J Data Semant (2016) 5:141–163
DOI 10.1007/s13740-016-0066-3

ORIGINAL ARTICLE

τOWL: A Systematic Approach to Temporal Versioning of Semantic Web Ontologies

Abir Zekri[1] · **Zouhaier Brahmia**[1] · **Fabio Grandi**[2] · **Rafik Bouaziz**[1]

Received: 31 July 2015 / Revised: 11 April 2016 / Accepted: 30 May 2016 / Published online: 10 June 2016

Abstract The W3C OWL 2 recommendation is an ontology language for the Semantic Web. It allows defining both schema (i.e., entities, axioms, and expressions) and instances (i.e., individuals) of ontologies. However, OWL 2 lacks explicit support for time-varying schema or for time-varying instances. Hence, knowledge engineers or maintainers of semantics-based Web resources have to use ad hoc techniques to specify OWL 2 time-varying ontologies. In this paper, for a disciplined and systematic approach to the temporal management of Semantic Web ontologies, we propose the adoption of a framework called temporal OWL 2 (τOWL), which is inspired by the τXSchema framework defined for XML data. In a way similar to what happens in τXSchema, 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 the ontology can vary over time; physical annotations specify how the time-varying aspects are represented in the OWL 2 document. Using annotations to integrate temporal aspects in the traditional Semantic Web, our framework (1) guarantees logical and physical data independence for temporal schemas and (2) provides a low-impact solution, since it requires neither modifications of existing Semantic Web ontologies, nor extensions to the OWL 2 recommendation and Semantic Web standards. Moreover, since the conventional schema and annotation documents could evolve over time to respond to new applications' requirements, τOWL supports temporal schema versioning by allowing changing these components and by keeping track of their evolution through the conventional schema versions and annotation document versions, respectively. Two complete sets of operations are proposed for changing the conventional schema and annotation documents; to complete the figure, a set of operations is also introduced for updating temporal schema which must be changed consequently each time one of the mentioned components evolves over time. To show the feasibility of our approach, a prototype tool, named τOWL-Manager, is presented.

Keywords Semantic Web · Ontology · OWL 2 · τXSchema · Logical annotations · Physical annotations · Conventional schema · Temporal schema · Temporal ontology · Temporal database · XML Schema · XML · Schema change · Schema versioning

1 Introduction

Due to the dynamic nature of the Web [3], ontologies [19] that are used on the Web—like other Web application components, such as Web databases and Web pages—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, to represent, store, and retrieve time-varying ontologies.

✉ Zouhaier Brahmia
zouhaier.brahmia@fsegs.rnu.tn

Abir Zekri
abir.zekri@fsegs.rnu.tn

Fabio Grandi
fabio.grandi@unibo.it

Rafik Bouaziz
raf.bouaziz@fsegs.rnu.tn

[1] University of Sfax, Sfax, Tunisia

[2] Alma Mater Studiorum, Università di Bologna, Bologna, Italy

Unfortunately, while there is a sustained interest for temporal and evolution aspects in the research community [16], existing Semantic Web [4] standards and the 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 [35] 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. 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, management tools with a built-in temporal support are needed. For that purpose, we propose in this paper a framework, called τOWL, for managing temporal Semantic Web ontologies, 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 with their τXSchema [9,30] to XML and XML Schema [32]. τXSchema is a powerful 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 [10]. To complete the framework and extend its functionalities, τXSchema has also been augmented by defining necessary schema change operations [5].

Being defined as a τXSchema-like framework, τOWL allows creating 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 the ontology can vary over time; physical annotations specify how the time-varying aspects are represented in the OWL 2 document. By using temporal schema and annotations to introduce temporal aspects in the conventional Semantic Web, our framework (1) guarantees logical and physical data independence [8] for temporal ontologies and (2) 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. In general, by Semantic Web (OWL 2 or RDF) document, we denote an OWL 2 or RDF file containing either the specification of an ontology schema, either the specification of a collection of ontology instances, or both.

In its first formulation [40], τOWL was defined as an environment for the management of ontologies with time-varying instances conforming to a common (time-invariant) schema. However, since ontology structural definitions are also evolving over time to reflect changes in real-world applications [28] and keeping a fully fledged history of ontology changes (i.e., changes affecting ontology instances and/or ontology schema) is a very required feature for advanced Semantic Web applications, the τOWL framework has been extended in Zekri et al. [41,42,44] to also support schema versioning [7] that is the most powerful technique for performing schema changes, consistently propagating their effects on underlying instances, and guaranteeing the maintenance of a complete history of schema and instances. In this context, we further illustrate our approach for managing temporal schema versioning in τOWL and its implementation in a prototype tool. In particular, we (1) describe the process of creating and evolving temporal ontologies in τOWL, (2) introduce schema change operations supported by τOWL, which help KBAs in defining and changing (conventional and temporal) ontology schema, and (3) show their user-friendly deployment through a prototype tool, named τOWL-Manager, which allows constructing and updating temporal ontologies.

The remainder of the paper is organized as follows. Section 2 motivates the need for an efficient management of time-varying Semantic Web ontologies. Section 3 describes the τOWL framework that we propose for extending the Semantic Web to temporal aspects: the architecture of τOWL is presented, details on its components and support tools are given, and the prototype system τOWL-Manager, which implements this framework, is introduced. Section 4 proposes our approach for temporal schema versioning in τOWL: the schema versioning process, schema change operations acting on conventional and temporal ontology schema, and the schema versioning support in τOWL-Manager. Section 5 discusses related work. Section 6 provides a summary of the paper and some remarks about our future work.

2 Motivation

In this section, we present a motivating example that shows the limitation of the OWL 2 language for explicitly supporting time-varying ontologies.

GoodRelations[1] is a standardized ontology for e-commerce, used to publish details on products sold online. For example, Best Buy Co., Inc.[2] is publishing RDF descriptions of their products (consumer electronics, personal computers, entertainment software, etc.), on the Web of Linked Data, using the GoodRelations ontology. A fragment of the GoodRelations RDF document of games sold by "Best Buy"

[1] http://www.heppnetz.de/ontologies/goodrelations/v1. [retrieved: April, 2016].

[2] http://www.bestbuy.com/ [retrieved: April, 2016].

```
…
  <gr:Offering rdf:ID="Offering_9223752">
    <gr:hasPriceSpecification>
      <gr:UnitPriceSpecification rdf:ID="UnitPriceSpecification_9223752_1">
        <gr:hasCurrency>USD</gr:hasCurrency>
        <gr:hasCurrencyValue>29.99</gr:hasCurrencyValue>
        <gr:priceType>SRP</gr:priceType>
        …
      </gr:UnitPriceSpecification>
    </gr:hasPriceSpecification>
    …
  </gr:Offering>
…
```

Fig. 1 Fragment of games GoodRelations RDF document on June 15, 2015

is shown in Fig. 1. It provides, according to the GoodRelations ontology, some details of a game offering having the ID "9223752": the currency (USD), the currency value (29.99), and the price type (SRP, for special reduction price).

Assume that information about that offering was added on June 15, 2015. On July 08, 2015, the KBA of "Best Buy" modified the currency value from "29.99" to "27.25" and the price type from "SRP" to "SP (for sale price)". Thus, the corresponding fragment of the GoodRelations RDF document was revised to that shown in Fig. 2.

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 [14]. 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 [9].

Suppose that the KBA of "Best Buy" would like to keep track of the changes performed on the GoodRelations RDF information by storing both versions of Figs. 1 and 2 in a single (temporal) RDF document. As a result, Fig. 3 shows a fragment of a time-varying Semantic Web document that captures the history of the specified details of the offering "9223752".

In this example, the KBA uses valid-time to capture the history of information. To timestamp the entities which can evolve over time, he/she uses the following optional tags: **hasCurrencyValueValidityStartTime** and **hasCurrencyValueValidityEndTime**, for recording the currency value evolution, and **priceTypeValidityStartTime** and **priceTypeValidityEndTime**, for keeping the price-type history. These are optional data properties which can be added to a temporal entity. The domain of hasCurrencyValueValidityEndTime or priceTypeValidityEndTime includes the value "now" [5]; 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 GoodRelations ontology presented in Fig. 4 contains the conventional schema for the GoodRelations RDF document presented in both Figs. 1 and 2. The conventional schema is the schema for an individual version, which allows updating and querying individual versions.

The problem is that the time-varying ontology (see Fig. 3) does not conform to the conventional schema (see Fig. 4). Thus, to resolve this problem, the KBA needs 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. Moreover, notice that this example involves the representation and management of an ontology with time-varying instances all conforming to a common (time-invariant) schema. The management of a time-varying schema and the support of schema versioning is a further complication that will also be addressed in this paper.

```
…
  <gr:Offering rdf:ID="Offering_9223752">
    <gr:hasPriceSpecification>
      <gr:UnitPriceSpecification rdf:ID="UnitPriceSpecification_9223752_1">
        <gr:hasCurrency>USD</gr:hasCurrency>
        <gr:hasCurrencyValue>27.25</gr:hasCurrencyValue>
        <gr:priceType>SP</gr:priceType>
        …
      </gr:UnitPriceSpecification>
    </gr:hasPriceSpecification>
    …
  </gr:Offering>
…
```

Fig. 2 Fragment of games GoodRelations RDF document on July 08, 2015

```
…
  <gr:Offering rdf:ID="Offering_9223752">
   <gr:hasPriceSpecification>
    <gr:UnitPriceSpecification rdf:ID="UnitPriceSpecification_9223752_1">
     <gr:hasCurrency>USD</gr:hasCurrency>
     <versionedHasCurrencyValue>
      <hasCurrencyValueVersion>
       <hasCurrencyValueValidityStartTime>2015-06-15</hasCurrencyValueValidityStartTime>
       <hasCurrencyValueValidityEndTime>2015-07-07</hasCurrencyValueValidityEndTime>
       <gr:hasCurrencyValue>29.99</gr:hasCurrencyValue>
      </hasCurrencyValueVersion>
      <hasCurrencyValueVersion>
       <hasCurrencyValueValidityStartTime>2015-07-08</hasCurrencyValueValidityStartTime>
       <hasCurrencyValueValidityEndTime>now</hasCurrencyValueValidityEndTime>
       <gr:hasCurrencyValue>27.25</gr:hasCurrencyValue>
      </hasCurrencyValueVersion>
     </versionedHasCurrencyValue>
     <versionedPriceType>
      <priceTypeVersion>
       <priceTypeValidityStartTime>2015-06-15</priceTypeValidityStartTime>
       <priceTypeValidityEndTime>2015-07-07</priceTypeValidityEndTime>
       <gr:priceType>SRP</gr:priceType>
      </priceTypeVersion>
      <priceTypeVersion>
       <priceTypeValidityStartTime>2015-07-08</priceTypeValidityStartTime>
       <priceTypeValidityEndTime>now</priceTypeValidityEndTime>
       <gr:priceType>SP</gr:priceType>
      </priceTypeVersion>
     </versionedPriceTypepriceType>
     …
    </gr:UnitPriceSpecification>
   </gr:hasPriceSpecification>
   …
  </gr:Offering>
…
```

Fig. 3 Fragment of the time-varying games GoodRelations RDF document

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:gr="http://purl.org/goodrelations/v1#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  …
  <owl:Class rdf:about="http://purl.org/goodrelations/v1#Offering">
   <owl:ObjectProperty rdf:about="http://purl.org/goodrelations/v1#hasPriceSpecification">
    <rdfs:range rdf:resource="http://purl.org/goodrelations/v1#PriceSpecification"/>
     …
     <rdfs:domain>
      <owl:Class>
       <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://purl.org/goodrelations/v1#Offering"/>
        <rdf:Description rdf:about="http://schema.org/Offer"/>
       </owl:unionOf>
      </owl:Class>
     </rdfs:domain>
     <rdfs:isDefinedBy rdf:resource="http://purl.org/goodrelations/v1"/>
   </owl:ObjectProperty>
   …
  </owl:Class>
  …
</rdf:RDF>
```

Fig. 4 RDF/XML extract from the OWL 2 GoodRelations ontology

3 The τOWL Framework

In this section, we present in a detailed way our τOWL framework. First, we provide the goals of this framework. Then, we describe its architecture. Next, we τOWL-Manager, a prototype tool which implements τOWL and allows creating and updating temporal ontologies. Finally, we resume our running example, introduced in the previous section, to illustrate the functioning of τOWL while providing some screenshots which demonstrate the use of τOWL-Manager for managing temporal ontologies of the same example.

3.1 Goals

There are several goals which can be fulfilled when augmenting the OWL 2 language to support time-varying instances. In particular, our τOWL proposal is a principled and systematic approach aimed at satisfying the following seven requirements:

(1) facilitating the management of time for KBAs;
(2) supporting both valid time and transaction time;
(3) supporting (temporal) versioning of OWL 2 instances;
(4) keeping compatibility with existing OWL 2 W3C recommendations, and editors, and not requiring any changes to them;
(5) supporting existing applications that are already using OWL 2 ontologies;
(6) providing OWL 2 data independence, so that changes at the logical level are isolated from those performed at the physical level, and vice versa;
(7) accommodating a variety of physical representations for time-varying OWL 2 instances.

3.2 Architecture

In the following, we describe the architecture of τOWL and the embedded 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 those of τOWL. The new 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 two principles: (1) separation between the conventional schema and the temporal schema, and also between the conventional instances and the temporal instances and (2) use of logical and physical annotations to specify temporal and physical aspects, respectively, at schema level.

Figure 5 shows the architecture of τOWL. Notice that, in this figure, 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).

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 [33], which is, according to the OWL 2 specification document [36], 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 [30] 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), and (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 [30] 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 [30].

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

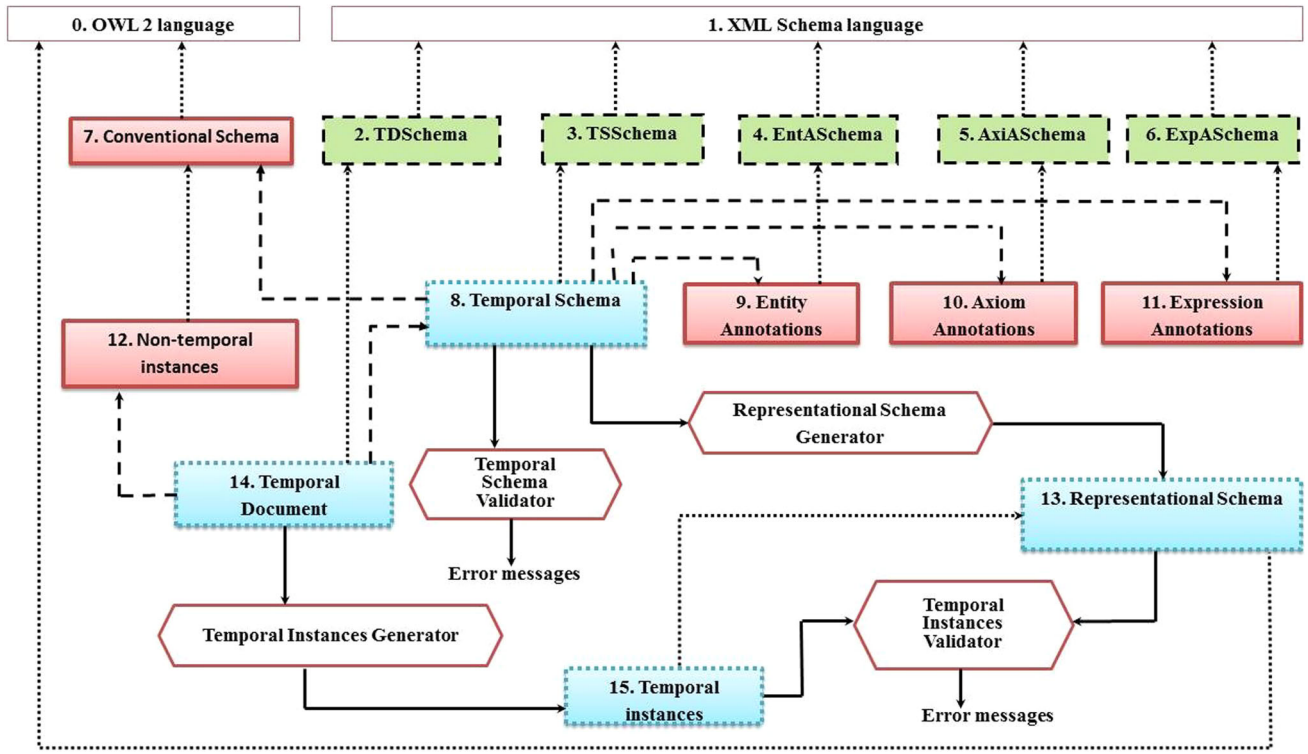

Fig. 5 τOWL architecture

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).

Notice that EntASchema, AxiASchema, and ExpASchema, which all contain both logical and physical annotations, are XML Schemas [32]. 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) 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 the 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) 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 the 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 doc-

ument 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 to ensure that the logical and physical entity annotations, axiom annotations, and expression annotations are (1) valid with respect to their corresponding schemas (i.e., EntASchema, AxiASchema, and ExpASchema, respectively) and (2) 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: (1) 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); (2) automatically from instances stored in a knowledge base, i.e., as the result of a "temporal query" or a "temporal view"; and (3) automatically from a third-party tool; (4) 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.

3.3 τOWL-Manager

τOWL-Manager is a prototype system which implements our τOWL approach and shows its feasibility. In particular, τOWL-Manager is a Java (JDK 1.7) application, developed in the Integrated Development Environment (IDE) "Eclipse Helios", using (1) the OWL Application Programming Interface (API) [22], which is a Java API and a reference implementation, for creating and manipulating OWL ontologies, and (2) the Java Document Object Model (JDOM) API for creating and manipulating XML files. The first released version of τOWL-Manager, described in Zekri et al. [43], allowed KBAs to manage temporal ontology instance documents in the τOWL framework. The current version of τOWL-Manager, presented in this paper, also allows the creation and maintenance of ontology schemata, adding support of ontology schema versioning to the implementation of the τOWL framework. In particular, updating an instance document gives rise (1) to the creation of a new version of this document with its corresponding transaction timestamp, and (2) to an update of the temporal document to integrate the new version in the τOWL environment. Moreover, in the current τOWL-Manager release, changing a conventional ontology schema and/or an ontology annotation document implies (1) the creation of a new version of this conventional schema and/or this annotation document with its corresponding transaction timestamp, (2) an update of the temporal ontology schema to integrate the new schema version in the τOWL environment, (3) an update of the ontology instances underlying the modified schema to adapt them to the new schema version (change propagation), and (4) the update of the temporal document to integrate the new document instance version in the τOWL environment (implicitly connecting the new version of the ontology instance document to the new ontology schema version). More details on τOWL-Manager and on its use for managing temporal ontologies will be showed in Sects. 3.4 and 4.4.

3.4 Running Example

To show the functionalities of the proposed approach, we continue our motivating example of Sect. 2, to show how management of temporal ontology versions is dealt with in the τOWL approach. On June 15, 2015, the KBA creates a conventional ontology schema (box 7), named "GameSaleSchema_V1.owl" (as in Fig. 4), and a conventional ontology document (box 12), named "GameSales_V1.rdf" (as in Fig. 1), which is valid with respect to this schema. Figure 6 shows the ontology instantiation which leads to the creation of "GameSales_V1.rdf", through the use of τOWL-Manager. In fact, after choosing a conventional ontology schema, the KBA can create and save its instances.

We assume also that the KBA defines 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 "GameSaleAnnotations_V1.xml", as shown in Fig. 7. Indeed, as to logical annotations, he/she decides to make the content of the "gr:hasCurrencyValue" data property and the content of the "gr:priceType" data property varying in valid time (to keep the history along valid time of the changes the currency value and the price type of each offering undergoes). As to physical annotations, he/she chooses to add a transaction-time physical timestamp to the "gr:UnitPriceSpecification" class (i.e., whenever any data property (e.g., "gr:hasCurrency", "gr:hasCurrencyValue", or "gr:priceType") of an instance

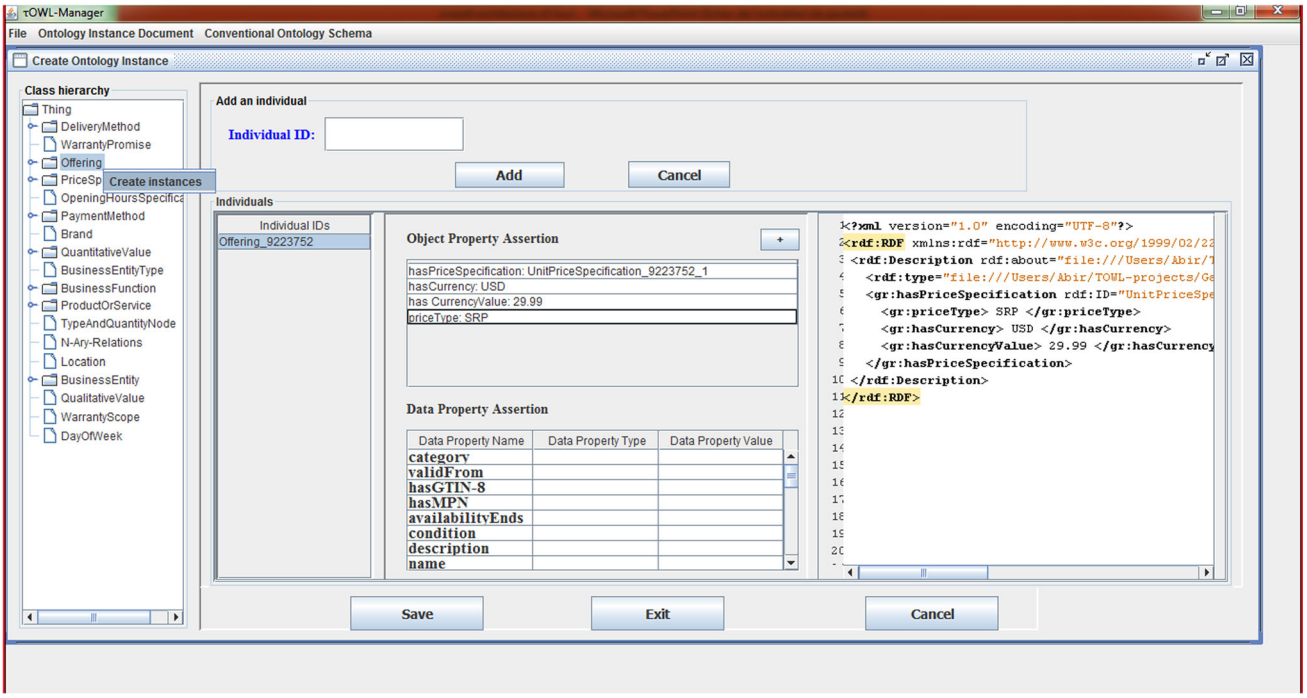

Fig. 6 Populating the new conventional ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<ontologyAnnotationSet>
  <logicalAnnotations>
    <item target="/Offering/hasPriceSpecification/UnitPriceSpecification/hasCurrencyValue">
      <validTime kind="state" content="varying" existence="constant"/>
    </item>
    <item target="/Offering/hasPriceSpecification/UnitPriceSpecification/priceType">
      <validTime kind="state" content="varying" existence="constant"/>
    </item>
  </logicalAnnotations>
  <physicalAnnotations>
    <stamp target="/Offering/hasPriceSpecification/UnitPriceSpecification"
           dataInclusion="expandedVersion">
      <stampkind timeDimension="transactionTime" stampBounds="extent"/>
    </stamp>
  </physicalAnnotations>
</ontologyAnnotationSet>
```

Fig. 7 Ontology annotation document on June 15, 2015

of the class "gr:UnitPriceSpecification" changes, the entire "gr:UnitPriceSpecification" instance is repeated to represent a new temporal version). Figures 8 and 9 show the specification of a logical annotation on the data property "gr:hasCurrencyValue" and the specification of a physical annotation on the class "gr:UnitPriceSpecification", respectively, through the use of τOWL-Manager.

After that, the system creates the temporal ontology schema (box 8) in Fig. 10, which ties "GameSaleSchema_V1.owl" and "GameSaleAnnotations_V1.xml" together; this temporal schema is saved in an XML file titled "GameSaleTemporalSchema.xml". Consequently, the system uses the temporal ontology schema of Fig. 10 and the conventional ontology document in Fig. 1 to create a temporal document (box 14) as in Fig. 11, 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 Fig. 12.

On July 08, 2015, the KBA updates the conventional ontology document "GameSales_V1.rdf" as presented in Sect. 2 to produce a new conventional ontology document named "GameSales_V2.rdf" (as in Fig. 2). Figures 13 and 14 show how such an update is performed using τOWL-Manager: first, the KBA chooses "GameSales_V1.rdf" as the ontology instance document version that must be updated (see Fig. 13); then, he/she changes the chosen version by modi-

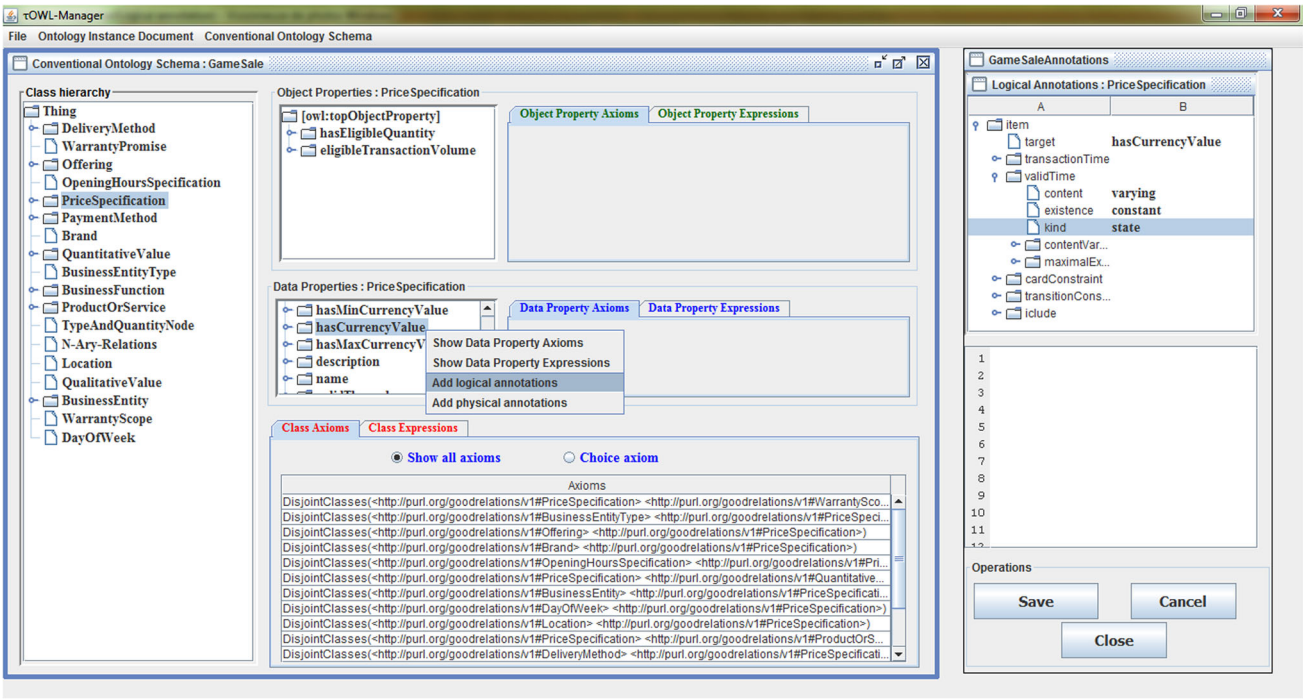

Fig. 8 Specifying a logical annotation on the data property "hasCurrencyValue"

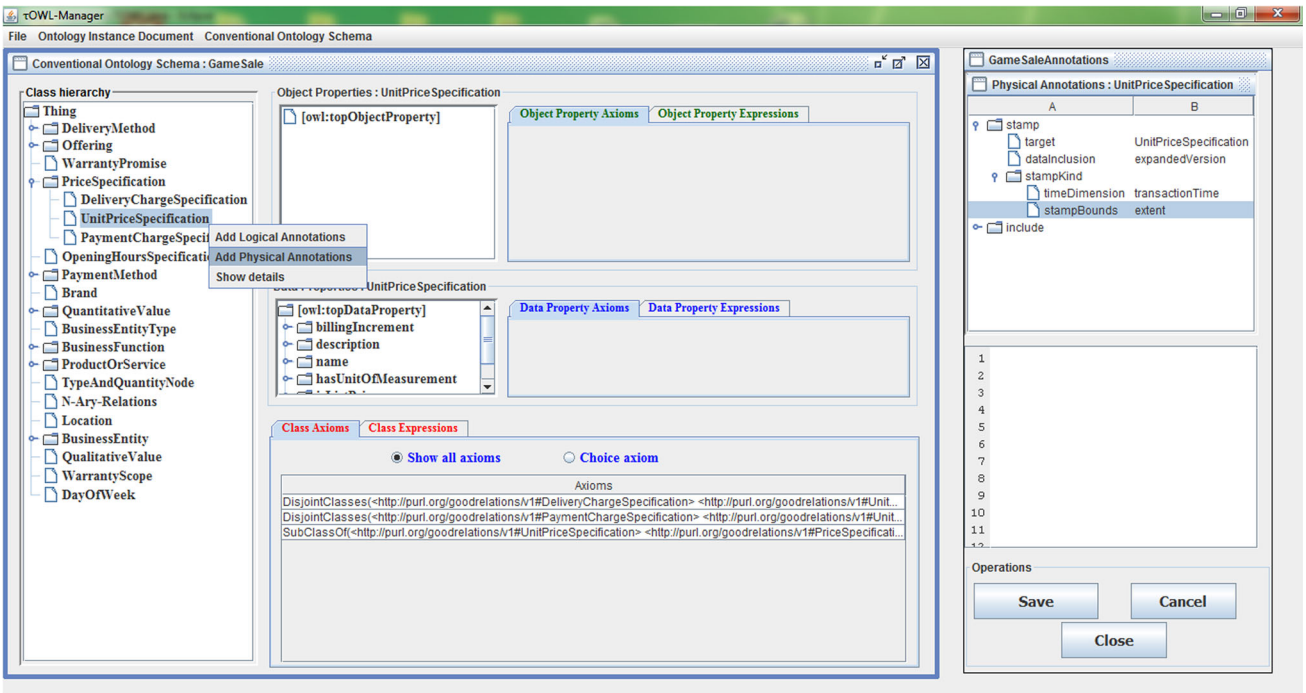

Fig. 9 Specifying a physical annotation on the class "UnitPriceSpecification"

fying the currency value (from "29.99" to "27.25") and the price type (from "SRP" to "SP") of the game offering having the ID "9223752" (see Fig. 14).

Since the conventional ontology schema (i.e., GameSaleSchema_V1.owl) and the ontology annotation document (i.e., GameSaleAnnotations_V1.xml) are not changed, the temporal ontology schema (i.e., GameSaleTemporalSchema.xml) is consequently not updated. However, the system updates the temporal document, to include the new slice of the new conventional ontology document, as shown in Fig. 15. The

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalOntologySchema>
 <conventionalOntologySchema>
  <sliceSequence>
   <slice location="GameSaleSchema_V1.owl" begin="2015-06-15" />
  </sliceSequence>
 </conventionalOntologySchema>
 <ontologyAnnotationSet>
  <sliceSequence>
   <slice location="GameSaleAnnotations_V1.xml" begin="2015-06-15" />
  </sliceSequence>
 </ontologyAnnotationSet>
</temporalOntologySchema>
```

Fig. 10 Temporal ontology schema on June 15, 2015

```
<?xml version="1.0" encoding="UTF-8"?>
 <td:temporalRoot temporalSchemaLocation="GameSaleTemporalSchema.xml" />
  <td:sliceSequence>
   <td:slice location="GameSales_V1.rdf" begin="2015-06-15" />
  </td:sliceSequence>
 </td:temporalRoot>
```

Fig. 11 Temporal document on June 15, 2015

```
…
 <gr:Offering rdf:ID="Offering_9223752">
  <gr:hasPriceSpecification>
   <gr:UnitPriceSpecification rdf:ID="UnitPriceSpecification_9223752_1">
    <gr:hasCurrency>USD</gr:hasCurrency>
    <hasCurrencyValue_RepItem>
     <hasCurrencyValue_Version>
      <timestamp_ValidExtent begin="2015-06-15" end="now" />
      <gr:hasCurrencyValue>29.99</gr:hasCurrencyValue>
     </hasCurrencyValue_Version>
    </hasCurrencyValue_RepItem>
    <priceType_RepItem>
     <priceType_Version>
      <timestamp_ValidExtent begin="2015-06-15" end="now" />
      <gr:priceType>SRP</gr:priceType>
     </priceType_Version>
    </priceType_RepItem>
    …
   </gr:UnitPriceSpecification>
  </gr:hasPriceSpecification>
  …
 </gr:Offering>
…
```

Fig. 12 Squashed document correponding to the temporal document on June 15, 2015

squashed version of the updated temporal document is provided in Fig. 16.

Obviously, each one of the squashed documents (see Figs. 12 and 16) should conform to the representational schema (box 13), which is generated (by the Representational Schema Generator) from the temporal ontology schema shown in Fig. 10.

The example will be completed in Sect. 4, after that the management of schema changes has been introduced.

4 Management of Temporal Schema Versioning in τOWL

In this section, we present our approach for managing temporal schema versioning in τOWL. First, we briefly describe how the conventional ontology schema and ontology annotation documents are versioned in the τOWL framework. Then, we propose primitives for changing the conventional and temporal schema. Next, we focus on the schema versioning support in our τOWL-Manager tool. Finally, we resume our running example to show the use of the proposed primitives for versioning conventional ontology, while providing some screenshots which illustrate the use of τOWL-Manager for the same purpose.

4.1 Temporal Schema Versioning Process

In our approach, the schema versioning process starts with the creation of a first schema version: the KBA defines a new conventional ontology schema (which is stored as an

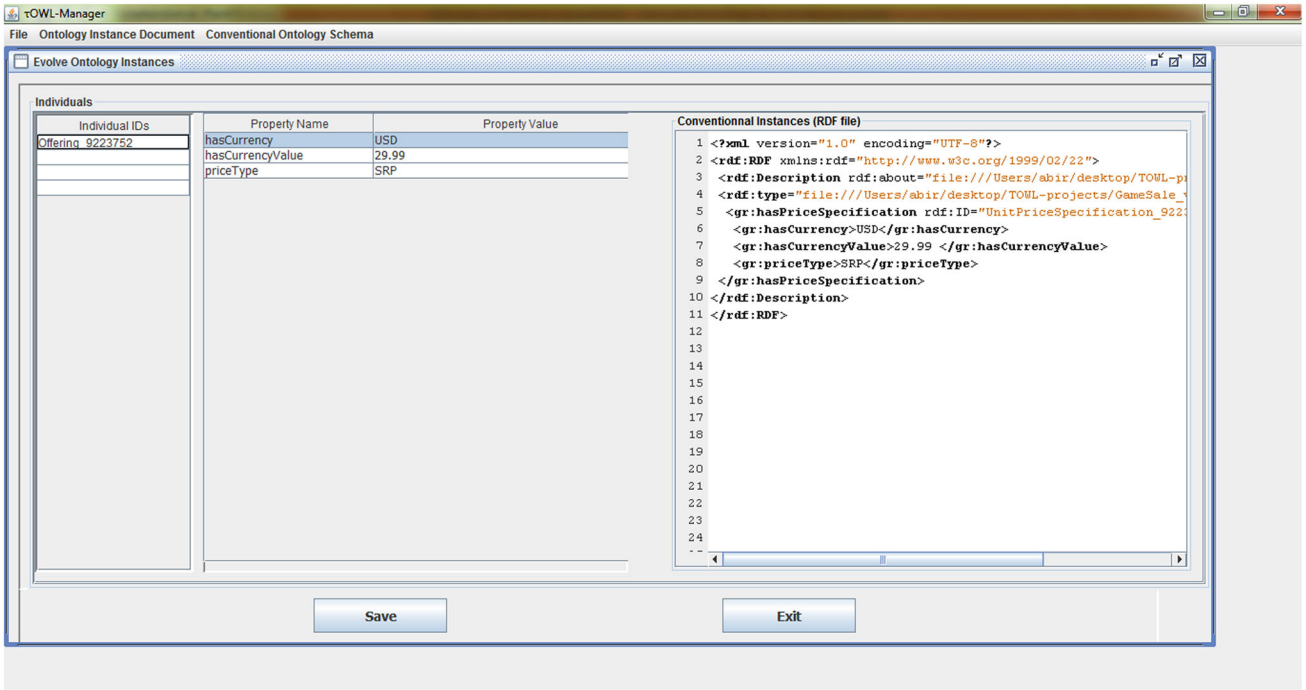

Fig. 13 Showing the chosen conventional ontology instance document version

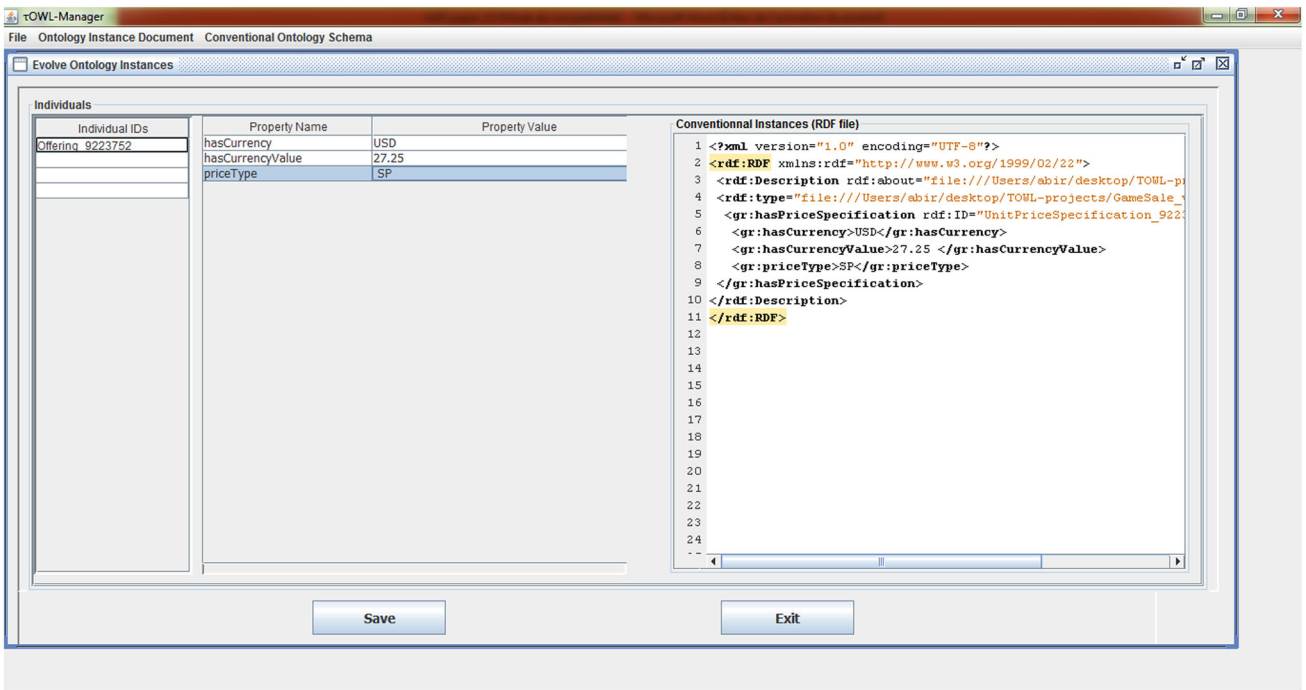

Fig. 14 Changing the chosen conventional ontology instance document version (modifying the currency value and the price type)

OWL 2 file) and annotates it with some logical and physical annotations (which are stored as an XML file). Consequently, the system creates the temporal ontology schema (also stored as an XML file) that ties together the conventional schema and the annotation document.

After that, when necessary, the KBA can independently change the conventional ontology schema or the annotation document. Changes to the conventional ontology schema give rise to a new version of it. Similarly, changes to the logical or physical annotations gives rise to a new version

```
<?xml version="1.0" encoding="UTF-8"?>
 <td:temporalRoot temporalSchemaLocation="GameSaleTemporalSchema.xml" />
  <td:sliceSequence>
   <td:slice location="GameSales_V1.rdf" begin="2015-06-15" />
   <td:slice location="GameSales_V2.rdf" begin="2015-07-08" />
  </td:sliceSequence>
 </td:temporalRoot>
```

Fig. 15 Temporal document on July 08, 2015

```
…
 <gr:Offering rdf:ID="Offering_9223752">
  <gr:hasPriceSpecification>
   <gr:UnitPriceSpecification rdf:ID="UnitPriceSpecification_9223752_1">
    <gr:hasCurrency>USD</gr:hasCurrency>
    <hasCurrencyValue_RepItem>
     <hasCurrencyValue_Version>
      <timestamp_ValidExtent begin="2015-06-15" end="2015-07-07" />
      <gr:hasCurrencyValue>29.99</gr:hasCurrencyValue>
     </hasCurrencyValue_Version>
     <hasCurrencyValue_Version>
      <timestamp_ValidExtent begin="2015-07-08" end="now" />
      <gr:hasCurrencyValue>27.25</gr:hasCurrencyValue>
     </hasCurrencyValue_Version>
    </hasCurrencyValue_RepItem>
    <priceType_RepItem>
     <priceType_Version>
      <timestamp_ValidExtent begin="2015-06-15" end="2015-07-07" />
      <gr:priceType>SRP</gr:priceType>
     </priceType_Version>
     <priceType_Version>
      <timestamp_ValidExtent begin="2015-07-08" end="now" />
      <gr:priceType>SP</gr:priceType>
     </priceType_Version>
    </priceType_RepItem>
    …
   </gr:UnitPriceSpecification>
  </gr:hasPriceSpecification>
  …
 </gr:Offering>
…
```

Fig. 16 Squashed document correponding to the temporal document on July 08, 2015

of the entire ontology annotation document. Moreover, the temporal ontology schema is automatically updated by the system after each transaction which includes changes applied by the KBA to the conventional schema and/or to the annotation document.

4.2 Schema Change Operations

τOWL is based on the OWL 2 language [35], 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. Therefore, we consider that the signature of an OWL 2 ontology O can be defined as follows: O = {E, A, Exp} such that:

1. E = {C, DP, OP, AP} represents the set of the entities with: (1) C: Class, represents the set of concepts; (2) DP: Data Property, represents the set of properties of the concepts; (3) OP: Object Property, represents the set of the semantic relations between the concepts; and (4) AP: Annotation Property, represents the set of annotations on the entities and those on the axioms.
2. A = {EAx, KAx} represents the set of axioms with: (1) EAx: Entity Axioms, represents the axioms which concern the entities; and (2) KAx: Key Axioms, represents all the identifiers associated to the various classes.
3. Exp = {CE, OPE, DPE} represents the set of the used expressions (an expression is a complex description which results from combinations of entities using constructors, such as enumeration, restriction of cardinality, and restriction of properties) with: (1) CE: Class Expressions, represents the set of combinations of concepts by using constructors; (2) OPE: Object Property Expressions, represents the set of combinations of relations; and (3) DPE: Data Property Expressions, represents the set of combinations of properties.

Based on this OWL 2 ontology definition, we propose a complete set of 28 primitives for changing a conventional

ontology schema. The idea is that each primitive deals with an OWL 2 ontology component (e.g., a class, a data property, and an object property), by creating, removing, or modifying such a component. These primitives are as follows (more detailed description of their operational semantics showing their effects on the COS can be found in [41,42,44]):

- **CreateConventionalOntologySchema**(COS.owl)
- **RenameConventionalOntologySchema**(oldCOS, newCOS)
- **DropConventionalOntologySchema**(COS.owl)
- **AddClass**(COS.owl, className)
- **RenameClass**(COS.owl, oldClassName, newClassName)
- **DropClass**(COS.owl, className)
- **AddDataProperty**(COS.owl, className, DataPropertyName, DataPropertyType)
- **DropDataProperty**(COS.owl, className, DataPropertyName)
- **RenameDataProperty**(COS.owl, className, oldDataPropertyName, newDataPropertyName)
- **ChangeDataPropertyDomain**(COS.owl, className, DataPropertyName, newDataPropertyDomain)
- **ChangeDataPropertyRange**(COS.owl, className, DataPropertyName, oldDataPropertyRange, newDataPropertyRange)
- **AddObjectProperty**(COS.owl, ObjectPropertyName, ObjectPropertyDomain, ObjectPropertyRange)
- **DropObjectProperty**(COS.owl, ObjectPropertyName)
- **RenameObjectProperty**(COS.owl, oldObjectPropertyName, newObjectPropertyName)
- **ChangeObjectPropertyDomain**(COS.owl, ObjectPropertyName, oldObjectPropertyDomain, newObjectPropertyDomain)
- **ChangeObjectPropertyRange**(COS.owl, ObjectPropertyName, oldObjectPropertyRange, newObjectPropertyRange)
- **AddAnnotationProperty**(COS.owl, propertyType, propertyName, annotationProperty)
- **DropAnnotationProperty**(COS.owl, propertyType, propertyName, annotationProperty)
- **ChangeAnnotationProperty**(COS.owl, propertyType, propertyName, oldAnnotationProperty, newAnnotationProperty)
- **AddEntityAxiom**(COS.owl, entityType, entityName, entityAxiom)
- **DropEntityAxiom**(COS.owl, entityType, entityName, entityAxiom)
- **ChangeEntityAxiom**(COS.owl, entityType, entityName, oldEntityAxiom, newEntityAxiom)
- **AddKeyAxiom**(COS.owl, className, keyAxiom)
- **DropKeyAxiom**(COS.owl, className, keyAxiom)
- **ChangeKeyAxiom**(COS.owl, className, oldKeyAxiom, newKeyAxiom)
- **AddEntityExpression**(COS.owl, entityType, entityName, entityExpression)
- **DropEntityExpression**(COS.owl, entityType, entityName, entityExpression)
- **ChangeEntityExpression**(COS.owl, entityType, entityName, oldEntityExpression, newEntityExpression)

Furthermore, since 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 a complete set of four primitives acting on a temporal ontology schema. These primitives are as follows (more details on them can be found in [41,42,44]):

- **CreateTemporalOntologySchema** (TOS.xml)
- **DropTemporalOntologySchema**(TOS.xml)
- **AddSlice** (TOS.xml, toWhat, sourceSlice, targetSlice)
- **DropSlice** (TOS.xml, fromWhat, targetSlice)

4.3 Schema Versioning Support in τOWL-Manager

The overall architecture of τOWL-Manager is shown in Fig. 17. It is composed of three layers: presentation layer, business layer, and storage layer.

The presentation layer includes an interface for constructing temporal ontology schema, an interface for creating and updating ontology instances, and an interface for changing temporal ontology schema.

To construct a new temporal ontology schema, the KBA has to start by providing a reference to an existing valid conventional ontology schema (definition of an ontology schema from scratch is not supported in the current version of τOWL-Manager), e.g., the GoodRelation ontology as in our running example. Then, the KBA annotates the new conventional ontology schema by some logical and physical annotations (as shown in Figs. 8 and 9).

The business layer contains three modules: the first for constructing and validating temporal ontology schema, named "Temporal Ontology Schema Construction Manager", the second for creating/updating and validating ontology instances, named "Ontology Instance Document Manager", and the third for evolving temporal ontology schema, named "Temporal Ontology Schema Change Manager".

The "Temporal Ontology Schema Construction Manager" first generates the files corresponding to the temporal ontology schema, that is the conventional schema file and the annotation document file, from the specifications expressed by the KBA in its interface. Then, it checks the validity of the

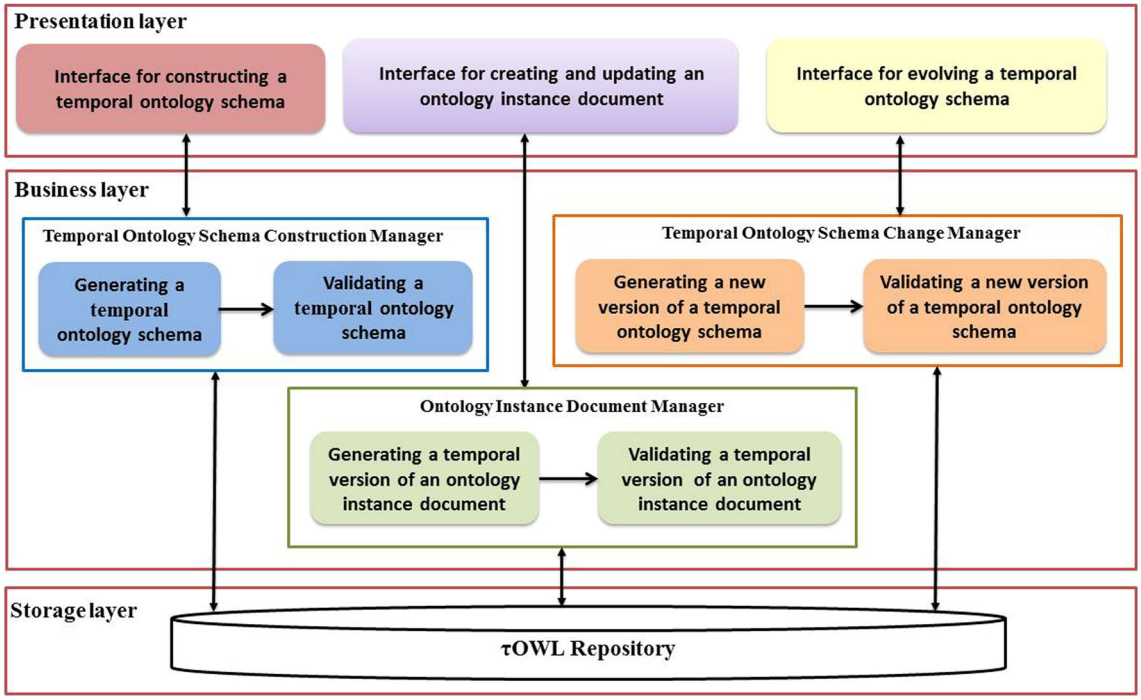

Fig. 17 Architecture of τOWL-Manager

generated files and creates the temporal schema file, which ties together the two other files.

The "Ontology Instance Document Manager" allows creating and versioning ontology instance documents. Figure 6 shows an ontology instantiation; Figs. 13 and 14 show an example of keeping track of ontology instance changes.

The "Temporal Ontology Schema Change Manager" first generates the file corresponding to the new conventional ontology schema version (i.e., a new OWL 2 file) and/or the file corresponding to the new ontology annotation document (i.e., a new XML file), from schema change operations specified by the KBA in its schema change interface. Then, it (1) checks the validity of the generated file(s), (2) adds it (or them) to the τOWL repository, (3) updates the temporal schema file to take into account the new version of the conventional schema version and/or the new version of the ontology annotation document, (4) propagates the effects of schema changes on existing ontology instance document versions to make them valid with regard to the new conventional ontology schema version, and (5) updates the temporal document to make reference to the new conventional ontology instance document versions. Some screenshots, which show the use of τOWL-Manager for managing schema changes, will be provided within the next subsection that illustrates temporal schema versioning in τOWL through our running example.

The storage layer contains the repository of resources making up temporal ontologies and associated instances, named τOWL Repository.

4.4 Running Eample Reprise

Let us resume the example, started in Sect. 2 and continued in Sect. 3.4, to show how temporal schema versioning is carried out in our context. Suppose that on July 22, 2015, the KBA decides to make some changes to the first version of the conventional ontology schema, to meet some changes in the code of the application that exploit such an ontology schema. These changes are as follows:

- add a new class, named "Invoice", having the following properties (or data properties): ID (String), date (Date), and total amount (float);
- add a new relationship (or object property), named "hasInvoice", between the class "Offering" and the new class "Invoice";
- specify that the price type of an offering must be either "SP", or "SRP";
- specify that the relationship "hasPriceSpecification" is irreflexive;
- specify an expression on the relationship "hasPriceSpecification", which indicates that each offering must have at least one price specification.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:gr="http://purl.org/goodrelations/v1#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  …
  <owl:Class rdf:about="http://purl.org/goodrelations/v1#Offering">
    <owl:ObjectProperty rdf:about="http://purl.org/goodrelations/v1#hasPriceSpecification">
      <rdfs:range rdf:resource="http://purl.org/goodrelations/v1#PriceSpecification"/>
        …
        <rdfs:domain>
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <rdf:Description rdf:about="http://purl.org/goodrelations/v1#Offering"/>
              <rdf:Description rdf:about="http://schema.org/Offer"/>
            </owl:unionOf>
          </owl:Class>
        </rdfs:domain>
        <rdfs:isDefinedBy rdf:resource="http://purl.org/goodrelations/v1"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="http://purl.org/goodrelations/v1#hasInvoice">
      <rdfs:range rdf:resource="http://purl.org/goodrelations/v1#Invoice"/>
      <rdfs:domain rdf:resource="http://purl.org/goodrelations/v1#Offering"/>
    </owl:ObjectProperty>
    …
  </owl:Class>
  <owl:Class rdf:about="http://purl.org/goodrelations/v1#Invoice">
    <owl:DataProperty rdf:about="http://purl.org/goodrelations/v1#ID">
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
      <rdfs:domain rdf:resource="http://purl.org/goodrelations/v1#Invoice"/>
    </owl:DataProperty>
    <owl:DataProperty rdf:about="http://purl.org/goodrelations/v1#date">
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#Date"/>
      <rdfs:domain rdf:resource="http://purl.org/goodrelations/v1#Invoice"/>
    </owl:DataProperty>
    <owl:DataProperty rdf:about="http://purl.org/goodrelations/v1#totalAmount">
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
      <rdfs:domain rdf:resource="http://purl.org/goodrelations/v1#Invoice"/>
    </owl:DataProperty>
  </owl:Class>
  <DatatypeDefinition>
    <Datatype IRI="priceType"/>
    <DataOneOf>
      <Literal datatypeIRI="http://www.w3.org/2001/XMLSchema#String">SP</Literal>
      <Literal datatypeIRI="http://www.w3.org/2001/XMLSchema#String">SRP</Literal>
    </DataOneOf>
  </DatatypeDefinition>
  <owl:IrreflexiveProperty rdf:about="hasPriceSpecification"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasPriceSpecification"/>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1
    </owl:minCardinality>
  </owl:Restriction>
  …
</rdf:RDF>
```

Fig. 18 Second version of the conventional ontology schema (GameSaleSchema_V2.owl), on July 22, 2015

The second version of the conventional ontology schema and the second version of each one the two conventional ontology instance documents are shown in Figs.18, 21, and 22, respectively. Figures 19 and 20 show the addition of the new class "Invoice" and the definition of the new relationship "hasInvoice" between the two classes "Offering" and "Invoice", respectively, through the use of τOWL-Manager. The temporal ontology schema is also updated by adding a new slice related to the new version of the conventional ontology schema, as shown in Fig. 23. Moreover, the temporal document is updated, to include two new slices corresponding to the two new conventional ontology instance documents, as shown in Fig. 24. 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 Figs. 12 and 16. Notice that changes are presented in red bold type, in Figs. 18, 21, 22, 23, and 24.

The sequence of primitives that have been specified by the KBA and performed by the "Temporal Ontology Schema Change Manager" (see Fig. 17) on the temporal ontology schema (GameSaleTemporalSchema.xml, Fig. 10), on the first version of the conventional ontology schema (GameSaleSchema_V1.owl, Fig. 4), on the first version of the conventional ontology instance document (GameSales_V1.rdf,

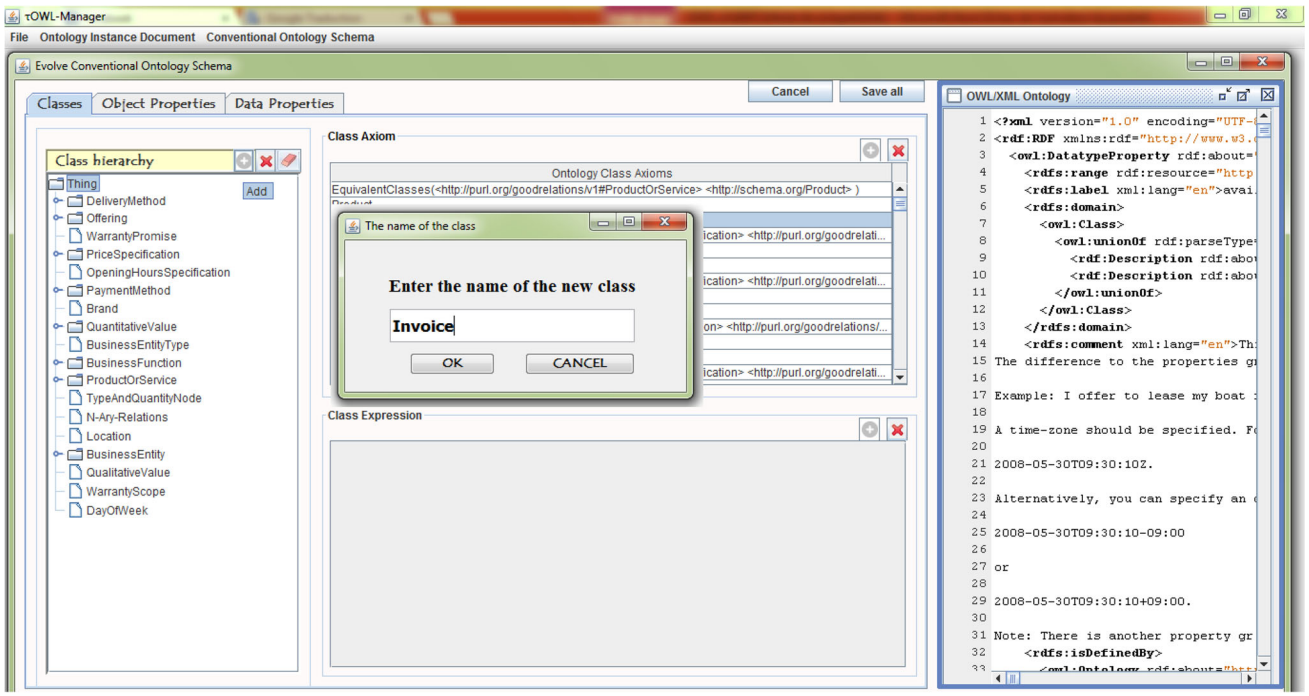

Fig. 19 Adding the new class "Invoice" to the conventional ontology schema

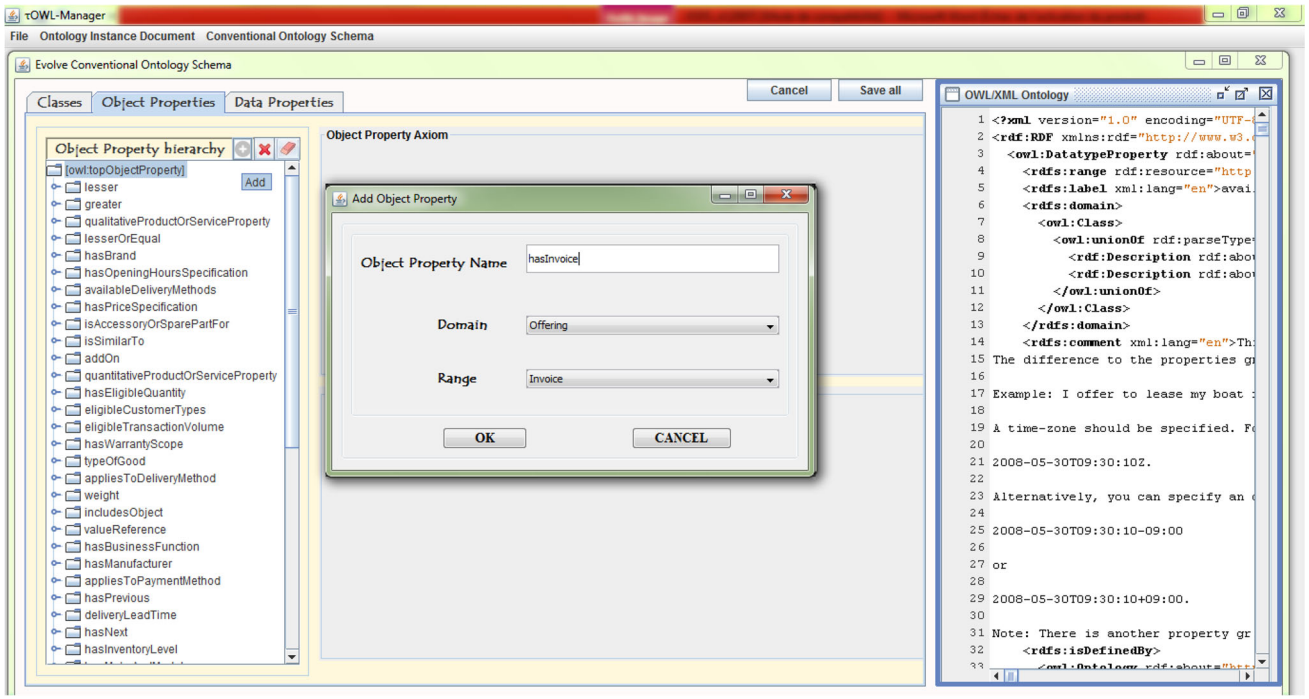

Fig. 20 Adding the new relationship "hasInvoice" between the class "Offering" and the class "Invoice"

Fig. 1) and on the second version of the conventional ontology instance document (GameSales_V2.rdf, Fig. 2), to update the temporal ontology schema (see Figure 23) and the temporal document (see Fig. 24) and to produce the second version of the conventional ontology schema (GameSaleSchema_V2.owl, Fig. 18), the third version of the conventional ontology instance document (GameSales_V3.rdf, Fig. 21), and the fourth version of the conventional ontol-

```
…
  <gr:Offering rdf:ID="Offering_9223752">
    <gr:hasPriceSpecification>
      <gr:UnitPriceSpecification rdf:ID="UnitPriceSpecification_9223752_1">
        <gr:hasCurrency>USD</gr:hasCurrency>
        <gr:hasCurrencyValue>29.99</gr:hasCurrencyValue>
        <gr:priceType>SRP</gr:priceType>
        …
      </gr:UnitPriceSpecification>
    </gr:hasPriceSpecification>
    <hasInvoice/>
    …
  </gr:Offering>
  <Invoice>
    <ID/>
    <date/>
    <totalAmount/>
  </Invoice>
…
```

Fig. 21 GameSales_V3.rdf": the second version of the conventional ontology instance document "GameSales_V1.rdf", on July 22, 2015

```
…
  <gr:Offering rdf:ID="Offering_9223752">
    <gr:hasPriceSpecification>
      <gr:UnitPriceSpecification rdf:ID="UnitPriceSpecification_9223752_1">
        <gr:hasCurrency>USD</gr:hasCurrency>
        <gr:hasCurrencyValue>27.25</gr:hasCurrencyValue>
        <gr:priceType>SP</gr:priceType>
        …
      </gr:UnitPriceSpecification>
    </gr:hasPriceSpecification>
    <hasInvoice/>
    …
  </gr:Offering>
  <Invoice>
    <ID/>
    <date/>
    <totalAmount/>
  </Invoice>
…
```

Fig. 22 GameSales_V4.rdf": the second version of the conventional ontology instance document "GameSales_V2.rdf", on July 22, 2015

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalOntologySchema>
  <conventionalOntologySchema>
    <sliceSequence>
      <slice location="GameSaleSchema_V1.owl" begin="2015-06-15" />
      <slice location="GameSaleSchema_V2.owl" begin="2015-07-22" />
    </sliceSequence>
  </conventionalOntologySchema>
  <ontologyAnnotationSet>
    <sliceSequence>
      <slice location="GameSaleAnnotations_V1.xml" begin="2015-06-15" />
    </sliceSequence>
</ontologyAnnotationSet>
</temporalOntologySchema>
```

Fig. 23 Temporal ontology schema (GameSaleTemporalSchema.xml), on July 22, 2015

```
<?xml version="1.0" encoding="UTF-8"?>
  <td:temporalRoot temporalSchemaLocation="GameSaleTemporalSchema.xml" />
    <td:sliceSequence>
      <td:slice location="GameSales_V1.rdf" begin="2015-06-15" />
      <td:slice location="GameSales_V2.rdf" begin="2015-07-08" />
      <td:slice location="GameSales_V3.rdf" begin="2015-07-22" />
      <td:slice location="GameSales_V4.rdf" begin="2015-07-22" />
    </td:sliceSequence>
</td:temporalRoot>
```

Fig. 24 Temporal document (GameSaleTemporalDocument.xml), on July 22, 2015

ogy instance document (GameSales_V4.rdf, Fig. 22), which are valid to "GameSaleSchema_V2.owl" (see Fig. 18), could make up the following transaction:

```
Begin Transaction
(i) AddSlice("GameSaleTemporalSchema.xml", conventionalOntologySchema, current,
     "GameSaleSchema_V2.owl")
(ii) AddClass("GameSaleSchema_V2.owl", "Invoice")
(iii) AddDataProperty("GameSaleSchema_V2.owl", "Invoice", "ID",
     "http://www.w3.org/2001/XMLSchema#String")
(iv) AddDataProperty("GameSaleSchema_V2.owl", "Invoice", "date",
     "http://www.w3.org/2001/XMLSchema#Date")
(v) AddDataProperty("GameSaleSchema_V2.owl", "Invoice", "totalAmount",
     "http://www.w3.org/2001/XMLSchema#float")
(vi) AddObjectProperty("GameSaleSchema_V2.owl", "hasInvoice", "Offering", "Invoice")
(vii) AddEntityAxiom("GameSaleSchema_V2.owl", ObjectProperty, "hasPriceSpecification",
     "IrreflexiveProperty")
(viii) AddEntityExpression("GameSaleSchema_V2.owl", ObjectProperty, "hasPriceSpecification",
     "minCardinality(1)")
Commit
```

Notice that the transaction time associated to the execution of the transaction above is July 22, 2015, which is used by the system 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 also that "GameSales_V3.rdf" and "GameSales _V4.rdf" are the results of the effects of schema changes on instances (i.e., the results of schema change propagation), to adapt all existing instances, stored in "GameSales_V1.rdf" and "GameSales_V2.rdf", to the new schema version "GameSaleSchema_V2.owl". Indeed, after creating "GameSales_V3.rdf" as a copy of "GameSales_V1.rdf" and "GameSales_V4.rdf" as a copy of "GameSales_V2.rdf", the two following XQuery Update Facility [34] statements could be executed on "GameSales_V3.rdf" and "GameSales_V4.rdf", respectively, to achieve the purpose:

```
for $p in fn:doc("GameSales_V3.rdf")//gr:Offering
return (
   insert node <hasInvoice/> after $p/gr:hasPriceSpecification,
   insert node <Invoice> <ID/> <date/> <totalAmount/> </Invoice> after $p
)
for $p in fn:doc("GameSales_V4.rdf")//gr:Offering
return (
   insert node <hasInvoice/> after $p/gr:hasPriceSpecification,
   insert node <Invoice> <ID/> <date/> <totalAmount/> </Invoice> after $p
)
```

These two XQuery Update Facility statements must actually be automatically derived by the "Temporal Ontology Schema Change Manager" as a part of the semantics of the schema change primitives. It is not part of what the KBA puts in his/her schema change transaction, but the system generates and adds it to the transaction that is actually executed.

To recap, on July 22, 2015, our τOWL repository is thus composed of the following nine resources: two successive versions of the conventional ontology schema (shown in Figs. 4 and 18, respectively), four versions of the conventional ontology instance document (shown in Figs. 1, 2, 21, and 22, respectively), one version of the ontology annotation document (shown in Fig. 7), the temporal document (shown in Fig. 24), and the temporal ontology schema (shown in Fig. 23).

5 Related Work Discussion

Time dimension(s) are explicitly added to Semantic Web languages and formalisms (e.g., RDF, OWL, and SPARQL) to represent time in semantic annotations, to build temporal ontologies, and to support temporal querying and reasoning. Furthermore, schema versioning is being integrated into the Semantic Web technologies for historical and legal reasons. An annotated bibliography of the previous work on these topics is presented in Grandi [16] and an excellent recent survey on ontology evolution and versioning approaches can be found in Zablith et al. [38]. Thus, several works have been proposed in the literature to manage temporal ontology instance versioning and/or temporal ontology schema versioning.

As for temporal ontology instance versioning, there are various contributions that propose to represent and manage temporal data in the Semantic Web.

In Gutiérrez et al. [20], the authors present a comprehensive framework to incorporate temporal reasoning into RDF, yielding temporal RDF graphs. They define a syntactic notion of temporal RDF graphs. Furthermore, [1] propose CHRONOS, a powerful system for reasoning over temporal information in OWL ontologies. 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" and"after" for temporal relations) and quantitative information (i.e., where temporal information is defined precisely, e.g., using dates). With regard to those approaches, we are not interested in temporal reasoning (and, thus, in spatio-temporal reasoning). In our present approach, reasoning facilities can be added as an external stratum on top of the τOWL framework: although no time-related concepts can be defined within an ontology version, temporal reasoning could be supported by combining concepts present in different ontology versions (as retrieved by our system) taking into account the timestamps of the versions.

Moti [26] 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. In Zamborlini et al. [39], two complementary proposals for modeling temporally changing information in OWL are presented. They are based on the perdurantist theory and benefit from results coming from the discipline of Formal Ontology, to restrict the appropriate use of the proposed frameworks. With regard to [26] and [39], our approach does not deal with modeling of time inside the ontology. It just supports temporal versioning.

In O'Connor and Das [27], the authors 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. Milea et al. [25], the authors 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: (1) Concrete Domains, allowing the representation of restrictions using concrete domain binary predicates, (2) Temporal Representation, introducing timepoints, relations between timepoints, intervals, and Allen's 13 interval relations into the language, and (3) TimeSlices/Fluents, implementing a perdurantist view on individuals and enabling the representation of complex temporal aspects, such as process state transitions. The main purpose of our approach is simply the support of the past ontology versions, which could 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, for instance, if a third party has to resolve now a controversy on some item sold online, it must be able to individuate the offer details valid at the time of the purchase even if they have been changed thereafter.

Notice that in [37], the authors present the annotation features of OWL 2 by showing that this latter allows for annotations on ontologies, entities, axioms, and so on. 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 and suite of tools).

As for ontology schema versioning, there are also several work which have dealt with such an issue [2,12,13,15,17,18,21,24,29,31,40–44].

Heflin and Pan [21] 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 and Groza [31] 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 and Marwaha [2] introduce an approach which 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 and Plexousakis [24] 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, which are more strictly related with our approach, are [12,13,15,17,18,29,40–44].

Grandi [12] 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 to manage temporal versions of an ontology. Grandi [15] focuses on temporal versioning of light-weight ontologies expressed in RDF(S) and show how the multi-temporal RDF data model proposed in Grandi [12] 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. Similarly to [12] and [15], our present work handles schema versioning of temporal ontologies but in a different temporal Semantic Web framework. In particular, whereas such approaches introduce ad hoc solutions requiring an extension of the Semantic Web standards, our proposal relies on the current OWL 2 recommendation and is fully compliant with off-the-shelf Semantic Web standards and available ontology specifications.

In [13], the authors introduce "The Valid Ontology" approach as a temporal extension of OWL. Indeed, they propose to use a single temporal XML document to represent and store a multi-version ontology and use a temporal XML query processor to efficiently extract valid OWL ontologies from the XML document as temporal snapshots. The result is an efficient ontology temporal versioning solution, relying on the standard XML technology. Different from the τOWL principled and flexible approach, "The Valid Ontology" consists of an *ad hoc* solution, with a fixed version encoding and timestamping scheme, without logical and physical independence support, and without the provision of supporting tools.

Grandi [17] proposes a storage scheme based on a temporal relation which can be used to represent and manage the class structure of a multi-version ontology (embodying a tree-shaped class hierarchy) in a temporal relational database. Furthermore, the author provides the definition of primitive operations which can be used for the maintenance of a multi-version ontology in such a framework. Grandi [18] extends this work by considering ontologies with a class hierarchy structured as a general directed graph, that is also supporting multiple inheritance and intersection classes, and showing how multi-version ontologies must be dealt with for the processing of ontology-based personalization queries. Contrarily to [17] and [18], we deal with the temporal multi-version management of a fully fledged ontology, not only of its class hierarchy. Therefore, our approach is much more general, as it extends to multi-version temporal management all kinds of Semantic Web applications where ontologies are needed, not only ontology-based personalization.

Jaziri et al. [23] provide an approach for managing ontology versioning. The authors state that their approach (1) keeps track of ontology versions and applied changes, (2) creates links between versions, and (3) maintains coherence (which means here structural consistency) of ontology versions. Notice that according to this approach, in some cases, a schema change applied to an ontology version does not lead to a new ontology version, but updates such a version with a destructive manner. In our approach, when applying a sequence of schema change operations to a given ontology schema version (always the current one), a new schema version will be created and the "evolved" one is kept unchanged. As for linking ontology versions, our approach links ontology schema versions automatically and easily through the use of the <slice/> elements in the temporal ontology schema; ontology instance document versions are also linked by <slice/> elements of the temporal document. As for structural consistency, our approach guarantees that each newly created ontology schema/instance version is structurally correct.

Sassi et al. [29] focus on the relevance of ontology versions produced by changes, which is evaluated according to four criteria (i.e., conceptualization, usage frequency, abstraction, and completeness), and introduce an "optimisation process" to reduce the number of versions to retain: when a maximum is reached, the least relevant versions are deleted. The maximum number of versions and the selection of relevance criteria to be applied are managed by the tool user. In our current work, we deal neither with ontology version relevance, nor with reducing the number of ontology schema versions.

Zekri et al. [40] introduce 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. This framework ensures logical and physical data independence, since it (1) separates conventional schema, logical annotations, and physical annotations and (2) allows each one of these three components to be changed independently and safely. The present work extends

[40] by (1) proposing a general approach for schema versioning in τOWL and (2) focusing on the proposition of a set of change primitives for supporting the evolution of both temporal and conventional ontology schema. Our approach helps KBAs in the management of the conventional schema changes in τOWL-based Semantic Web repositories and guarantees the maintenance of a full history of evolving conventional ontology instances and schemata. Zekri et al. [41] extend the work presented in [40] by showing how, in the τOWL framework, temporal ontology instance versioning could be managed in the presence of temporal ontology schema versioning, simultaneously and in a consistent manner. In [42], the authors propose two complete sets of schema change primitives, one for changing conventional ontology schema and the other for updating temporal ontology schema, in the τOWL context. Zekri et al. [44] propose an approach, which extends the contribution of [42], for managing time-varying knowledge, since an ontology can be used for knowledge management. Zekri et al. [43] present τOWL-Manager, a prototype tool for defining temporal ontology schema and managing temporal versioning of temporal ontology instances, in the τOWL framework. Our present work extends [43] by adding schema versioning support to τOWL-Manager and showing its functionalities.

6 Conclusion

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. τOWL ensures logical and physical data independence, since it (1) separates conventional schema, logical annotations, and physical annotations and (2) allows each one of these three components to be changed independently and safely. Furthermore, adopting τOWL provides for a low-impact solution, since it requires neither modifications of existing Semantic Web documents, including ontology specifications, 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).

Furthermore, we have extended τOWL to support temporal schema versioning, by introducing an approach for defining and evolving temporal ontologies, proposing two complete set of primitives for changing both conventional and temporal ontology schema, and illustrating such an approach and such primitives through a running example. To demonstrate the feasibility of our τOWL approach, we have also proposed τOWL-Manager, a tool for the management of temporal ontologies. It supports temporal versioning of both ontology instances and ontology schema. It allows the KBA to (1) construct and change temporal ontology schema and (2) create and update ontology instance documents, within a schema versioning context. Our approach and our tool assist the KBA in his/her tasks of specifying and maintaining conventional ontology schema in τOWL-based repositories. They also allow obtaining a complete history of both conventional ontology schema and instances.

Currently, we are extending the present work by (1) 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 and (2) completing the development of τOWL-Manager to support all schema change primitives proposed in this paper and also the new primitives for changing ontology annotation documents. In the next future, we plan to propose high-level schema change operations, since they are more user-friendly that schema change primitives; a high-level operation is a valid and optimized sequence of primitives, which correspond to frequent schema evolution needs and allows expressing complex changes in a more compact way [6]. Moreover, we plan to deeply study conventional ontology schema change propagation and how the system could generate automatically the optimized list of XQuery Update Facility statements associated to every schema change transaction. Finally, we intend to study querying instances of τOWL ontologies, under schema versioning. We could start from T-SPARQL language [14], which allows expressing temporal queries on multi-temporal RDF triples in an environment which contains a single conventional ontology schema version, and extend it with necessary features to support queries involving several schema versions [16].

References

1. Anagnostopoulos E, Batsakis S, Petrakis EGM (2013) CHRONOS: a reasoning engine for qualitative temporal information in OWL. In: Proceedings of the 17th international conference in knowledge-based and intelligent information and engineering systems (KES 2013), Kitakyushu, Japan, 9–11 September, pp 70–77
2. Bedi P, Marwaha S (2007) Versioning OWL ontology using temporal tags. In: Proceedings of the 21st international conference on computer, electrical, systems science and engineering (CESSE'07), Vienna, Austria, 25–27 May, pp 332–337
3. Berners-Lee T, Cailliau R, Luotonen A, Nielsen HF, Secret A (1994) The World Wide Web. Commun ACM 37(8):76–82
4. Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. Sci Am 284(5):34–43
5. Brahmia Z, Grandi F, Oliboni B, Bouaziz R (2014a) Schema change operations for full support of schema versioning in the τXSchema framework. Int J Inf Technol Web Eng 9(2):20–46
6. Brahmia Z, Grandi F, Oliboni B, Bouaziz R (2014b) High-level operations for changing temporal schema, conventional schema

and annotations, in the τXSchema framework. Technical Report TR-96, TimeCenter, 56 pp, January. http://timecenter.cs.aau.dk/TimeCenterPublications/TR-96.pdf. **(retrieved: April, 2016)**
7. Brahmia Z, Grandi F, Oliboni B, Bouaziz R (2015) Schema versioning. In: Khosrow-Pour M (ed) Encyclopedia of information science and technology, 3rd edn. IGI Global, Hershey, PA, pp 7651–7661. doi:10.4018/978-1-4666-5888-2.ch754
8. Burns T, Fong E, Jefferson D, Knox R, Mark L, Reedy C et al (1986) Reference model for DBMS standardization, database architecture framework task group (DAFTG) of the ANSI/X3/SPARC database system study group. SIGMOD Rec 15(1):19–58
9. Currim F, Currim S, Dyreson CE, Snodgrass RT (2004) A tale of two schemas: creating a temporal XML schema from a snapshot schema with tXSchema. In: Proceedings of the 9th international conference on extending database technology (EDBT) 2004, Heraklion, Crete, Greece, 14–18 March, pp 348–365
10. Dyreson CE, Grandi F (2009) Temporal XML. In: Liu L, Özsu MT (eds) Encyclopedia of database systems. Springer, US, pp 3032–3035
11. Grandi F (2002) A relational multi-schema data model and query language for full support of schema versioning. In: Proceedings of SEBD 2002—national conference on advanced database systems, Isola d'Elba, Italy, 19–21 June, pp 323–336
12. Grandi F (2009) Multi-temporal RDF ontology versioning. In: Proceedings of the 3rd international workshop on ontology dynamics (IWOD 2009), Washington DC, USA, 26 October. CEUR workshop proceedings (CEUR-WS.org), vol 519. http://ceur-ws.org/Vol-519/grandi.pdf. (**retrieved: April, 2016**)
13. Grandi F, Scalas MR (2009) The valid ontology: a simple OWL temporal versioning framework. In: Proceedings of the 3rd international conference on advances in semantic processing (SEMAPRO 2009), Sliema, Malta, 11–16 October, pp 98–102
14. Grandi F (2010) T-SPARQL: a TSQL2-like temporal query language for RDF. In: Proceedings of the 1st international workshop on querying graph structured data (GraphQ 2010), Novi Sad, Serbia, 20 September, pp 21–30
15. Grandi F (2011) Light-weight ontology versioning with multi-temporal RDF schema. In: Proceedings of the 5th international conference on advances in semantic processing (SEMAPRO 2011), Lisbon, Portugal, 20–25 November, pp 42–48
16. Grandi F (2012) An annotated bibliography on temporal and evolution aspects in the semantic web. SIGMOD Rec 41(4):18–21
17. Grandi F (2013) Dynamic multi-version ontology-based personalization. In: Proceedings of the 2nd international workshop on querying graph structured data (GraphQ 2013), Genoa, Italy, 22 March, pp 224–232
18. Grandi F (2016) Dynamic class hierarchy management for multi-version ontology-based personalization. J Comput Syst Sci 82(1):69–90
19. Guarino N (ed) (1998) Formal ontology in information systems. IOS Press, Amsterdam
20. Gutiérrez C, Hurtado CA, Vaisman AA (2007) Introducing time into RDF. IEEE Trans Knowl Data Eng 19(2):207–218
21. Heflin J, Pan Z (2004) A model theoretic semantics for ontology versioning. In: Proceedings of the 3rd international semantic web conference (ISWC 2004), Hiroshima, Japan, 7–11 November, pp 62–76
22. Horridge M, Bechhofer S (2011) The OWL API: a Java API for OWL ontologies. Semant Web 2:11–21
23. Jaziri W, Sassi N, Gargouri F (2010) Approach and tool to evolve ontology and maintain its coherence. Int J Metadata Semant Ontol 5(2):151–166
24. Kondylakis H, Plexousakis D (2013) Ontology evolution without tears. J Web Semant 19:42–58
25. Milea V, Frasincar F, Kaymak U (2012) tOWL: a temporal web ontology language. IEEE Trans Syst Man Cybern Part B 42(1):268–281
26. Motik B (2010) Representing and querying validity time in RDF and OWL: a logic-based approach. In: Proceedings of the 9th international semantic web conference (ISWC 2010), Shanghai, China, 7–11 November, pp 550–565
27. O'Connor MJ, Das AK (2011) A method for representing and querying temporal information in OWL. Biomedical engineering systems and technologies, volume 127 of communications in computer and information science. Springer-Verlag, Heidelberg, Germany, pp 97–110
28. Rogozan D, Paquette G (2005) Managing ontology changes on the semantic web. In: Proceedings of the 2005 IEEE/WIC/ACM international conference on web intelligence (WI 2005), Compiegne, France, 19–22 September, pp 430–433
29. Sassi N, Jaziri W, Alharbi S (2015) Supporting ontology adaptation and versioning based on a graph of relevance. J Exp Theor Artif Intell. doi:10.1080/0952813X.2015.1056239
30. Snodgrass RT, Dyreson CE, Currim F, Currim S, Joshi S (2008) Validating quicksand: schema versioning in τXSchema. Data Knowl Eng 65(2):223–242
31. Völkel M, Groza T (2006) SemVersion: an RDF-based ontology versioning system. In: Proceedings of the IADIS international conference on WWW/Internet (ICWI 2006), Murcia, Spain, 5–8 October, vol 1, pp 195–202. http://www.xam.de/2006/10-SemVersion-ICIW2006.pdf. **(retrieved: April, 2016)**
32. W3C (2004a) XML schema part 0: primer second edition. W3C Recommendation, 28 October 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/. **(retrieved: April, 2016)**
33. W3C (2004b) RDF/XML syntax specification (Revised). W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. **(retrieved: April, 2016)**
34. W3C (2011) XQuery update facility 1.0. W3C Candidate Recommendation, 17 March 2011. http://www.w3.org/TR/2011/REC-xquery-update-10-20110317/. **(retrieved: April, 2016)**
35. W3C (2012a) OWL 2 web ontology language—primer (2nd Edition). W3C Recommendation, 11 December 2012. http://www.w3.org/TR/owl2-primer/. **(retrieved: April, 2016)**
36. W3C (2012b) OWL 2 web ontology language—document overview (2nd Edition). W3C Recommendation, 11 December 2012. http://www.w3.org/TR/owl2-overview/. **(retrieved: April, 2016)**
37. W3C (2012c) OWL 2 web ontology language—new features and rationale (2nd Edition). W3C Recommendation, 11 December 2012. http://www.w3.org/TR/owl2-new-features/. **(retrieved: April, 2016)**
38. Zablith F, Antoniou G, d'Aquin M, Flouris G, Kondylakis H, Motta E et al (2015) Ontology evolution: a process-centric survey. Knowl Eng Rev 30(1):45–75
39. Zamborlini V, Guizzardi G (2010) On the representation of temporally changing information in OWL. In: Workshops proceedings of the 14th IEEE international enterprise distributed object computing conference (EDOCW 2010), Vitória, Brazil, 25–29 October, pp 283–292
40. Zekri A, Brahmia Z, Grandi F, Bouaziz R (2014) τOWL: a framework for managing temporal semantic web documents. In: Proceedings of the 8th international conference on advances in semantic processing (SEMAPRO 2014), Rome, Italy, 24–28 August, pp 33–41
41. Zekri A, Brahmia Z, Grandi F, Bouaziz R (2015a) τOWL: a framework for managing temporal semantic web documents supporting temporal schema versioning. Int J Adv Softw 8(1&2):85–102 **(IARIA)**

42. Zekri A, Brahmia Z, Grandi F, Bouaziz R (2015b) Temporal schema versioning in τOWL. In:Proceedings of the 2nd international conference on knowledge management, information and knowledge systems (KMIKS 2015), Hammamet, Tunisia, 16–18 April, pp 81–92
43. Zekri A, Brahmia Z, Grandi F, Bouaziz R (2015c) τOWL-Manager: a tool for managing temporal semantic web documents in the τOWL framework. In: Proceedings of the 9th international conference on advances in semantic processing (SEMAPRO 2015), Nice, France, 19–24 July, pp 56–64
44. Zekri A, Brahmia Z, Grandi F, Bouaziz R (2016) Temporal schema versioning in τOWL: a systematic approach for the management of time-varying knowledge. Accepted for publication on Journal of Decision Systems