, className)**

It adds a new class having the name "className" to the COS.

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about="">
    <owl:Class rdf:about="className"/>
  </owl:Ontology>
</rdf:RDF>
```

- **RenameClass(COS.owl, oldClassName, newClassName)**

It changes the name of a class from "oldClassName" to "newClassName", in the COS.

- **DropClass(COS.owl, className)**

It removes the class having the name "className" from the COS.

3) Primitives acting on a data property

We have defined five primitives:

- **AddDataProperty(COS.owl, className, DataPropertyName, DataPropertyType)**

It adds a new data property having the name "DataPropertyName" and the type "DataPropertyType" to the class "className", in the COS.

Notice that the "className" and the "DataPropertyType" are considered as the "DataPropertyDomain" and the "DataPropertyRange", respectively.

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about="">
    <owl:Class rdf:about="className"/>
    <owl:DatatypeProperty
rdf:about="DataPropertyName">
      <rdfs:domain rdf:resource="className"/>
      <rdfs:range rdf:resource="DataPropertyType"/>
    </owl:DatatypeProperty>
  </owl:Ontology>
</rdf:RDF>
```

- **DropDataProperty(COS.owl, className, DataPropertyName)**

It removes the data property having the name "DataPropertyName" from the class "className", in the COS.

- **RenameDataProperty(COS.owl, className, oldDataPropertyName, newDataPropertyName)**

It changes the name of a data property from "oldDataPropertyName" to "newDataPropertyName" in the class “className”, in the COS.

- **ChangeDataPropertyDomain(COS.owl, className, DataPropertyName, newDataPropertyDomain)**

It replaces the domain (or class) “className” of the data property “DataPropertyName” with a new domain “newDataPropertyDomain”, in the COS.

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
 <owl:Ontology rdf:about="">
   <owl:Class rdf:about="newDataPropertyDomain"/>
   <owl:DatatypeProperty
rdf:about="DataPropertyName">
    <rdfs:domain
rdf:resource="newDataPropertyDomain"/>
    <rdfs:range rdf:resource="DataPropertyType"/>
   </owl:DatatypeProperty>
 </owl:Ontology>
</rdf:RDF>
```

- **ChangeDataPropertyRange(COS.owl, className, DataPropertyName, oldDataPropertyRange, newDataPropertyRange)**

It replaces the range (or type) “oldDataPropertyRange” of the data property “DataPropertyName” of the class “className” with a new range “newDataPropertyRange”, in the COS.

4) Primitives acting on an object property

We have defined five primitives:

- **AddObjectProperty(COS.owl, ObjectPropertyName, ObjectPropertyDomain, ObjectPropertyRange)**

It creates an object property (a relation) having the name “ObjectPropertyName” between a source class “ObjectPropertyDomain” and a target class “ObjectPropertyRange”, in the COS.

The effect of such a primitive, that is, the contents of the COS.owl file after its application, is as follows:

```
<rdf:RDF>
 <owl:Ontology rdf:about="">
   …
   <owl:ObjectProperty
rdf:about="ObjectPropertyName">
    <rdfs:domain
rdf:resource="ObjectPropertyDomain"/>
    <rdfs:range
rdf:resource="ObjectPropertyRange"/>
   </owl:ObjectProperty>
 </owl:Ontology>
</rdf:RDF>
```

- **DropObjectProperty(COS.owl, ObjectPropertyName)**

It removes the object property “ObjectPropertyName” from the COS.

- **RenameObjectProperty(COS.owl, oldObjectPropertyName, newObjectPropertyName)**

It changes the name of an object property from “oldObjectPropertyName” to “newObjectPropertyName”, in the COS.

- **ChangeObjectPropertyDomain(COS.owl, ObjectPropertyName, oldObjectPropertyDomain, newObjectPropertyDomain)**

It replaces the domain “oldObjectPropertyDomain” of the object property “ObjectPropertyName” with a new domain “newObjectPropertyDomain”, in the COS.

- **ChangeObjectPropertyRange(COS.owl, ObjectPropertyName, oldObjectPropertyRange, newObjectPropertyRange)**

It replaces the range “oldObjectPropertyRange” of the object property “ObjectPropertyName” with a new range “newObjectPropertyRange”, in the COS.

5) Primitives acting on an annotation property

We have defined three primitives:

- **AddAnnotationProperty(COS.owl, propertyType, propertyName, annotationProperty)**

It defines a new annotation property “annotationProperty” on the propertyType (i.e., Class, DataProperty, ObjectProperty, EntityAxiom, or KeyAxiom) named “propertyName”, in the COS.

- **DropAnnotationProperty(COS.owl, propertyType, propertyName, annotationProperty)**

It removes the annotation property “annotationProperty” defined on the propertyType (i.e., Class, DataProperty, ObjectProperty, EntityAxiom, or KeyAxiom) named “propertyName”, in the COS.

- **ChangeAnnotationProperty(COS.owl, propertyType, propertyName, oldAnnotationProperty, newAnnotationProperty)**

It replaces the annotation property “oldAnnotationProperty” defined on the propertyType (i.e., Class, DataProperty, ObjectProperty, EntityAxiom, or KeyAxiom) named “propertyName”, in the COS, with a new annotation property “newAnnotationProperty”.

6) Primitives acting on an entity axiom

We have defined three primitives:

- **AddEntityAxiom(COS.owl, entityType, entityName, entityAxiom)**

It defines a new entity axiom “entityAxiom” on the entityType (i.e., Class, DataProperty, ObjectProperty, or AnnotationProperty) named “entityName”, in the COS.

- **DropEntityAxiom(COS.owl, entityType, entityName, entityAxiom)**

It removes the entity axiom “entityAxiom” defined on the entityType (i.e., Class, DataProperty, ObjectProperty, or AnnotationProperty) named “entityName”, in the COS.

- **ChangeEntityAxiom(COS.owl, entityType, entityName, oldEntityAxiom, newEntityAxiom)**

It replaces the entity axiom “oldEntityAxiom” defined on the entityType (i.e., Class, DataProperty, ObjectProperty, or AnnotationProperty) named “entityName”, in the COS, with a new entity axiom “newEntityAxiom”.

7) Primitives acting on a key axiom

We have defined also three primitives:

- **AddKeyAxiom(COS.owl, className, keyAxiom)**

It defines a new key axiom “keyAxiom” on the class “className”, in the COS.

- **DropKeyAxiom(COS.owl, className, keyAxiom)**

It removes the key axiom “keyAxiom” defined on the class “className”, in the COS.

- **ChangeKeyAxiom(COS.owl, className, oldKeyAxiom, newKeyAxiom)**

It replaces the key axiom “oldKeyAxiom” defined on the class “className” in the COS, with a new key axiom “newKeyAxiom”.

8) Primitives acting on an entity expression

We have only three primitives:

- **AddEntityExpression(COS.owl, entityType, entityName, entityExpression)**

It defines a new entity expression “entityExpression” on the entityType (i.e., Class, DataProperty, or ObjectProperty) named “entityName”, in the COS.

- **DropEntityExpression(COS.owl, entityType, entityName, entityExpression)**

It removes the entity expression “entityExpression” defined on the entityType (i.e., Class, DataProperty, or ObjectProperty) named “entityName”, in the COS.

- **ChangeEntityExpression(COS.owl, entityType, entityName, oldEntityExpression, newEntityExpression)**

It replaces the entity expression “oldEntityExpression” defined on the entityType (i.e., Class, DataProperty, or ObjectProperty) named “entityName”, in the COS, with a new entity expression “newEntityExpression”.

C. *Primitives for changing the temporal ontology schema*

Changing the temporal ontology schema is a task that must be done within the same transaction that changes the corresponding conventional ontology schema and/or the ontology annotation document. We also propose in this sub-section a complete set of primitives acting on a temporal ontology schema (their total number is four). For each primitive, we provide specifications for its actions and explanation of its parameters. We also present the effects of some of them. These primitives are as follows:

- **CreateTemporalOntologySchema(TOS.xml)**

It produces a valid empty TOS. According to the second design principle, the argument is the name of the XML file where the new TOS is stored.

The effect of the CreateTemporalOntologySchema(TOS.xml) primitive, that is, the contents of the COS.xml file after its application, is as follows:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<temporalOntologySchema/>
```

- **DropTemporalOntologySchema(TOS.xml)**

It removes the TOS.xml file from disk, with the constraint that the argument represents an empty TOS (i.e., like the one above initially created by the CreateTemporalOntologySchema primitive). Any other contents must have been removed before.

- **AddSlice(TOS.xml, toWat, sourceSlice, targetSlice)**

It adds the <slice/> element with specified sourceSlice and targetSlice to the toWhat (i.e., <conventionalOntologySchema/> or <ontologyAnnotationSet/>) container.

- The sourceSlice parameter could be:

a) The keyword empty; in this case, the resource pointed by targetSlice is initialized to an empty conventional ontology schema or ontology annotation document according to the toWhat value.

b) The keyword current; in this case, the resource pointed by targetSlice is initialized with a copy of the current conventionalOntologySchema or ontologyAnnotationSet resource (according to toWhat), whose location is found in the TOS.xml temporal schema file by choosing the slice with the maximum value of begin in the corresponding sliceSequence (note: after the creation of the first schema version, this is the normal case).

c) A specified file name (URL): in this case, a copy of the specified resource is renamed as targetSlice and used as the new location (e.g., this case is used to create a new conventional ontology schema version from an already existing OWL 2 file, which could be quite common when creating the first schema version but can be used also later for reuse purpose and/or integrating independently developed schemata into a τOWL framework).

- The targetSlice parameter is the value assigned to the location attribute of <slice/> and must not correspond to the URL of any already existing OWL 2 file/resource.

For example, the effects of the AddSlice(“TOS.xml”, conventionalOntologySchema, empty, “COS_V1.owl”) primitive are described in the following:

i) The contents of the TS.xml file is updated as follows (the transaction time associated to the execution of the transaction that includes this primitive is March 01, 2012, which is used as value of begin in the <slice/> element):

```
<?xml version=”1.0” encoding=”UTF-8”?>
<temporalOntologySchema>
  <conventionalOntologySchema>
    <sliceSequence>
      <slice location=”COS_V1.owl”
             begin=”2012-03-01” />
    </sliceSequence>
  </conventionalOntologySchema>
</temporalOntologySchema>
```

ii) A new empty conventional ontology schema, titled “COS_V1.owl”, is created as follows:

```
<rdf:RDF>
  <owl:Ontology rdf:about=”” />
</rdf:RDF>
```

- **DropSlice(TOS.xml, fromWat, targetSlice)**

It removes the <slice/> element with specified targetSlice from the fromWhat (i.e., <conventionalOntologySchema/> or <ontologyAnnotationSet/>) container.

D. *Running example conclusion*

Let us resume the example started in Section II.A. Suppose that on July 18, 2014, the KBA decides to make some changes to the first version of the conventional ontology schema, in order to meet some changes in the code of the application that exploit such an ontology schema. These changes are as follows:

– define an irreflexive relationship (or object property), named "childOf", on the class "Person";

– create two new classes, named "Man" and "Woman", which inherit from the class "Person";

– define a symmetric relationship, named "hasSpouse", between the class "Man" and the class "Woman";

– specify a relationship, named "hasWife", between the class "Man" and the class "Woman". This relationship inherits from the relationship "hasSpouse";

– change the name of the property (or data property) "name" of the class "Person" to "fullName";

– add a new property, named "age" and having the XSD type "nonNegativeInteger", to the class "Person";

– specify an expression on the relationship "holdsAccount", which indicates that each person must have at least one online account.

The second version of the conventional ontology schema and the second version of each one the two conventional ontology instance documents are shown in Figure 12, Figure 13, and Figure 14, respectively. The temporal ontology schema is also updated by adding a new slice related to this new version of the conventional ontology schema, as shown in Figure 15. Moreover, the temporal document is updated, in order to include two new slices corresponding to the two new conventional ontology instance documents, as shown in Figure 16. The squashed version of the updated temporal document that consequently can be generated by the Temporal Instances Generator tool is similar to documents provided in Figure 9 and Figure 11. Notice that changes are presented in red, in Figures 12-16.

```
<rdf:RDF>
  <owl:Ontology
rdf:about="http://purl.org/az/foaf#">
    <owl:Class rdf:about="Person"/>
    <owl:Class rdf:about="OnlineAccount"/>
    <owl:Class rdf:about="Man"/>
    <owl:Class rdf:about="Woman"/>
    <owl:DatatypeProperty rdf:about="accountName">
      <rdfs:domain rdf:resource="Person"/>
      <rdfs:range rdf:resource=
      "http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="nick">
      <rdfs:domain rdf:resource="Person"/>
      <rdfs:range rdf:resource=
      "http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="fullName">
      <rdfs:domain rdf:resource="Person"/>
      <rdfs:range rdf:resource=
      "http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="age">
      <rdfs:domain rdf:resource="Person"/>
      <rdfs:range rdf:resource=
      "http://www.w3.org/2001/XMLSchema#nonNegativeI
nteger"/>
    </owl:DatatypeProperty>
```

```
    <owl:ObjectProperty rdf:about="holdsAccount">
      <rdfs:domain rdf:resource="Person"/>
      <rdfs:range rdf:resource="OnlineAccount"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="hasSpouse">
      <rdfs:domain rdf:resource="Man"/>
      <rdfs:range rdf:resource="Woman"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="hasWife">
      <rdfs:domain rdf:resource="Man"/>
      <rdfs:range rdf:resource="Woman"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="childOf">
      <rdfs:domain rdf:resource="Person"/>
      <rdfs:range rdf:resource="Person"/>
    </owl:ObjectProperty>
    <owl:Class rdf:about="Man">
      <rdfs:subClassOf rdf:resource="Person"/>
    </owl:Class>
    <owl:Class rdf:about="Woman">
      <rdfs:subClassOf rdf:resource="Person"/>
    </owl:Class>
    <owl:SymmetricProperty rdf:about="hasSpouse"/>
    <owl:IrreflexiveProperty rdf:about="childOf"/>
    <owl:ObjectProperty rdf:about="hasWife">
      <rdfs:subPropertyOf rdf:resource="hasSpouse"/>
    </owl:ObjectProperty>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#holdsAccount"/>
      <owl:minCardinality rdf:datatype=
      "http://www.w3.org/2001/XMLSchema#nonNegativeI
nteger">1
      </owl:minCardinality>
    </owl:Restriction>
    …
  </owl:Ontology>
</rdf:RDF>
```

Figure 12. Second version of the conventional ontology schema (PersonSchema_V2.owl), on July 18, 2014.

```
…
<foaf:Person rdf:ID="#Person1">
  <foaf:fullName>Nouredine Tounsi</foaf:fullName>
  <foaf:nick>Nor</foaf:nick>
  <age/>
  <childOf/>
  <hasSpouse/>
  <hasWife/>
  <foaf:holdsAccount>
    <foaf:OnlineAccount rdf:about=
    "https://www.facebook.com/Nouredine.Tounsi">
      <foaf:accountName>Nor_Tunsi</foaf:accountName>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
…
```

Figure 13. "Persons_V3.rdf": the second version of the conventional ontology instance document "Persons_V1.rdf", on July 18, 2014.

```
…
<foaf:Man rdf:ID="#Person1">
  <foaf:fullName>Nouredine Tounsi</foaf:fullName>
  <foaf:nick>Nouri</foaf:nick>
  <age/>
  <childOf/>
  <hasSpouse/>
  <hasWife/>
  <foaf:holdsAccount>
    <foaf:OnlineAccount rdf:about=
       "https://www.facebook.com/Nouredine.Tounsi">
      <foaf:accountName>Nouri_Tunsi
      </foaf:accountName>
    </foaf:OnlineAccount>
  </foaf:holdsAccount>
</foaf:Person>
…
```

Figure 14. “Persons_V4.rdf”: the second version of the conventional ontology instance document “Persons_V2.rdf”, on July 18, 2014.

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalOntologySchema>
  <conventionalOntologySchema>
    <sliceSequence>
      <slice location="PersonSchema_V1.owl"
begin="2014-01-15" />
      <slice location="PersonSchema_V2.owl"
begin="2014-07-18" />
    </sliceSequence>
  </conventionalOntologySchema>
  <ontologyAnnotationSet>
    <sliceSequence>
      <slice location="PersonAnnotations_V1.xml"
begin="2014-01-15" />
    </sliceSequence>
  </ontologyAnnotationSet>
</temporalOntologySchema>
```

Figure 15. The temporal ontology schema (PersonTemporalSchema.xml), on July 18, 2014.

```
<?xml version="1.0" encoding="UTF-8"?>
  <td:temporalRoot
    temporalSchemaLocation="PersonTemporalSchema.xml
" />
    <td:sliceSequence>
      <td:slice location="Persons_V1.rdf"
                begin="2014-01-15" />
      <td:slice location="Persons_V2.rdf"
                begin="2014-02-08" />
      <td:slice location="Persons_V3.rdf"
                begin="2014-07-18" />
      <td:slice location="Persons_V4.rdf"
                begin="2014-07-18" />
    </td:sliceSequence>
  </td:temporalRoot>
```

Figure 16. The temporal document (PersonTemporalDocument.xml) on July 18, 2014.

The transaction listed in the following contains the sequence of primitives that have been performed on the temporal ontology schema (PersonTemporalSchema.xml, in Figure 7), on the first version of the conventional ontology schema (PersonSchema_V1.owl in Figure 4), on the first version of the conventional ontology instance document (Persons_V1.rdf in Figure 1) and on the second version of the conventional ontology instance document (Persons_V2.rdf in Figure 2), in order to update the temporal ontology schema (see Figure 15) and the temporal document (see Figure 16) and to produce the second version of the conventional ontology schema, named “PersonSchema_V2.owl” (see Figure 12), and the third and the fourth versions of the conventional ontology instance document, named “Persons_V3.rdf” (see Figure 13) and “Persons_V4.rdf” (see Figure 14), respectively, which are valid with respect to “PersonSchema_V2.owl”:

```
Begin Transaction
(i) AddSlice("PersonTemporalSchema.xml",
   conventionalOntologySchema, current,
   "PersonSchema_V2.owl")
(ii) AddObjectProperty("PersonSchema_V2.owl",
   "childOf", "Person", "Person")
(iii) AddEntityAxiom("PersonSchema_V2.owl",
   ObjectProperty, "childOf",
   "IrreflexiveProperty")
(iv) AddClass("PersonSchema_V2.owl", "Man")
(v) AddClass("PersonSchema_V2.owl", "Woman")
(vi) AddEntityAxiom("PersonSchema_V2.owl", Class,
   "Man", "subClassOf(Person)")
(vii) AddEntityAxiom("PersonSchema_V2.owl", Class,
   "Woman", "subClassOf("Person")")
(viii) AddObjectProperty("PersonSchema_V2.owl",
   "hasSpouse", "Man", "Woman")
(ix) AddEntityAxiom("PersonSchema_V2.owl",
   ObjectProperty, "hasSpouse",
   "SymmetricProperty")
(x) AddObjectProperty("PersonSchema_V2.owl",
   "hasWife", "Man", "Woman")
(xi) AddEntityAxiom("PersonSchema_V2.owl",
   ObjectProperty, "hasWife",
   "subObjectPropertyOf("hasSpouse")")
(xii) RenameDataProperty("PersonSchema_V2.owl",
   "Person", "name", "fullName")
(xiii) AddDataProperty("PersonSchema_V2.owl",
   "Person", "age",
   "http://www.w3.org/2001/XMLSchema#nonNegativeIn
   teger")
(xiv) AddEntityExpression("PersonSchema_V2.owl",
   ObjectProperty, "holdsAccount",
   "minCardinality(1)")
Commit
```

The transaction time associated to the execution of the transaction above is July 18, 2014, which is used as value of the attribute “begin” of the new <slice/> element, corresponding to the new conventional ontology schema version, in the temporal ontology schema file.

Notice that on July 18, 2014, our multiversion τOWL framework is thus composed of two successive versions of the conventional ontology schema (shown in Figure 4 and Figure 12, respectively), four versions of the conventional ontology instance documents (shown in Figure 1, Figure 2, Figure 13, and Figure 14, respectively), one version of the ontology annotation document (shown in Figure 6), the temporal document (shown in Figure 16) and the temporal ontology schema (shown in Figure 15).

Notice also that “Persons_V3.rdf” (shown in Figure 13) and “Persons_V4.rdf” (shown in Figure 14) are the results of schema change propagation (i.e., the effects of schema changes on instances), in order to adapt all existing instances, stored in “Persons_V1.rdf” (shown in Figure 1) and “Persons_V2.rdf” (shown in Figure 2), to the new schema version “PersonSchema_V2.owl” (shown in Figure 12). In fact, after creating "Persons_V3.rdf" as a copy of "Persons_V1.rdf" and "Persons_V4.rdf" as a copy of "Persons_V2.rdf", the two following XQuery Update Facility [31] statements could be executed on "Persons_V3.rdf" and "Persons_V4.rdf", respectively, to achieve the purpose:

```
for $p in fn:doc("Persons_V3.rdf")//foaf:Person
return {
   rename node $p/foaf:name as "foaf:fullName",
   insert node <age/> after $p/foaf:nick,
   insert node <childOf/> after $p/age,
   insert node <hasSpouse/> after $p/childOf,
   insert node <hasWife/> after $p/hasSpouse
}
for $p in fn:doc("Persons_V4.rdf")//foaf:Person
return {
   rename node $p/foaf:name as "foaf:fullName",
   insert node <age/> after $p/foaf:nick,
   insert node <childOf/> after $p/age,
   insert node <hasSpouse/> after $p/childOf,
   insert node <hasWife/> after $p/hasSpouse
}
```

These statements are derived, in an automatic and transparent way, by the system as a part of the semantics of the schema change primitives. They are not part of what the KBA puts in his/her schema change transaction, but it is the system to generate and add them to the transaction that is actually executed.

VI. RELATED WORK DISCUSSION

In the literature, there are several proposals that deal with managing temporal aspects in ontologies or Semantic Web. OWL-Time (formerly DAML-Time) [32] is a temporal ontology that has been developed for describing the temporal content of Web pages and the temporal properties of Web services. Excepting language constructs for representing time in ontologies, mechanisms for representing evolution of concepts (e.g., events) over time are absent. Furthermore, temporal relations cannot be expressed directly in OWL, since they are ternary (i.e., properties of objects that change in time involve also a temporal value in addition to the object and the subject); representing such temporal relations in OWL requires appropriate methods (e.g., 4D-fluents [33]). Our approach allows a KBA to represent (i) evolution of concepts over time, and (ii) temporal relations.

In [34], the authors present the annotation features of OWL 2 by showing that it allows for annotations on ontologies, entities, anonymous individuals, axioms (e.g., giving information about who asserted an axiom or when), and annotations themselves. In our work, we took another direction from using OWL 2 annotation features because we rather wanted to exploit the power of the τXSchema approach (e.g., including the exploitation of a τXSchema-like underlying infrastructure).

Time dimension(s) are explicitly added to Semantic Web languages and formalisms (e.g., RDF, OWL and SPARQL [35]) in order to represent time in semantic annotations, to build temporal ontologies and to support temporal querying and reasoning. An annotated bibliography of previous work in this area is presented in [13], and a survey on the models and query languages for temporally annotated RDF is provided in [36]. In particular, in the literature, there are various contributions that propose to represent temporal data in the Semantic Web.

Gutiérrez et al. [37] presented a comprehensive framework to incorporate temporal reasoning into RDF, yielding temporal RDF graphs. They define a syntactic notion of temporal RDF graphs. A powerful system, called CHRONOS, for reasoning over temporal information in OWL ontologies is presented in [38]. Since qualitative representations are very common in natural language expressions such as in free text or speech and can be proven to be valuable in the Semantic Web, the authors choose to represent both qualitative temporal (i.e., information whose temporal extents are unknown such as “before”, “after” for temporal relations) and quantitative information (i.e., where temporal information is defined precisely, e.g., using dates). The CHRONOS reasoner can be applied to temporal relations in order to infer implied relations and to detect inconsistencies while retaining soundness, completeness and tractability over the supported relations set. The paper [39] proposes a logic-based approach to introduce valid-time into RDFS and OWL 2 languages. An extension of SPARQL that can be used to query temporal RDF(S) and OWL 2 is also presented. Moreover, the author describes a general query evaluation algorithm that can be used with all entailment relations used in the Semantic Web. Finally, he presents two optimizations of the algorithm that are applicable to entailment relations characterized by a set of deterministic rules, such RDF(S) and OWL 2 RL/RDF Entailment. As opposed to Gutiérrez et al. [37], Anagnostopoulos et al. [38] and Motik [39], in our present approach, we are not interested in temporal (or spatio-temporal) reasoning.

Two complementary and alternative proposals for modeling temporally changing information in OWL are proposed in [40]. They are based on the perdurantist theory and benefit from results coming from the discipline of Formal Ontology, in order to restrict the appropriate use of the proposed frameworks. In the first proposal, the authors combine the perdurantist worm view with the notion of individual concepts for formulating a conceptual structure that allows one to separate, from the information that define all the individuals, the information concerning those that can possibly change. In the second proposal, they extend the first proposal with the distinction between objects and moments and the notion of qua individuals, where a qua individual is the way an object participates in a certain relation.

Differently from Zamborlini et al. [40], our approach does not deal with modeling of time inside the ontology, but just aims at supporting temporal versioning.

O'Connor et al. [41] present a methodology and a set of tools for representing and querying temporal information in OWL ontologies. Their approach uses a lightweight temporal model to encode the temporal dimension of data. It also uses the OWL-based Semantic Web Rule Language (SWRL) and the SWRL-based OWL query language (SQWRL) to reason with and query the temporal information represented using the proposed model. By now, our approach does not support temporally-aware semantic rules.

The authors of [42] propose a new language, called temporal OWL (tOWL), which is an extension of the Ontology Web Language Description Logics (OWL-DL) to the temporal aspect. It enables the representation of time and change in dynamic domains. Through a layered approach, they introduce three extensions: (i) Concrete Domains, which allow the representation of restrictions using concrete domain binary predicates, (ii) Temporal Representation, which introduces timepoints, relations between timepoints, intervals, and Allen's 13 interval relations [43] into the language, and (iii) TimeSlices/Fluents, which implement a perdurantist view on individuals and enable the representation of complex temporal aspects such as process state transitions. The main purpose of our approach is to support past ontology versions, to be accessed via time-slice queries. We think that supporting temporal ontology versions is very interesting for several purposes and in different areas. The problem of not having temporal versions is that, e.g., if we have now to investigate on someone having put some illegal material on Facebook last week, we want to be able to individuate the account details even if they have been changed thereafter.

As far as ontology schema evolution and versioning problems are concerned, we can find also several studies which have dealt with them. In general, we could summarize them under the three following groups of issues taken into account:

– modeling, implementing, and detecting changes in ontologies [44][45][46][47][48];

– preserving the consistency of evolving ontologies [49][50][51][52];

– ontology versioning support [53][54][55][56][57][58][59][60][61].

Our approach belongs to the last set of contributions. In [53], the authors consider the notion of context as an abstraction mechanism to deal with multi-representation ontologies (contextual ontologies). A formal representation language based on modal description logics is proposed to comply with the requirements of multiple perspectives of domain ontology.

Bouquet et al. [54] show how ontologies can be contextualized, by proposing Context OWL (C-OWL), a language whose syntax and semantics have been obtained by extending the OWL syntax and semantics to allow for the representation of contextual ontologies. Notice that an ontology is said to be contextualized when its contents are kept local, and, therefore, not shared with other ontologies, and mapped with the contents of other ontologies via explicit (context) mappings.

Heflin et al. [55] show that the Semantic Web needs a formal semantics for the various kinds of links between ontologies and other documents, and then provide a model theoretic semantics that takes into account ontology extension and ontology versioning.

Völkel et al. [56] present an RDF-centric versioning approach and an implementation called SemVersion. The proposed approach separates the management aspects from the versioning core functionality. SemVersion provides structural and semantic versioning for RDF models and RDF-based ontology languages like RDFS, considering blank node enrichment as a technique to identify the blank nodes in the versioned models.

Bedi et al. [57] introduce an approach that combines the concepts of temporal frame and slot versioning with the ontology to create temporal tagged ontologies with embedded versioning. The authors also propose to enhance the existing OWL to enable the creation of temporal tagged OWL ontologies: two new tags, "rdf:Validity" and "rdf:Timestamp", are introduced and a scheme is presented for the value of the "rdf:Id" and "rdf:Resource" tags to make the temporal tagged ontologies consistent with the non-temporal ontologies.

Kondylakis et al. [61] propose a solution that allows query answering in data integration systems under evolving ontologies without mapping redefinition. This is achieved by rewriting queries among ontology versions and then forwarding them to the underlying data integration systems to be answered.

The works that are more strictly related with our approach are [58], [59], and [60]. Grandi [58] provides a multi-temporal RDF database model; a database consists in a set of RDF triples timestamped along the valid and/or transaction time axes. The data model is equipped with manipulation operations which allow the KBA to maintain a multi-temporal RDF database in order to manage temporal versions of an ontology. Grandi et al. [59] introduce "The Valid Ontology", a framework to represent and store multiple temporal versions of an ontology in a compact temporal XML format and efficiently extract ontology snapshots from the multiversion XML document via a temporal XML processor. Grandi [60] focuses on temporal versioning of light-weight ontologies expressed in RDF(S) and show how the multi-temporal RDF data model proposed in [58] can be used to support RDF(S) ontology versioning. The data model is equipped with a complete set of primitive ontology change operations, which are defined in terms of low-level updates acting on RDF triples. When used within the transaction template, which has also been introduced, the proposed ontology changes allow a KBA to define and manage temporal versions of an RDF(S) ontology.

However, whereas all the works in this group, including [58], [59], and [60], basically propose *ad hoc* solutions for the management of temporal versions of RDF, RDF(S) or OWL resources, we introduce a τXSchema-like general framework embodying a disciplined and principled approach to temporal versioning of Semantic Web documents, both at instance and at schema levels.

VII. Conclusion and Future Work

In this paper, we proposed τOWL, a τXSchema-like framework, which allows creating a temporal OWL 2 ontology from a conventional OWL 2 ontology and a set of logical and physical annotations. Our framework ensures logical and physical data independence, since it (i) separates conventional schema, logical annotations, and physical annotations, and (ii) allows each one of these three components to be changed independently and safely. Furthermore, adoption of τOWL provides for a low-impact solution, since it requires neither modifications of existing Semantic Web documents, nor extensions to the OWL 2 recommendation and Semantic Web standards. The extension of OWL 2 to temporal and versioning aspects is performed without having to depend on approval of proposed extensions by standardization committees (and on upgrade of existing tools conforming to standards to comply with approved extensions).

Moreover, we have extended our τOWL framework by proposing a general approach for schema versioning in it and focusing on the definition of a set of change primitives for supporting the evolution of both temporal and conventional ontology schema. Our approach helps the KBA in the management of conventional schema changes in τOWL-based Semantic Web repositories and guarantees the maintenance of a full history of evolving conventional ontology instances and schemata.

In order to embed our approach into a user-friendly environment at the disposal of KBAs, a tool for the management of temporal ontologies in the τOWL framework is under development at the University of Sfax. A first release of the tool, named τOWL-Manager [62], is already available and implements our τOWL framework with the support of temporal versioning of ontology instances. The new release currently under development will support all schema change primitives proposed in this paper, and put them at the disposal of KBAs, via an intuitive interface which assists them in expressing their needs to fulfill application requirements. Furthermore, we are also extending the present work by defining a complete set of schema change primitives for the ontology annotation document which stores logical and physical annotations specified on the conventional ontology schema.

Besides, in order to further simplify the work of KBAs and to make our approach more useful, we intend to propose in our future work high-level and more user-friendly schema change operations, based on the primitives introduced in this paper and on those that will be defined for changing annotations. A high-level operation is a valid sequence of primitives, which correspond to frequent schema evolution needs and allows expressing complex changes in a more compact way [63]. Moreover, we will also allow the KBA to build his/her own high-level schema change operations, by combining in a consistent way pre-defined high-level operations and/or primitives, through the use of a specific tool that will be integrated in a future release of the τOWL-Manager environment.

As a part of our future work, we will also thoroughly study the propagation of changes performed on conventional ontology schema, i.e., their effects on conventional ontology instances stored in conventional ontology instance documents, which are valid with respect to the conventional ontology schema.

Finally, we also plan to address querying of temporal ontology instances under schema versioning, in the τOWL framework. The starting point for this extension will be the T-SPARQL language [27], which allows end users and KBAs to express queries on multi-temporal ontology instances (which are composed of multi-temporal RDF triples) under a single ontology schema version; such a language could be extended with features to support schema versions and specify multi-schema queries, i.e., queries involving instances of several schema versions [64].

References

[1] A. Zekri, Z. Brahmia, F. Grandi, and R. Bouaziz, "τOWL: A Framework for Managing Temporal Semantic Web Documents," Proceedings of the 8th International Conference on Advances in Semantic Processing (SEMAPRO 2014), Rome, Italy, 24-28 August 2014, pp. 33-41.

[2] C. S. Jensen and R. T. Snodgrass, "Temporal Data Management," IEEE Transactions on Knowledge and Data Engineering, vol. 11, January/February 1999, pp. 36-44.

[3] O. Etzion, S. Jajodia, and S. Sripada (eds.), "Temporal Databases: Research and Practice," LNCS 1399, Springer-Verlag, 1998.

[4] C. S. Jensen and R. T. Snodgrass, "Temporal Database," in Liu L., Özsu M.T., (Eds.), Encyclopedia of Database Systems, Springer US, 2009, pp. 2957-2960.

[5] F. Grandi, "Temporal Databases," in M. Koshrow-Pour, (Ed.), Encyclopedia of Information Science and Technology (3rd Ed.), IGI Global, Hershey, in press.

[6] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, "The World Wide Web," Communications of the ACM, vol. 37, August 1994, pp. 76-82.

[7] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," Scientific American, vol. 284, May 2001, pp. 34-43.

[8] W3C Semantic Web Activity. <http://www.w3.org/2001/sw/> [retrieved: May, 2015]

[9] N. Guarino (Ed.), Formal Ontology in Information Systems, IOS Press, Amsterdam, 1998.

[10] W3C, OWL 2 Web Ontology Language – Primer (Second Edition), W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-primer/> [retrieved: May, 2015]

[11] W3C, OWL 2 Web Ontology Language – Document Overview (Second Edition), W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-overview/> [retrieved: May, 2015]

[12] W3C, OWL 2 Web Ontology Language – Profiles (Second Edition), W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-profiles/> [retrieved: May, 2015]

[13] F. Grandi, "Introducing an Annotated Bibliography on Temporal and Evolution Aspects in the Semantic Web," SIGMOD Record, vol. 41, December 2012, pp. 18-21.

[14] F. Currim, S. Currim, C. E. Dyreson, and R. T. Snodgrass, "A Tale of Two Schemas: Creating a Temporal XML Schema from a Snapshot Schema with tXSchema," Proceedings of the 9th International Conference on Extending Database Technology (EDBT 2004), Heraklion, Crete, Greece, 14-18

March 2004, pp. 348-365.

[15] R. T. Snodgrass, C. E. Dyreson, F. Currim, S. Currim, and S. Joshi, "Validating Quicksand: Schema Versioning in τXSchema," Data Knowledge and Engineering, vol. 65, May 2008, pp. 223-242.

[16] F. Currim, S. Currim, C. E. Dyreson, S. Joshi, R. T. Snodgrass, S. W. Thomas, and E. Roeder, "τXSchema: Support for Data- and Schema-Versioned XML Documents," TimeCenter Technical Report TR-91, 279 pages, September 2009. <http://timecenter.cs.aau.dk/TimeCenterPublications/TR-91.pdf> [retrieved: May, 2015]

[17] C. E. Dyreson and F. Grandi, "Temporal XML," in L. Liu and M. T. Özsu (Eds.), Encyclopedia of Database Systems, Springer US, 2009, pp. 3032-3035.

[18] Z. Brahmia, R. Bouaziz, F. Grandi, and B. Oliboni, "Schema Versioning in τXSchema-Based Multitemporal XML Repositories," Proceedings of the 5th IEEE International Conference on Research Challenges in Information Science (RCIS 2011), Guadeloupe - French West Indies, France, 19-21 May 2011, pp. 1-12.

[19] Z. Brahmia, F. Grandi, B. Oliboni, and R. Bouaziz, "Versioning of Conventional Schema in the τXSchema Framework," Proceedings of the 8th International Conference on Signal Image Technology & Internet Systems (SITIS'2012), Sorrento – Naples, Italy, 25-29 November 2012, pp. 510-518.

[20] Z. Brahmia, F. Grandi, B. Oliboni, and R. Bouaziz, "Schema Change Operations for Full Support of Schema Versioning in the τXSchema Framework," International Journal of Information Technology and Web Engineering, vol. 9, April-June 2014, pp. 20-46.

[21] T. Burns, E. Fong, D. Jefferson, R. Knox, L. Mark, C. Reedy, L. Reich, N. Roussopoulos, and W. Truszkowski, "Reference Model for DBMS Standardization, Database Architecture Framework Task Group (DAFTG) of the ANSI/X3/SPARC Database System Study Group," SIGMOD Record, vol. 15, March 1986, pp. 19-58.

[22] J. F. Roddick, "Schema Versioning," in Liu L., Özsu M.T., (Eds.), Encyclopedia of Database Systems, Springer US, 2009, pp. 2499-2502.

[23] Z. Brahmia, F. Grandi, B. Oliboni, and R. Bouaziz, "Schema Versioning," in M. Khosrow-Pour (Ed.), Encyclopedia of Information Science and Technology (3rd Ed.), IGI Global, 2014, pp. 7651-7661.

[24] D. Rogozan and G. Paquette, "Managing ontology changes on the semantic web," Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2005), Compiegne, France, 19-22 September 2005, pp. 430-433.

[25] The Friend of a Friend (FOAF) project. <http://www.foaf-project.org/> [retrieved: May, 2015]

[26] W3C, Resource Description Framework (RDF), Semantic Web Standard. <http://www.w3.org/RDF/> [retrieved: May, 2015]

[27] F. Grandi, "T-SPARQL: a TSQL2-like temporal query language for RDF," Proceedings of the 1st International Workshop on Querying Graph Structured Data (GraphQ 2010), Novi Sad, Serbia, 20 September 2010, pp. 21-30.

[28] J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass, "On the Semantics of "Now" in Databases," ACM Transactions on Database Systems, vol. 22, June 1997, pp. 171–214.

[29] W3C, RDF/XML Syntax Specification (Revised), W3C Recommendation, 10 February 2004. <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/> [retrieved: May, 2015]

[30] XML Schema Part 0: Primer Second Edition, W3C Recommendation, 28 October 2004. <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/> [retrieved: May, 2015]

[31] W3C, XQuery Update Facility 1.0, W3C Candidate Recommendation, 17 March 2011. <http://www.w3.org/TR/2011/REC-xquery-update-10-20110317/> [retrieved: May, 2015]

[32] W3C, Time Ontology in OWL, W3C Working Draft, 27 september 2006. <http://www.w3.org/TR/owl-time/> [retrieved: May, 2015]

[33] C. A. Welty and R. Fikes, "A Reusable Ontology for Fluents in OWL," Proceedings of the 4th International Conference on Formal Ontology in Information Systems (FOIS 2006), Baltimore, Maryland, USA, 9-11 November 2006, pp. 226-236.

[34] W3C, OWL 2 Web Ontology Language – New Features and Rationale (Second Edition), W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-new-features/> [retrieved: May, 2015]

[35] W3C, SPARQL Query Language for RDF, W3C Recommendation, 15 January 2008, <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/> [retrieved: May, 2015]

[36] A. Analyti and I. Pachoulakis, "A survey on models and query languages for temporally annotated RDF," International Journal of Advanced Computer Science and Applications, vol. 3, September 2012, pp. 28-35.

[37] C. Gutiérrez, C. A. Hurtado, and A. A. Vaisman, "Introducing time into RDF," IEEE Transactions on Knowledge and Data Engineering, vol. 19, February 2007, pp. 207-218.

[38] E. Anagnostopoulos, S. Batsakis, and E. G. M. Petrakis, "CHRONOS: A Reasoning Engine for Qualitative Temporal Information in OWL," Proceedings of the 17th International Conference in Knowledge-Based and Intelligent Information & Engineering Systems (KES 2013), Kitakyushu, Japan, 9-11 September 2013, pp. 70-77.

[39] B. Motik, "Representing and Querying Validity Time in RDF and OWL: A Logic-based Approach," Proceedings of the 9th International Semantic Web Conference (ISWC 2010), Shanghai, China, 7-11 November 2010, pp. 550-565.

[40] V. Zamborlini and G. Guizzardi, "On the representation of temporally changing information in OWL," Workshops Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference (EDOCW 2010), Vitória, Brazil, 25-29 October 2010, pp. 283-292.

[41] M. J. O'Connor and A. K. Das, "A method for representing and querying temporal information inOWL," In Biomedical Engineering Systems and Technologies, volume 127 of Communications in Computer and Information Science, pp. 97-110. Springer-Verlag, Heidelberg, Germany, 2011.

[42] V. Milea, F. Frasincar, and U. Kaymak, "tOWL: A Temporal Web Ontology Language," IEEE Transactions on Systems, Man, and Cybernetics, Part B, vol. 42, February 2012, pp. 268-281.

[43] J. F. Allen, "Maintaining Knowledge About Temporal Intervals," Communications of the ACM, vol. 26, November 1983, pp. 832-843.

[44] M. C. A. Klein and D. Fensel, "Ontology versioning on the Semantic Web," Proceedings of the 1st Semantic Web Working Symposium (SWWS 2001), Stanford University, California, USA, 30 July – 1 August 2001, pp. 75-91.

[45] M. C. A. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov, "Ontology Versioning and Change Detection on the Web," Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW 2002), Siguenza, Spain, 1-4 October 2002, pp. 197-212.

[46] N. F. Noy and M. A. Musen, "Ontology versioning in an ontology management framework," IEEE Intelligent Systems, vol. 19, July 2004, pp. 6-13.

[47] J. Eder and C. Koncilia, "Modelling changes in ontologies," Proceedings of the OTM Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004, Agia Napa, Cyprus, 25-29 October 2004, pp. 662-673.

[48] T. Redmond, M. Smith, N. Drummond, and T. Tudorache, "Managing change: an ontology version control system," Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008), Karlsruhe, Germany, 26-27 October 2008. CEUR Workshop Proceedings (CEUR-WS.org), Vol-432. <http://ceur-ws.org/Vol-432/owled2008eu_submission_33.pdf> [retrieved: May, 2015]

[49] P. De Leenheer, "Revising and managing multiple ontology versions in a possible worlds setting," Proceedings of the OTM Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004, Agia Napa, Cyprus, 25-29 October 2004, pp. 798-809.

[50] P. Haase and L. Stojanovic, "Consistent Evolution of OWL Ontologies," Proceedings of the 2nd European Semantic Web Conference (ESWC 2005), Heraklion, Crete, Greece, 29 May – 1 June 2005, pp. 182-197.

[51] N. Sassi, W. Jaziri, and F. Gargouri, "How to Evolve Ontology and Maintain Its Coherence - A Corrective Operations-based Approach," Proceedings of the International Conference on Knowledge Engineering and Ontology Development (KEOD 2009), Funchal - Madeira, Portugal, 6-8 October 2009, pp. 384-387.

[52] W. Jaziri, N. Sassi, and F. Gargouri, "Approach and tool to evolve ontology and maintain its coherence," International Journal of Metadata, Semantics, and Ontologies, vol. 5, May 2010, pp. 151-166.

[53] A. Arara and D. Benslimane, "Towards formal ontologies requirements with multiple perspectives," Proceedings of the 6th International Conference on Flexible Query Answering Systems (FQAS 2004), Lyon, France, 24-26 June 2004, pp. 150-160.

[54] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt, "Contextualizing ontologies," Journal of Web Semantics, vol. 1, October 2004, pp. 325-343.

[55] J. Heflin and Z. Pan, "A model theoretic semantics for ontology versioning," Proceedings of the 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan, 7-11 November 2004, pp. 62-76.

[56] M. Völkel and T. Groza, "SemVersion: An RDF-based Ontology Versioning System," Proceedings of the IADIS International Conference on WWW/Internet (ICWI 2006), Murcia, Spain, 5-8 October 2006, vol. 1, pp. 195-202. <http://www.xam.de/2006/10-SemVersion-ICIW2006.pdf> [retrieved: May, 2015]

[57] P. Bedi and S. Marwaha, "Versioning OWL ontology using temporal tags," Proceedings of the 21st International Conference on Computer, Electrical, Systems Science and Engineering (CESSE'07), Vienna, Austria, 25-27 May 2007, pp. 332-337.

[58] F. Grandi, "Multi-temporal RDF ontology versioning," Proceedings of the 3rd International Workshop on Ontology Dynamics (IWOD 2009), Washington DC, USA, 26 October 2009. CEUR Workshop Proceedings (CEUR-WS.org), Vol-519. <http://ceur-ws.org/Vol-519/grandi.pdf> [retrieved: May, 2015]

[59] F. Grandi and M. R. Scalas, "The valid ontology: A simple OWL temporal versioning framework," Proceedings of the 3rd International Conference on Advances in Semantic Processing (SEMAPRO 2009), Sliema, Malta, 11-16 October 2009, pp. 98-102.

[60] F. Grandi, "Light-weight Ontology Versioning with Multi-temporal RDF Schema," Proceedings of the 5th International Conference on Advances in Semantic Processing (SEMAPRO 2011), Lisbon, Portugal, 20-25 November 2011, pp. 42-48.

[61] H. Kondylakis and D. Plexousakis, "Ontology evolution without tears," Journal of Web Semantics, vol. 19, March 2013, pp. 42-58.

[62] A. Zekri, Z. Brahmia, F. Grandi, and R. Bouaziz, "τOWL-Manager: A Tool for Managing Temporal Semantic Web Documents in the τOWL Framework," Proceedings of the 9th International Conference on Advances in Semantic Processing (SEMAPRO 2015), Nice, France, 19-24 July 2015, in press.

[63] Z. Brahmia, F. Grandi, B. Oliboni, and R. Bouaziz, "High-level Operations for Changing Temporal Schema, Conventional Schema and Annotations, in the τXSchema Framework," TimeCenter Technical Report TR-96, 56 pages, January 2014. <http://timecenter.cs.aau.dk/TimeCenterPublications/TR-96.pdf> [retrieved: May, 2015]

[64] F. Grandi, "A relational multi-schema data model and query language for full support of schema versioning," Proceedings of SEBD 2002 – National Conference on Advanced Database Systems, Isola d'Elba, Italy, 19-21 June 2002, pp. 323-336.