

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Ontology Versioning Driven by Instance Evolution in the τ OWL Framework

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:

Brahmia Z., Grandi F., Zekri A., Bouaziz R. (2022). Ontology Versioning Driven by Instance Evolution in the τ OWL Framework. JOURNAL OF INFORMATION & KNOWLEDGE MANAGEMENT, 21(1), 1-46 [10.1142/S0219649222500022].

Availability:

This version is available at: <https://hdl.handle.net/11585/882266> since: 2024-01-23

Published:

DOI: <http://doi.org/10.1142/S0219649222500022>

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (<https://cris.unibo.it/>).
When citing, please refer to the published version.

(Article begins on next page)

VERSIONE REVISIONATA E ACCETTATA

DELL' ARTICOLO PUBBLICATO COME <https://doi.org/10.1142/S0219649222500022>

Ontology Versioning Driven by Instance Evolution in the τ OWL Framework

Zouhaier Brahmia^{1,*}, Fabio Grandi^{2,**}, Abir Zekri^{1,***}, and Rafik Bouaziz^{1,****}

¹ Multimedia, InforMation Systems, and Advanced Computing Laboratory, University of Sfax,
Road of the Aerodrome, Km 4.5, P.O.Box 1088, 3018 Sfax, Tunisia

² Department of Computer Science and Engineering, Alma Mater Studiorum – Università di Bologna,
Viale Risorgimento 2, I-40136 Bologna, Italy

* zouhaier.brahmia@fsegs.rnu.tn

** fabio.grandi@unibo.it

*** abir.zekr@gmail.com

**** rafik.bouaziz@usf.tn

Abstract. Like other components of Semantic Web-based applications, ontologies are evolving over time to reflect changes in the real world. Several of these applications require keeping a full-fledged history of ontology changes so that both ontology instance versions and their corresponding ontology schema versions are maintained. Updates to an ontology instance could be *non-conservative*, that is leading to a new ontology instance version no longer conforming to the current ontology schema version. If, for some reasons, a non-conservative update has to be executed, in spite of its consequence, it requires the production of a new ontology schema version to which the new ontology instance version is conformant so that the new ontology version produced by the update is globally consistent. In this paper, we first propose an approach that supports ontology schema changes which are triggered by non-conservative updates to ontology instances and, thus, gives rise to an ontology schema versioning driven by instance updates. Notice that in an engineering perspective, such an approach can be used as an incremental ontology construction method driven by the modification of instance data, whose exact structure may not be completely known at the initial design time. After that, we apply our proposal to the already established τ OWL (Temporal OWL 2) framework, which allows defining and evolving temporal OWL 2 ontologies in an environment that supports temporal versioning of both ontology instances and ontology schemas, by extending it to also support the management of non-conservative updates to ontology instance versions. Last, we show the feasibility of our approach by dealing with its implementation within a new release of the τ OWL-Manager tool.

Keywords: τ OWL, ontology schema, ontology instance, ontology instance update, ontology instance versioning, ontology instance version, ontology schema change, ontology schema versioning, ontology schema version

1. Introduction

1.1. Context of Work

The Semantic Web (Berners-Lee *et al.*, 2001; Antoniou *et al.*, 2012) can be defined as a common framework that allows knowledge and data to be shared and reused by different users and applications via their formalization and representation as ontologies (Gruber, 1995; Guarino, 1998). To this purpose, ontologies are composed of a terminological component (intensional knowledge), representing the data schema, and an assertional component (extensional knowledge) representing the data instances. Large scale Semantic Web applications require the management of ontologies with very large instance repositories (Heymans *et al.*, 2008; Kuiler, 2014; Konys, 2016), for which the adoption of database techniques to efficiently access data instances has been advocated (Seylan *et al.*, 2009; Al-Jadir *et al.*, 2010).

As πάντα ῥεῖ (“everything flows”, quoting Heraclitus), also Semantic Web ontologies cannot help but change over time and evolve (Heflin & Hendler, 2000a; Flahive *et al.*, 2015; Zablith *et al.*, 2015). Ontology evolution can be due to many reasons: changes in the domain (including enactment of new Laws) or in the user requirements, revision of the knowledge conceptualization (including error correction), or expansion of the domain representation (including ontology integration and merging). Several applications require keeping track of all the changes that have been applied to the underlying ontologies, in order to be able to recover past ontology versions (Klein *et al.*, 2002; Grandi, 2009; Im *et al.*, 2012; Taleb *et al.*, 2014), to track ontology changes over time (Noy *et al.*, 2004; Plessers *et al.*, 2007; Khattak *et al.*, 2013; Lambrix *et al.*, 2016), and to query temporal (Grandi, 2010; O'Connor & Das, 2011; Artale *et al.*, 2017) or multi-version (Liu *et al.*, 2014) ontologies. Such requirements can effectively be met with the adoption of the temporal ontology versioning technique, as proposed for instance in (Grandi, 2011) or implemented in the τ OWL framework (Zekri *et al.*, 2016; Zekri *et al.*, 2017), supporting the maintenance of a complete history of ontology instance versions with their corresponding ontology schema versions. Notice that in our work we define an ontology as a consistent set of two dependent components: an ontology schema and an ontology instance, such that the ontology instance must be always conformant to its ontology schema (i.e., the structure of the ontology instance is always equal to or compatible with the ontology schema).

In the state-of-the-art of (temporal) ontology schema evolution and versioning (Zablith *et al.*, 2015), changes to an ontology schema are explicitly made by an ontology administrator through a graphical interface or using a textual ontology schema change language (we assume the ontology administrator to be a special person, the only one allowed to create, modify, and remove ontologies). A new ontology schema version (different from the current one) can be

supplied as a whole by the ontology administrator or derived by incrementally applying schema change operations to the current ontology schema version. A sequence of ontology instance update operations may also be needed to adapt ontology instances to the new ontology schema (ontology schema change propagation), in order to ensure the consistency of the new ontology version, i.e., to guarantee that all ontology instances are syntactically/structurally conformant to the new ontology schema version. It is worth mentioning that the term “consistency” means, in this paper, syntactical or structural conformity between an ontology schema and its ontology instance(s); it is similar to the notion of validity of an XML instance document to an XML schema document, in the XML world.

On the other hand, when considering modifications of ontology instances, we can distinguish between updates that are *conservative* or *non-conservative* with respect to the ontology schema.

- Conservative updates add new ontology instances or modify existing ones such that the resulting ontology instances still conform to the current ontology schema.
- On the contrary, we call non-conservative updates the ones that would produce ontology instances non-conformant to the current ontology schema.

1.2. Problems

Non-conservative updates are threatening the ontology consistency (i.e., conformity of ontology instances w.r.t. their ontology schema) and, thus, cannot be thoroughly executed without being preceded by an ontology schema change. A commonly adopted approach for dealing with non-conservative updates consists of simply rejecting them (Heflin & Hendler, 2000a; Flahive *et al.*, 2015; Zablith *et al.*, 2015). To the best of our knowledge, only conservative updates are allowed and supported in existing ontology evolution/versioning management tools, like SHOE (Heflin & Hendler, 2000b), OntView (Klein & Fensel, 2001), PROMPTDiff (Noy & Musen, 2002), SemVersion (Völkel & Groza, 2006), CEX (Konev *et al.*, 2012), CHRONOS Ed (Preventis *et al.*, 2014), and τ OWL-Manager (Zekri *et al.*, 2016). Nevertheless, in many cases, non-conservative updates have to be executed in order to fulfill cogent application requirements, like (i) integrating new ontology instances into an ontology repository although they do not correspond to the current ontology schema (e.g., in a collaborative environment where there are multiple actors and, thus, multiple points of view, or when the ontology administrator wants to reuse some old ontology instance collection or extend the current ontology with new instances found in online Semantic Web repositories (Tzitzikas *et al.*, 2008; Merrill *et al.*, 2014)), (ii) providing more flexibility to ontology administrators by allowing them to perform non-conservative updates (e.g., owing to a decision coming from the managers or the decision makers in the enterprise), or (iii) taking into account some new knowledge which corrects or expands the current one (e.g., in

scientific, biological and medical applications). Therefore, if a non-conservative ontology instance update must be executed and at the state-of-the-art there are no available tools for helping him/her in such a task, the ontology administrator has to intervene in an ad hoc manner in order to carefully construct a new ontology schema version that can consistently accommodate the new ontology instance resulting from the update.

1.3. Objective and Contributions

Hence, the novel contribution of this work is that we propose that such a task be automated and, thus, we consider the execution of ontology schema changes driven by ontology instance evolution and propose practical solutions to deal with them that can be implemented in a setting where very large (or big) ontology instances (i.e., those with very large sizes) can be managed: when a new ontology instance created by an insertion or update does not fit into the current ontology schema (making the current ontology inconsistent), we want an implicit ontology schema change is triggered to automatically adapt the ontology schema to accept the new instance.

In particular, as a starting point for the management of non-conservative updates, we consider the τ OWL framework (Zekri *et al.*, 2016; Zekri *et al.*, 2017), which is an environment already supporting both temporal ontology instance versioning and temporal ontology schema versioning, in an integrated manner. Hence, in this paper, we propose an approach that extends τ OWL with implicit ontology schema versioning driven by the execution of non-conservative ontology instance updates. In other words, in our new approach, changes at instance level not only produce a new ontology instance version but may also lead to the automatic creation of a new ontology schema version. Moreover, from an engineering point of view, our approach can serve as an incremental ontology construction method driven by the ontology instances whose exact nature and structure may not be completely known at the initial design time.

1.4. Organization

The rest of the paper is organized as follows. Section 2 presents our approach for managing ontology schema changes that are generated by non-conservative ontology instance updates, in an environment that supports ontology versioning both at schema and instance levels. Section 3 applies such an approach to the τ OWL framework. Section 4 deals with the implementation of our proposal through the enhancement of the τ OWL-Manager tool, implementing the original τ OWL framework, with new modules that are necessary for supporting the new proposed functionalities. Section 5 discusses related work and clarifies the contributions of our approach and of its implementation with regard to the state of the art. Section 6 summarizes the paper and sketches directions for our future work.

2. Ontology Schema Changes Implicitly Triggered by Non-conservative Updates to Ontology Instances

In this section, we motivate and introduce our approach for automated support of ontology schema changes which are generated by non-conservative ontology instance updates.

In general, the main difference between the semantics of data in a database and in the Semantic Web is the fact that the former relies on a Closed World Assumption (CWA), whereas the latter is traditionally based on an Open World Assumption (OWA) (Patel-Schneider & Horrocks, 2007). Being the CWA purposely suitable to constraining and validating data, the OWA is considered more appropriate to describe knowledge in an extensible way, assuming data to be incomplete by default. In such a way, data intentionally underspecified can be easily reused and extended by others for their own ontology. Although the OWA is the “classical” viewpoint in the ontology world, the CWA has been adopted in several Semantic Web applications (Etzioni *et al.*, 1997; Heflin & Munoz-Avila, 2002) which require conformance of instances to a set of constraints: with the CWA, ontology definitions act like database schema specifications, which are rigid constraints on the values data may assume. The introduction of the CWA in the Semantic Web has been recently investigated also from a theoretical point of view, assuming the adoption of an instance repository obeying the CWA, called DBox in (Seylan *et al.*, 2009); also the coexistence of OWA and CWA for the same ontology has been studied in (Lutz *et al.*, 2012). The main motivation for this strategic move has to be sought in the convenience to resort to standard database technologies for efficiently querying very large instance repositories.

Notice that our approach is strictly limited to ontologies with only a DBox, which “basically” behave like a database and follow the CWA, for which we can talk about an “ontology schema”.

Besides, although basically sticking to the CWA, our proposed contribution is a way to add some kind of “openness” to the CWA approach by means of the implicit ontology versioning triggered by non-conservative updates. In practice, this happens in the same way that, in the database field, the adoption of a semistructured data model, like XML (Brahmia *et al.*, 2014a) or JSON (Brahmia *et al.*, 2017), strongly mitigates the inability to represent incomplete data of the relational model (where the only allowed incompleteness feature is represented by the controversial use of null values) by allowing the representation of flexible and extensible data structures. New ontology instances not conforming to the old ontology schema can be merged with the old ontology instances by creating a new ontology schema (from the old ontology schema), both compliant with the old and the new instances, by means of implicit schema changes automatically executed by the system (the ontology administrator must only approve their application) within the same transaction that adds the new instances. The use of the schema versioning technique (Brahmia *et al.*, 2018; Roddick, 2018) implies that the old

ontology schema version with the old ontology instance is also retained and can be retrieved on demand (e.g., to be still used by legacy applications or for audit purposes).

More precisely, and in a formal way, the implicit ontology schema changes are triggered by non-conservative updates to ontology instance documents, in the cases presented below. Notice that, for the description of these cases, we have used the following variables and functions:

- Variables:
 - C, C', Q, Q' : classes;
 - c, c' : an instance of the class C, C' , respectively;
 - q, q' : an instance of the class Q, Q' , respectively;
 - DP_i : data property i of a class;
 - S : current ontology schema;
 - ctr : a constraint specified in S .
- Functions:
 - $value(DP_i, c)$: it returns the value of “ DP_i ” in “ c ”;
 - $type(v)$: it returns the data type of the value “ v ”;
 - $violate(arg, ctr)$: it returns true, if the argument arg (a value “ v ”, or an instance “ r ” of an object property R) violates the constraint ctr , or false otherwise;
 - $involve(ctr, x)$: it returns true, if the definition of the constraint “ ctr ” takes into account the argument “ x ” (i.e., a data property of a class, or an object property) in an explicit manner, or false otherwise;
 - $ObjectProperty(C, Q)$: it returns the object property that links the two classes “ C ” and “ Q ”;
 - $mandatory(x, y)$: it returns true, if the argument “ x ” (i.e., a data property, or an object property) has been defined as mandatory in the argument “ y ” (i.e., a class, or an ontology schema, respectively), or false otherwise.

Case 1: a **rename** operation, which modifies the name of either the class of a class instance or the data property of a class instance, has been executed.

Case 2: an **insertion** operation that adds: (i) a new instance c of a class C that has not been already defined in the current ontology schema “ S ” (*Case 2.1*), (ii) a new instance c of an already existing class C , which violates some constraint in S , (*Case 2.2*), (iii) an instance r of an object property R not belonging to S (*Case 2.3*), or (iv) a new instance r of an already existing object property R , which violates some constraint in S (*Case 2.4*). These three sub-cases are detailed as follows:

Case 2.1: an *insertion* of a new instance c of a class C not already defined in the current ontology schema “S”

$$(c \in C \wedge C \notin S)$$

Case 2.2: an *insertion* of a new instance c of an already existing class C , which violates some constraint(s) in S

$$(c \in C \wedge C \in S) \wedge$$

$$[(\exists v = \text{value}(\text{DPi}, c) \wedge \text{DPi} \notin C) \vee$$

$$(\exists v = \text{value}(\text{DPi}, c) \wedge \text{DPi} \in C \wedge \text{type}(v) \neq \text{type}(\text{DPi})) \vee$$

$$(\exists v = \text{value}(\text{DPi}, c) \wedge \text{DPi} \in C \wedge \exists \text{ctr1} \in S \text{ such that } \text{violate}(v, \text{ctr1})) \vee$$

$$(\exists \text{ctr2} \in S \text{ such that } \text{violate}(c, \text{ctr2}))]$$

Case 2.3: an *insertion* of an instance $r = (c1, c2)$ of an object property R that does not belong to the current ontology schema “S”

$$(c1 \in C1 \wedge c2 \in C2) \wedge$$

$$(\nexists R \in S \text{ such that } R = \text{ObjectProperty}(C1, C2))$$

Case 2.4: an *insertion* of a new instance r of an already existing object property R , which violates some constraint in S

$$(c1 \in C1 \wedge C1 \in S \wedge c2 \in C2 \wedge C2 \in S) \wedge$$

$$(\exists R \in S \text{ such that } R = \text{ObjectProperty}(C1, C2) \wedge r \in R \wedge$$

$$\exists \text{ctr} \in S \text{ such that } \text{violate}(r, \text{ctr}))$$

Case 3: a **modification** operation, which replaces: (i) an instance c of a class C with a new instance c' of C (*Case 3.1*) or (ii) an instance r of an object property R with a new instance r' (*Case 3.2*), while violating some constraint already defined in the current ontology schema “S”. The two sub-cases are detailed in the following:

Case 3.1: a **modification** operation that replaces an instance c of a class C with a new instance c' of C , which violates some constraint in S

$$(C \in S \wedge \text{DPi} \in C \wedge \exists \text{newV} = \text{value}(\text{DPi}, c') \text{ such that}$$

$$[\text{type}(\text{newV}) \neq \text{type}(\text{DPi}) \vee$$

$$\exists \text{ctr} \in S \text{ such that } \text{involve}(\text{ctr}, \text{DPi}) \wedge \text{violate}(\text{newV}, \text{ctr})])$$

Case 3.2: a **modification** operation that replaces an instance r of an object property R with a new instance $r' = (c', q')$, which violates some constraint in S

$$(c' \in C' \wedge C' \notin S) \vee (q' \in Q' \wedge Q' \notin S) \vee$$

$$\begin{aligned}
& (c' \in C' \wedge C' \in S \wedge q' \in Q' \wedge Q' \in S) \wedge \\
& [(\nexists R \in S \text{ such that } R = \text{ObjectProperty}(C', Q') \wedge (R \in S)) \vee \\
& (\exists R \in S \text{ such that } R = \text{ObjectProperty}(C', Q') \wedge (R \in S) \wedge \\
& \exists \text{ctr} \in S \text{ such that } \text{involve}(\text{ctr}, R) \wedge \text{violate}(r', \text{ctr}))]
\end{aligned}$$

Case 4: a **deletion** operation, which removes: (i) the value v of a data property DPi (*Case 4.1*), (ii) an instance c of a class C (*Case 4.2*), or (iii) an instance r of an object property R (*Case 4.3*), while violating some constraint (explicitly) specified in the current ontology schema “S”. These three sub-cases are detailed below.

Case 4.1: a **deletion** operation that removes the value of a data property from an instance c of a class C , which violates some constraint in S

$$(c \in C \wedge C \in S \wedge \text{DPi} \in C \wedge \exists v = \text{value}(\text{DPi}, c) \text{ such that } \text{mandatory}(\text{DPi}, C))$$

Case 4.2: a **deletion** operation that removes an instance c of a class C , which violates some constraint in S

$$\begin{aligned}
& (C \in S \wedge \exists Q \in S \wedge \exists R \in S \text{ such that} \\
& R = \text{ObjectProperty}(Q, C) \wedge \text{mandatory}(R, S))
\end{aligned}$$

Case 4.3: a **deletion** operation that removes an instance r of an object property R , which violates some constraint in S

$$(R \in S \wedge \text{mandatory}(R, S))$$

To illustrate our approach, we provide in the following two examples.

Illustrative example no. 1: it deals with the addition of a new instance “c11” of a class “C1”, to the current ontology instance document version “D1_V1” that is conformant to the current ontology schema version “S1_V1”, while “c11” has a value “v” for a data property “DPx” that does not belong to “C1” in “S1_V1”. This scenario is similar to the above Case 2.2. To satisfy this requirement, an ontology schema change must be executed before performing the addition operation. Indeed, first “S1_V1” must be changed through an operation that adds a new data property “DPx” (whose data type is deduced from the type of the value “v”) to the class “C1”. Since we are in a multi-schema-version environment, changing “S1_V1” generates two components: “S1_V2”, the new/second ontology schema version, and “D1_V2”, the new/second ontology instance document version that is conformant to “S1_V2” and that is automatically created as a copy of “D1_V1”. After that, “c11” is stored in “D1_V2”, as it could be added to this latter without problems. At instance level, one of the two following situations will occur:

- If the new data property “DPx” has been declared, by the ontology administrator, as mandatory (in “C1”), the previous instances of “C1”, which have been initially inserted

under “S1_V1” and have “logically” migrated from “D1_V1” to “D1_V2” during the execution of the ontology schema change, are not conformant to “S1_V2” and therefore one of the two following situations will happen for each previous instance “prev_inst_C1” of “C1”, in order to preserve the consistency of the new ontology version:

- if the ontology administrator provides a value for “DPx”, this instance “prev_inst_C1” will be updated and kept in “D1_V2”;
- else, if the ontology administrator could not supply a value for “DPx”, this instance “prev_inst_C1” will be removed from “D1_V2”.
- Else, if “DPx” has been specified as optional, the previous instances of “C1” are conformant to both “S1_V1” and “S1_V2”.

Illustrative example no. 2: it deals with the addition of a new instance “c21” of a class “C2”, to the current ontology instance document version “D2_V1” which is conformant to the current ontology schema version “S2_V1”, while “c21” has not a value for a mandatory data property “DPy” that belongs to “C2” in “S2_V1”. This scenario is also similar to the above Case 2.2. To fulfill this requirement, an ontology schema change must be carried out before adding the new class instance. In fact, first “S2_V1” has to be changed via one of the following three operations:

- an operation that drops, from “C2”, the axiom that defines “DPy” as a mandatory data property;
- an operation that changes, in “C2”, the axiom that specifies “DPy” as a mandatory data property to an axiom that declares it as an optional one;
- an operation that drops, from “C2”, the data property “DPy” (and implicitly all axioms that are related to it).

This ontology schema change gives rise to two ontology components: “S2_V2”, the second ontology schema version, and “D2_V2”, the second ontology instance document version that is conforming to “S2_V2” and that is built as a copy of “D2_V1”. Thereafter, “c21” is added to “D2_V2”, since it could be stored in this latter without any problem. At instance level, one of the three following situations will occur:

- If the axiom that defines “DPy” as a mandatory data property, has been dropped from “C2” or if the axiom that specifies “DPy” as a mandatory data property of “C2”, has been changed to declare it as an optional one, then the previous instances of “C2”, introduced under “S2_V1” and migrated from “D2_V1” to “D2_V2” when executing the ontology schema change, are conformant also to “S2_V2” (by default, a data property is considered as optional).
- If the data property “DPy” has been dropped from “C2”, the previous instances of “C2”,

stored in “D2_V1” and copied in “D2_V2” during the accomplishment of the ontology schema change, are not conforming to “S2_V2” and, as a consequence, each previous instance “prev_inst_C2” of “C2” will be removed from “D2_V2”, in order to preserve the whole consistency of the new ontology version.

As for the ontology schema change operations that are implicitly triggered by the system to force the execution of the non-conservative ontology instance updates, they should be correctly generated, based on the details of the instance update operations and on the environment in which these updates are performed. Indeed, we propose two methods that allow a multi-version ontology system to generate the sequence of ontology schema change operations that is necessary for applying a sequence of non-conservative updates to an ontology instance document version while preserving the consistency of the new ontology version: the interactive and the non-interactive method. **(i) The interactive method** is based on the interaction of the ontology administrator with the system, while he/she is updating ontology instances through a suitable graphical user interface (GUI). The necessary ontology schema change operations are implicitly generated by a specific module, named the “Instance Update Interface Manager”. Such a module continuously collects all information supplied or chosen by the ontology administrator on the GUI, detects any possible ontology schema constraint violation, and automatically produces a valid sequence of ontology schema change operations whose execution (and the automatic propagation of their effects to the corresponding ontology instances) enables a consistent execution of the ontology instance update(s). **(ii) The non-interactive method** is based on the provision, by the ontology administrator, of a new entire ontology instance document version that has to be integrated into the current ontology version. The ontology schema change operations that are necessary to accommodate the new instance document version in the current version of the corresponding ontology, are implicitly triggered by a specific module, named the “Conformity Checker”. This module checks the conformity of the ontology instances, stored in the provided ontology instance document version, with respect to the current version of the ontology schema. If all instances are conforming to it, they are automatically integrated. In case there are some instances that are not conforming to the current ontology schema version, the “Conformity Checker” module generates a sequence of ontology schema change operations whose execution (and the propagation of their effects to the corresponding ontology instances) permit the addition of the new ontology instance document version to the current ontology schema version, in a consistent manner.

Notice that, in our present work, we consider only the first method (i.e., the interactive one), and apply it to the τ OWL framework and to its τ OWL-Manager tool, as shown in the next two sections, respectively. The second method will be studied in a future work.

3. Extension of the τ OWL Framework to Support Implicit Ontology Schema Changes

In this section, we first present the τ OWL framework (Zekri *et al.*, 2014; Zekri *et al.*, 2016). Then, we apply our approach to τ OWL by extending it to support non-conservative updates to ontology instances, which require the execution of implicit ontology schema changes.

3.1. The τ OWL Framework

In this subsection, first we provide the requirements that are satisfied through the building of τ OWL. Then, we present the architecture of this framework, in a detailed way.

3.1.1. Requirements

The τ OWL framework fulfills the following set of requirements:

- to make easy the management of time for ontology administrators and to allow him/her specifying and changing the temporal format of any OWL 2 ontology component;
- to support valid time and transaction time, for the management of the evolution over time of OWL 2 ontologies;
- to support both ontology instance versioning and ontology schema versioning, in an integrated manner;
- to keep compatibility with existing OWL 2 W3C recommendations, and editors, without any need to modify them;
- to support existing applications that exploit OWL 2 ontologies;
- to guarantee logical data independence (Burns *et al.*, 1986) for temporal OWL 2 ontologies by isolating changes to logical annotations from those to physical ones, and vice versa;
- to provide several physical representations for the same logical specification of a temporal OWL 2 ontology.

3.1.2. Architecture

τ OWL allows an ontology administrator to create a temporal OWL 2 (W3C, 2012a) schema and manage the corresponding temporal OWL 2 instances from a conventional OWL 2 schema, logical annotations, and physical annotations. In doing that, the τ OWL approach separates the ontology instances (assertional component) from the ontology schema (terminological component) by storing them as distinct OWL 2 documents; such a separation is usually considered as a good ontology design practice (Bergman, 2009). To this purpose, all the data instances belonging to an ontology are stored together in what we call an ontology instance document. The ontology instance document is assumed to be an OWL 2 file in RDF/XML format (W3C, 2004b), which is, according to the OWL 2 specification (W3C,

2012b), the only syntax that must mandatorily be supported by OWL 2 tools. However, for future releases of the τ OWL framework, it is also planned to support a different serialization format (e.g., JSON files or relational tables) in order to enable a more efficient processing of very large ontology instances.

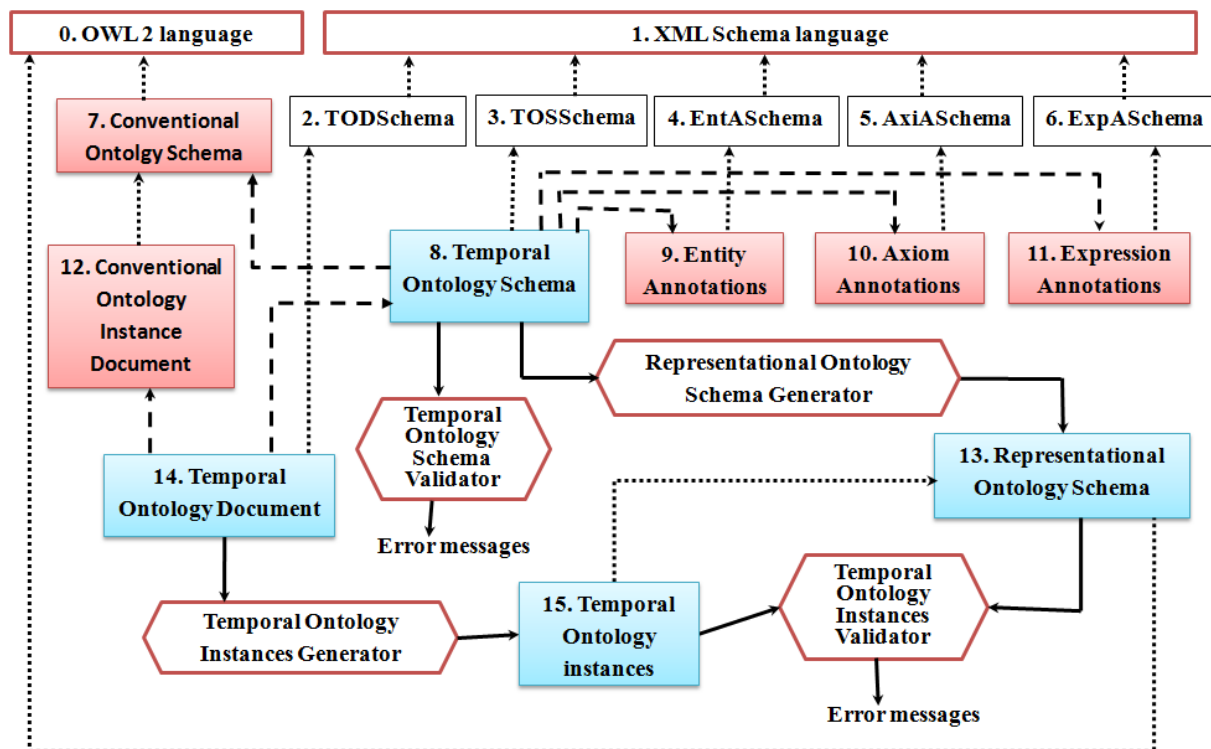


Figure 1. The overall τ OWL architecture.

After that, the ontology administrator augments the conventional ontology 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 such a schema, as described in the following.

- Logical annotations, which are inspired from those proposed in (Snodgrass *et al.*, 2008), allow the ontology administrator to specify (i) whether a conventional ontology schema component varies over valid time and/or transaction time, (ii) whether its lifetime is described as a continuous state or a single event, (iii) whether the component may appear at certain times (and not at others), and (iv) whether its content changes. If no logical annotations are provided, the default logical annotation is that anything can change. However, once the conventional ontology schema is annotated, components that are not described as time-varying are static and, thus, they must have the same value across every conventional ontology instance document (box 12).
- Physical annotations, which are inspired from those introduced in (Snodgrass *et al.*, 2008), allow the ontology administrator to specify the chosen timestamp representation options, such as where the timestamps are located, their kind (i.e., valid time or transaction time), and their representation. The location of timestamps is largely independent of which ontology components vary over time. Timestamps can be located either on time-varying components (as specified by the logical annotations) or somewhere above such components (i.e., on the components (grand) parents of these components). Two temporal OWL 2 documents with the same logical information will look very different if the ontology administrator changes the location of their physical timestamps. Changing an aspect of even one timestamp can make a big difference in the representation. τ OWL supplies a default set of physical annotations, which is to timestamp the root element with valid and transaction time. However, explicitly defining them can lead to more compact representations (Snodgrass *et al.*, 2008).

In order to improve conceptual clarity and also to enable a more efficient implementation, a “separation of concerns” principle is adopted in τ OWL: since the entities, the axioms and the expressions of an OWL 2 ontology evolve over time independently, the authors of τ OWL distinguish between three separate types of annotations to be defined and to be associated to a conventional ontology 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).

Notice that EntASchema, AxiASchema, and ExpASchema, which all contain both logical and physical annotations, are XML Schemas (W3C, 2004a). 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 ontology administrator finishes annotating the conventional ontology schema and asks the system to commit his/her work, the system creates the temporal ontology schema (box 8) in order to provide the linking information between the conventional ontology schema and its corresponding logical and physical annotations. The temporal ontology schema is a standard XML document, which ties the conventional ontology schema, the entity annotations, the axiom annotations, and the expression annotations together. In the τ OWL framework, the temporal ontology 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 ontology schema definitions with entity annotations, axiom annotations, and expression annotations, along with the time span during which the association was (or is) in effect. The schema for the temporal ontology schema document is the XML Schema Definition document TOSSchema (box 3).

To complete the picture in the considered temporal context, after creating the temporal ontology schema, the system creates a temporal ontology document (box 14) in order to link each conventional ontology instance document (box 12), which is conformant 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 ontology schema). A temporal ontology 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 ontology 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 (or is) in effect. Thus, the temporal ontology 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 ontology document is the XML Schema Definition document TODSchema (box 2).

Notice that, whereas TODSchema (box 2), TOSSchema (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 way similar to what happens in the τ XSchema framework (Snodgrass *et al.*, 2008) from which τ OWL is inspired, the temporal ontology schema document (box 8) is processed by the temporal ontology 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 ontology schema. The temporal ontology schema validator tool reports whether the temporal ontology schema document is valid or invalid.

Once all the annotations are found to be consistent, the representational ontology schema generator tool generates the representational ontology schema (box 13) from the temporal ontology schema (i.e., from the conventional ontology schema and the logical and physical annotations); it is the result of transforming the conventional ontology schema according to the requirements expressed through the different annotations. The representational ontology schema becomes the schema for temporal ontology instances (box 15). These latter are stored in a document, called the “squashed ontology document”; they are created automatically from the temporal ontology document (box 14), i.e., from the conventional ontology instances (box 12) and the temporal ontology schema (box 8), using the temporal ontology instances generator tool (such an operation is called “squash” in the original τ XSchema approach). Moreover, the temporal ontology instances are validated against the representational ontology schema through the temporal ontology instances validator tool, which reports whether the temporal ontology instances document or the squashed ontology document (box 15) is valid or invalid.

Notice that the τ OWL framework has been implemented within a tool, named τ OWL-Manager (Zekri *et al.*, 2015b; Zekri *et al.*, 2016). It is programmed in Java (JDK 1.7) within the IDE Eclipse Mars, using the OWL API (Horridge & Bechhofer, 2011) for creating and manipulating OWL 2 ontology files, and the JDOM API for creating and manipulating XML files. The first version of this tool, presented in (Zekri *et al.*, 2015b), (i) allows constructing a temporal ontology schema, by specifying some logical and physical annotations on an existing valid conventional ontology schema, and (ii) supports only temporal versioning of temporal ontology instances. The second version of this tool, presented in the subsection 4.3 of (Zekri *et al.*, 2016), supports temporal versioning of the conventional ontology schema.

3.2. Extension of τ OWL to Support Implicit Ontology Versioning

As ontology instances evolve over time to reflect the dynamics of the modelled reality, corresponding ontology instance documents are also being updated to put into effect such a dynamics. In the τ OWL environment, instances updates are applied only on the current

version of the corresponding ontology instance document, since both instance updates and schema changes are managed along transaction time, which means that only the current instance document version can be changed by adding new instances, or by modifying or deleting already existing instances.

The τ OWL-Manager tool provides a GUI that allows the ontology administrator to interactively update ontology instances. By receiving a command from the ontology administrator involving the commit of the performed update operation(s), τ OWL-Manager launches the execution of the “Ontology Instance Document Update Processor” module whose pseudo-code algorithm is given in Figure 2.

```

Algorithm Ontology_Instance_Document_Update_Processor

Inputs: ConvOntDoc_CV, ConvOntSch_CV, OIU_Ops, OSC_Ops, TempOntDoc,
          TempOntSch

Outputs: ConvOntDoc_NV, ConvOntSch_NV, SquashOntDoc_NV

Begin
  1. CopyOntInstDoc (ConvOntDoc_CV, ConvOntDoc_NV);
  2. ExecuteOntInstanceUpdates (ConvOntDoc_NV, OIU_ops);
  3. resultComp := CompareOntInstDoc (ConvOntDoc_NV, ConvOntDoc_CV);
  4. If (resultComp) Then
  5.   RemoveOntInstDoc (ConvOntDoc_NV);
  6.   Display("In reality, no instance updates have been executed on
              the current version of the conventional ontology instance
              document");
  7. Else
  8.   UpdateTemporalOntologyDoc (TempOntDoc, ConvOntDoc_NV);
  9.   If (!Empty(OSC_Ops)) Then /* ConvOntDoc_NV is not
                                conformant to ConvOntSch_CV */
  10.    CopyOntSchema (ConvOntSch_CV, ConvOntSch_NV);
  11.    ExecuteOntSchemaChanges (ConvOntSch_NV, OSC_Ops);
  12.    PropagateOntSchemaChangesTo (ConvOntDoc_NV);
  13.    UpdateTemporalOntologySchema (TempOntSch, ConvOntSch_NV);
  14.  End If
  15.  GenerateSquashedOntologyDoc (SquashOntDoc_NV, TempOntSch,
                                ConvOntDoc_NV);
  16. End If

End

```

Figure 2. The algorithm executed by the “Ontology Instance Document Update Processor”.

This pseudo-code algorithm uses the following variables, functions, and procedures.

- Variables:
 - ConvOntDoc_CV: the current version of the conventional ontology instance document;
 - ConvOntDoc_NV: the new version of the conventional ontology instance document;
 - ConvOntSch_CV: the current version of the conventional ontology schema;
 - ConvOntSch_NV: the new version of the conventional ontology schema;
 - OIU_Ops: a valid sequence of update operations to conventional ontology instances, which is generated by the “Instance Update Interface Manager”;
 - OSC_Ops: a valid sequence of change operations to conventional ontology schema, which is generated by the “Instance Update Interface Manager”;
 - SquashOntDoc_NV: the new squashed ontology document, associated to the updated temporal ontology document TempOntDoc;
 - TempOntDoc: the temporal ontology document that links together the conventional ontology instance document versions and the temporal ontology schema;
 - TempOntSch: the temporal ontology schema that links together the conventional ontology schema versions and the temporal ontology annotation document versions.
- Functions:
 - CompareOntInstDoc(oid1, oid2): it compares two ontology instance documents (oid1 and oid2) and returns true if they have the same contents, or false otherwise;
 - Empty(osc_ops): it returns true if the sequence of ontology schema change operations (osc_ops) passed as argument is empty, or false otherwise.
- Procedures:
 - CopyOntInstDoc(oid_cv, oid_nv): it creates a new conventional ontology instance document (oid_nv) as a copy of the one passed as argument (oid_cv);
 - ExecuteInstanceUpdates(conv_oid, oi_u_ops): it executes the sequence of ontology instance update operations (oi_u_ops) on the conventional ontology instance document passed as argument (conv_oid);
 - RemoveOntInstDoc(conv_oid): it removes, from the disc, the conventional ontology instance document passed as argument (conv_oid);
 - Display(msg): it displays the message passed as argument (msg);
 - CopyOntSchema(os_cv, os_nv): it creates a new conventional ontology schema (os_nv) as a copy of the one passed as argument (os_cv);
 - ExecuteOntSchemaChanges(conv_os, osc_ops): it executes the sequence of ontology schema change operations (osc_ops) on the conventional ontology schema passed as

argument (conv_os);

- UpdateTemporalOntologyDoc(tod, conv_oid): it adds, to the temporal ontology document passed as argument (tod), a new slice associated to a new conventional ontology instance document version (conv_oid);
- PropagateOntSchemaChangesTo(conv_oid): it propagates the conventional ontology schema change operations executed so far to the conventional ontology instance document passed as argument;
- UpdateTemporalOntologySchema(tos, conv_os): it adds, to the temporal ontology schema passed as argument (tos), a new slice associated to a new conventional ontology schema version (conv_os);
- GenerateSquashedOntologyDoc(sod, tos, conv_oid): it creates a new squashed ontology document (sod) based on a new conventional ontology instance document version (conv_oid) and its temporal ontology schema (tos).

In general, our approach simplifies the checking of ontology instance conformity and the generation of implicit ontology schema change operations, which are then performed in an efficient way, since the system relies on both the current ontology schema version and the full sequence of instance update operations (which have been executed on the current ontology instance document version), to detect the ontology administrator's operations (i) which move inside this ontology schema version (i.e., conservative updates to ontology instances) or (ii) which cross its borders (i.e., non-conservative updates to ontology instances) and consequently require implicit changes to be done on the current ontology schema. More precisely, we describe below how ontology instances are updated through the GUI of the τ OWL-Manager tool. To this purpose, τ OWL-Manager is based on two main components: the "Instance Update Interface Manager" and the "Ontology Instance Document Update Processor".

To add a new class instance, the ontology administrator starts by selecting the target class from the class hierarchy to the left of the GUI (as shown in Figure 3) and clicking on the contextual menu "Create instances" or by writing the class name in a suitable text field. If the entered name actually does not correspond to an existing class, the "Instance Update Interface Manager" assumes the ontology administrator wants to create a new class (i.e., it detects a non-conservative ontology instance update) and, after asking for confirmation, asks the ontology administrator where the new class should be put in the hierarchy (i.e., to indicate its superclass); once the superclass of the new class is provided, the "Instance Update Interface Manager" generates the corresponding ontology schema changes:

```
AddClass(newClassName);  
AddSubClass(newClassName, superClassName);
```

After that, the "Instance Update Interface Manager" shows the Data Property Assertions

displaying the data properties of the selected class, or the data properties that are inherited from the superclass in case of a new subclass, that the ontology administrator can use to fill in the slots in order to specify values for the existing data properties.

Notice here that when adding a new instance of an existing class, the ontology administrator could also perform the following tasks:

- to add new data properties, by clicking on the link “Add Data Property Instance(s)” and specifying a data property name and possibly a set of values for the new data property; thus, a non-conservative ontology instance update is detected and the required ontology schema changes is generated:

```
AddDataProperty(className, newDataPropertyName);
```

Confirmation is then asked to the ontology administrator whether the new data property has to be defined as optional and/or the number of inserted values has to be taken as a cardinality constraint; in the former case the following ontology schema change has to be added

```
SetCardinality(className, newDataPropertyName, minCard, 0);
```

whereas in the latter case the following ontology schema change has to be added

```
SetCardinality(className, newDataPropertyName, maxCard,  
                numberInsertedValues);
```

- to change the data type of existing data properties (e.g., from `xsd:integer` to `xsd:string`), since the inserted data property values may be non compatible with the type currently defined for that data property; hence a new data type is derived from the inserted values and, after asking confirmation to the ontology administrator, the following ontology schema change is generated:

```
ChangeDataPropertyType(className, dataPropertyName, newDataType);
```

- to rename existing data properties, since data properties names, which come either from the selected class or from the specified superclass, are editable; hence, the “Instance Update Interface Manager” detects a non-conservative ontology instance update and generates the corresponding ontology schema change:

```
RenameDataProperty(className, dataPropertyName, newName);
```

- to change the minimum cardinality of existing data properties (e.g., to change the data property from mandatory to optional), by deleting one or more values from those currently defined for the data property such that the remaining values are under the current minimum cardinality constraint (including deleting the single value of an existing mandatory data property); hence, the “Instance Update Interface Manager” detects a non-conservative ontology instance update and builds the corresponding ontology schema change:

```
SetCardinality(className, dataPropertyName, minCard,
               newNumberOfValues);
```

- to change the maximum cardinality of existing data properties (e.g., to change the data property from functional to generic), by adding one or more values to those currently defined for the data property such that the resulting values are over the current maximum cardinality constraint (including adding a second value to an existing functional data property); hence, the “Instance Update Interface Manager” detects a non-conservative ontology instance update and builds the corresponding ontology schema change:

```
SetCardinality(className, dataPropertyName, maxCard,
               newNumberOfValues);
```

- to remove some existing data properties, by clicking on the link “Remove Data Property Instance(s)”; consequently, in case the data property was currently defined as mandatory, a non-conservative ontology instance update is detected and the corresponding ontology schema changes are generated but confirmation is asked to the ontology administrator whether the data property has to be removed from the class definition or it has simply to be made optional: in the former case the generated ontology schema change is

```
RemoveDataProperty(className, dataPropertyName);
```

whereas in the latter case the generated ontology schema change is

```
SetCardinality(className, dataPropertyName, minCard, 0);
```

- to specify some instances of the object properties (i.e., relationships) of this class, by clicking on the button having the symbol “+” as a name and located to the right on top of the table reserved to Object Property Assertions, while some range values correspond to classes that do not exist in the current conventional ontology schema version (using the graphical editor, the domain of an object property instance cannot be changed since it corresponds to the chosen class on which the ontology administrator is working). In such a case, the “Instance Update Interface Manager” infers that the ontology administrator wants to change the range of some object property and the ontology administrator is interactively asked where to put the new classes in the class hierarchy; according to the ontology administrator’s answers, the “Instance Update Interface Manager” generates, for each changed range, the following required ontology schema changes:

```
AddClass(newClassName);
AddSubClass(newClassName, superClassName);
ChangeRange(objectPropertyName, newClassName);
```

Automatic change of cardinality constraints of object properties, triggered by non-

conservative insertions, deletions or modifications of object property instances, is managed similarly to changes of cardinality constraints of data properties, explained above.

In order to update existing class instances (with their data property instances and object property instances), the ontology administrator has to go to the menu “Ontology Instance Document” and to click on the submenu “Evolve Ontology Instances”. Hence, the “Instance Update Interface Manager” will let him/her choosing a conventional ontology schema (e.g., PersonFOAF in our example), or more precisely and implicitly the current version of this ontology schema, and according to the choice of the ontology administrator, the current version of the conventional ontology instance document, associated to the chosen conventional ontology schema, will be displayed. The ontology administrator could then update (in a broader sense) data property instances and/or object property instances of each class instance, either these updates are conservative or not. Everything the ontology administrator may choose, modify, delete or insert, that violates a constraint specified in the current version of the involved conventional ontology schema, the “Instance Update Interface Manager” assumes that he/she wants to perform a non-conservative ontology instance update that consequently requires an ontology schema change. Hence, the “Instance Update Interface Manager” uses the specifications and values provided by the ontology administrator to compile the required sequence of all necessary implicit ontology schema changes. Notice that the ontology schema changes generated during the interaction are similar to those presented above for the insertions of class instances.

When the ontology administrator finishes updating conventional ontology instance(s) and asks τ OWL-Manager to commit his/her work, the “Instance Update Interface Manager” receives his/her order and calls the module “Ontology Instance Document Update Processor” (whose pseudo-algorithm is listed in Figure 2), while passing to it four arguments: the current conventional ontology schema version (ConvOntSch_CV), the current conventional ontology instance document version (ConvOntDoc_CV) which is being “updated”, the final sequence of ontology instance update operations that have been made by the ontology administrator (OIU_Ops), and the complete sequence of generated ontology schema change operations (OSC_Ops); the former sequence constitute a log of the operations executed by the ontology administrator in the whole conventional ontology instance update session and the latter has been automatically generated during the interaction of the ontology administrator with τ OWL-Manager as detailed above. Being called, the module “Ontology Instance Document Update Processor” executes the actions presented in the pseudo-algorithm of Figure 2. Indeed, it generates a copy of ConvOntDoc_CV, executes the sequence of ontology instance update operations (OIU_Ops) on this copy, and compares the updated copy to ConvOntDoc_CV; if, there is no difference between them, it removes the updated copy and informs the ontology administrator that actually no changes have been made. However, if the updated copy ConvOntDoc_NV is different from ConvOntDoc_CV, it updates the temporal ontology

document corresponding to `ConvOntDoc_CV`, to take into consideration the new ontology instance document version `ConvOntDoc_NV`, and checks the conformity of `ConvOntDoc_NV` with respect to `ConvOntSch_CV`, by verifying if `OSC_Ops` is empty or not. In case `OSC_Ops` is not empty, then the module “Ontology Instance Document Update Processor” applies `OSC_Ops` on `ConvOntSch_CV` and obtains the new conventional ontology schema version `ConvOntSch_NV`. If necessary, it also propagates (under the interactive guidance of the ontology administrator) the conventional ontology schema changes (`OSC_Ops`) to the conventional ontology instances stored in `ConvOntDoc_CV`, which have not been affected by the ontology instance updates (`OIU_Ops`). After that, it updates the corresponding temporal ontology schema to include `ConvOntSch_NV`, and generates the new squashed ontology document version corresponding to `ConvOntSch_CV`.

It is worth mentioning that we have chosen to present the ontology schema change operations generated by the “Instance Update Interface Manager” (`OSC_Ops`) as high-level operations (Brahmia *et al.*, 2014b), since they correspond to frequent ontology schema evolution needs and allow to express intuitive ontology schema changes in a compact and user-friendly way. However, each one of these operations can easily be mapped onto a valid sequence of primitive operations for changing ontology schemas, which have been proposed in previous works (Zekri *et al.*, 2015a; Zekri *et al.*, 2016; Zekri *et al.*, 2017).

4. Implementation

Since the τ OWL-Manager prototype tool (Zekri *et al.*, 2015b; Zekri *et al.*, 2016) already supports the τ OWL approach, we have decided to show the feasibility of our proposal through its implementation as an extension of such a tool. This extension has consisted mainly in the two following tasks:

- We have revised the “Ontology Instance Document Change Manager” module so that it can also manage non-conservative ontology instance updates by allowing the creation of a new version of the conventional ontology instance document, which is not conformant to the current version of the conventional ontology schema; notice that such a conventional ontology instance document version was automatically rejected, in the previous releases of τ OWL-Manager.
- We have built a new module, called “Implicit Ontology Schema Change Manager”, which generates, in an automatic and transparent manner, a new version of the conventional ontology schema and evolves its temporal ontology schema, every time the “Ontology Instance Document Change Manager” module calls it.

In order to clarify the usage of the new release of the τ OWL-Manager tool, we provide below two examples that show how this tool manages non-conservative updates to ontology instance documents. We assume to work on an ontology `PersonFOAF`, whose schema is initially a

copy of the well-known FOAF definition¹.

In the first example (see Figures from 3 to 11), the non-conservative update consists in a new instance of the class “Person” with two new instances for two data properties (i.e., “address” and “phone” properties), which do not belong to the current PersonFOAF ontology schema version. Notice that in this version, the class “Person” has only three data properties (for the sake of simplicity) whose names are “name”, “surname”, and “country”, and which are all mandatory.

In the second example (see Figures from 12 to 16), the non-conservative update consists in adding a new instance of the class “Person” that does not include a value for a mandatory data property (i.e., “phone”).

In the instance update session corresponding to example 1, the creation of a new instance of the class "Person" is started in Figure 3, and the tool is asked to add new data property instance(s) in Figure 4. In Figure 5, two new data property instances (address, optional with the value “24, Avenue of the Revolution, 1000 Tunis”, and phone, mandatory with the value “1122334455”) are being prepared to be added to the new Person instance. Adding the two new data property instances (of address and phone) to the new Person instance is then shown: in Figure 6 the ontology administrator asks the tool to add these two new instances, and in Figure 7 the two new property instances are shown at the end of the list of property instances of the new Person instance. In Figure 8, the tool is asked to save the new Person instance and informs the ontology administrator that a new conventional ontology schema version will be added to the τ OWL repository and asks him/her to continue (i.e., saving the new Person instance while adding a new ontology schema version) or to cancel the work. There is also an option to view this new ontology schema version. Figures from 9 to 11 show the creation of the new conventional ontology instance document version (OD_PersonFOAF_V2.rdf) and the new conventional ontology schema version (PersonFOAFSchema_V2.owl), while updating the temporal ontology document and the temporal ontology schema. In particular, in Figure 9, the tool provides this information, in Figure 10 it informs the ontology administrator that he/she has defined the phone property as mandatory and asks him/her supplying values for this property in old instances if he/she wants that they also conform to the new ontology schema version (the ontology administrator has not supplied any value), and in Figure 11 the tool displays the code of the new ontology instance document version and the code of the new ontology schema version.

¹ <http://xmlns.com/foaf/spec/>

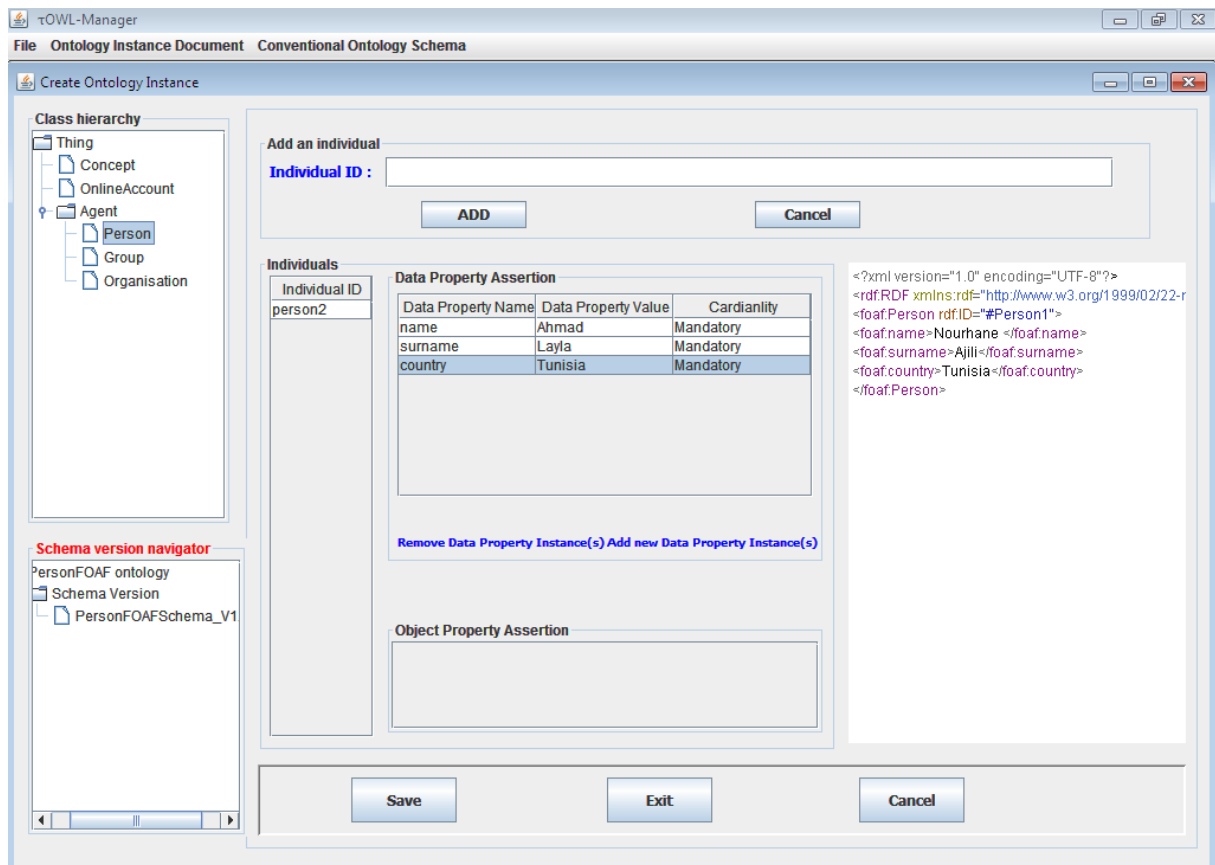


Figure 3. Step 1 of the ontology instance update process session of example 1. The ontology administrator starts the addition of a new instance of the class “Person” with the following data property values: "name"="Ahmad", "surname"="Layla", and "country"="Tunisia".

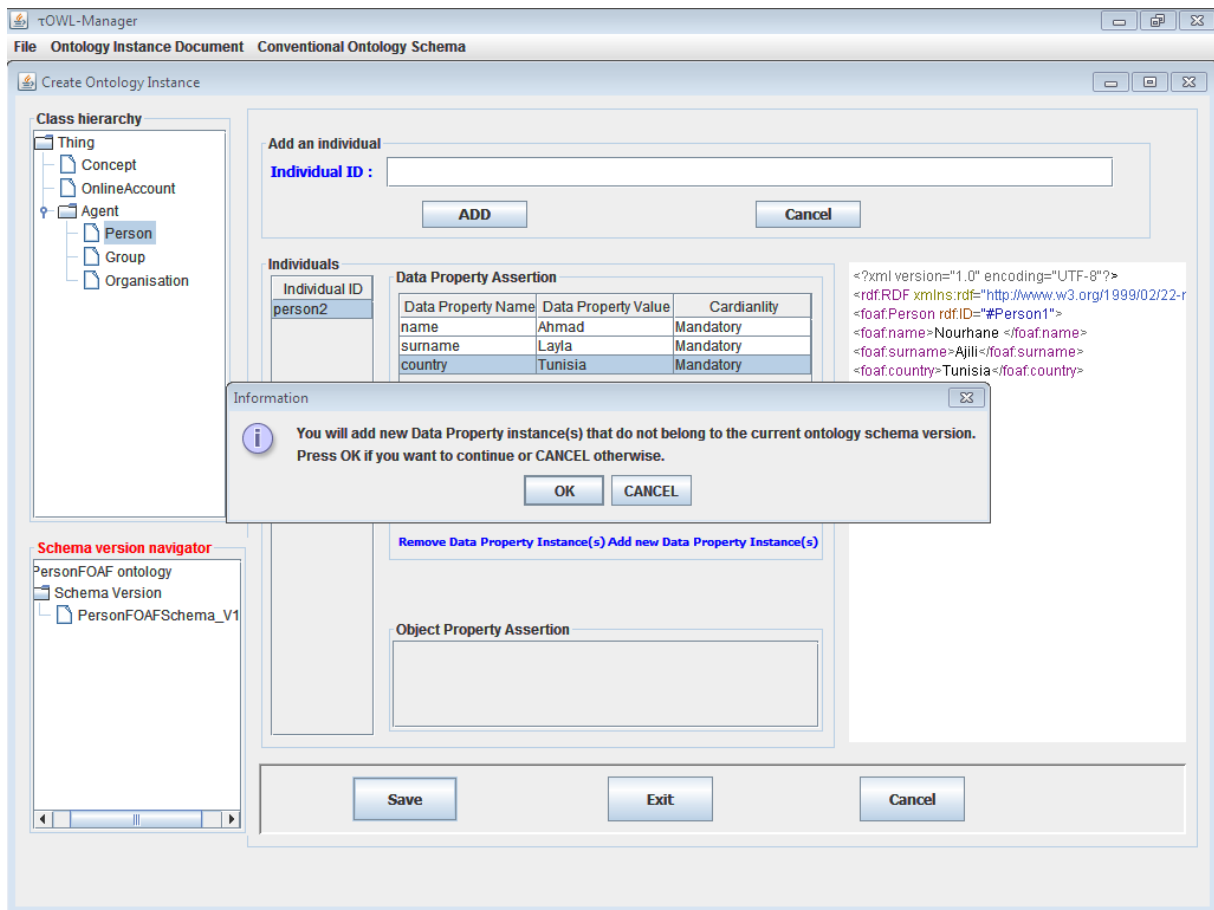


Figure 4. Step 2 of the ontology instance update process session of example 1. The ontology administrator requests the addition of new data property instance(s), for the new “Person” instance, and the tool informs him/her that he/she will add new data properties that do not belong to the definition of the class “Person”, in the current ontology schema version.

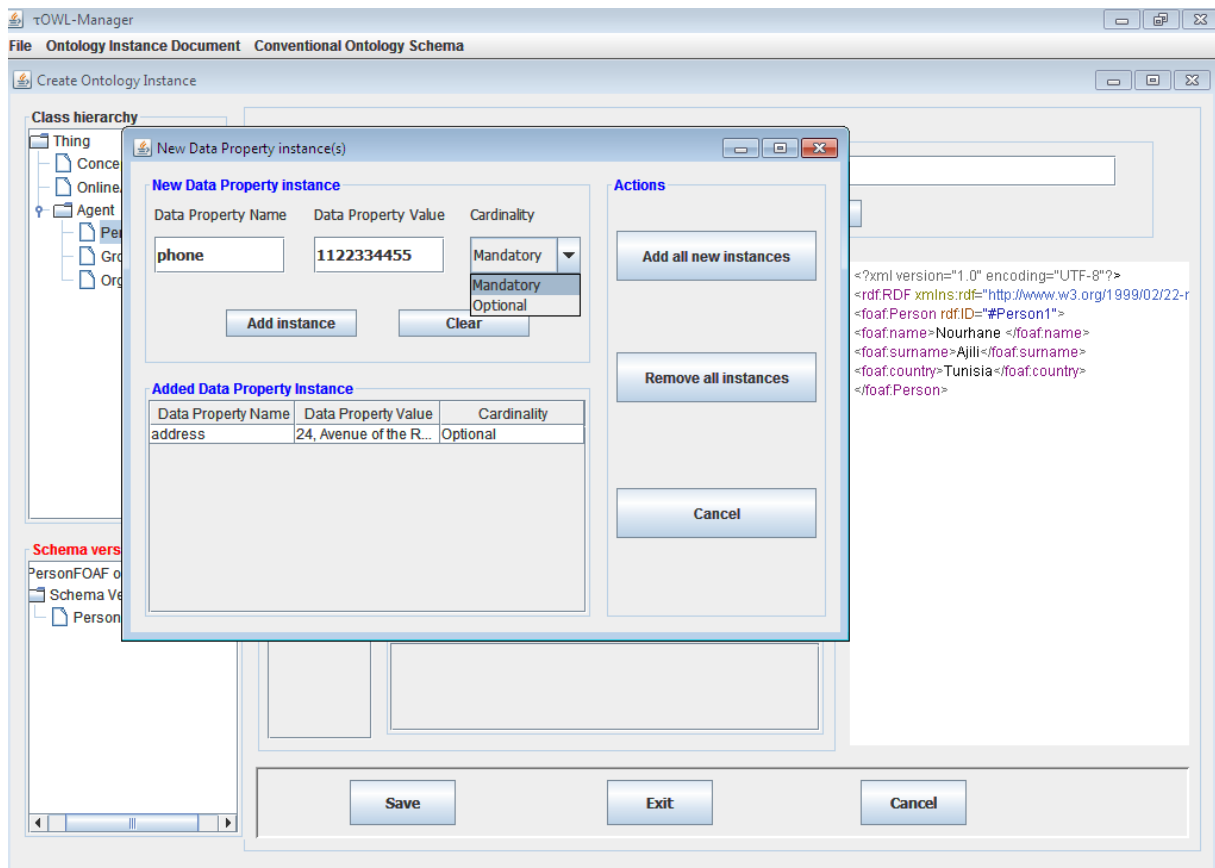


Figure 5. Step 3 of the ontology instance update process session of example 1. The ontology administrator specifies, for the same “Person” instance, the name, the value, and the cardinality constraint of each new data property (“address”, and “phone”), and asks the tool to save such a new instance.

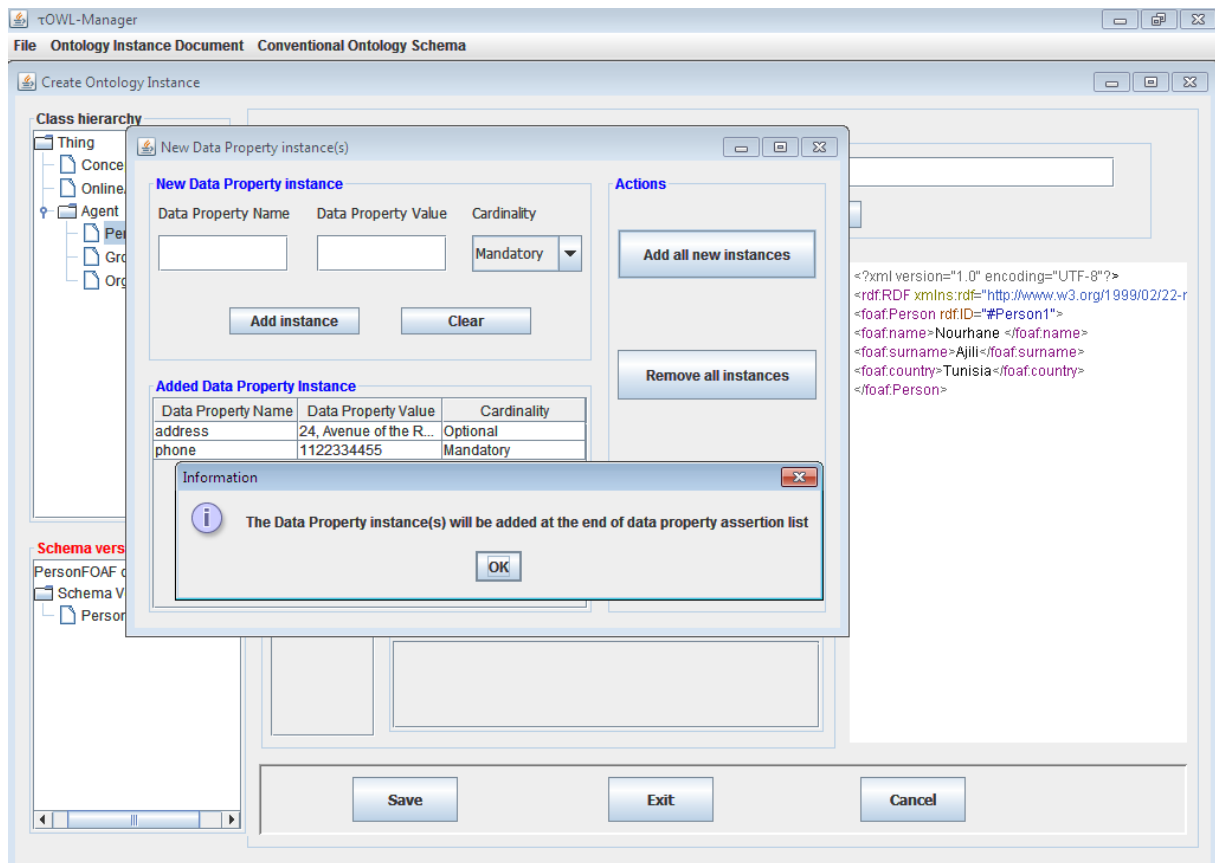


Figure 6. Step 4 of the ontology instance update process session of example 1. The tool informs the ontology administrator that the new data property instances will be added at the end of the data property assertion list associated to the new instance of the class “Person”.

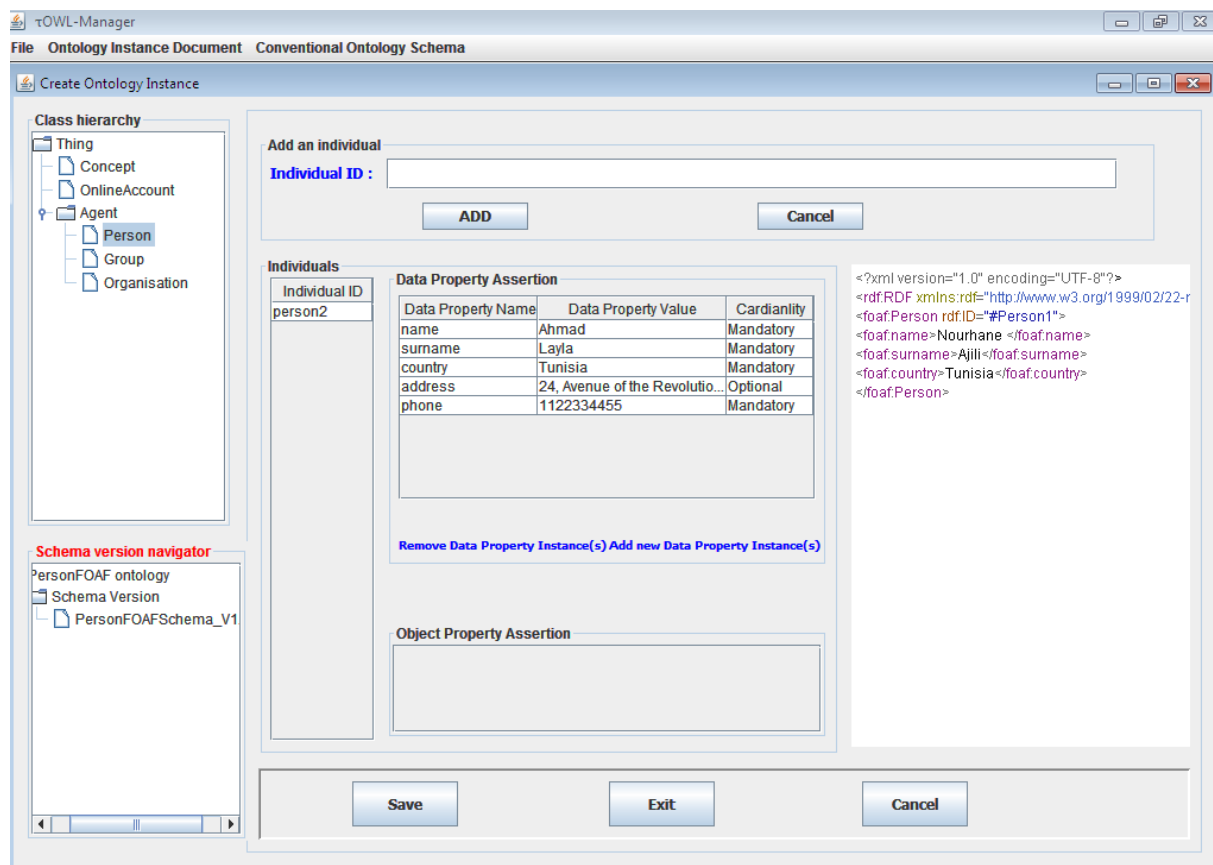


Figure 7. Step 5 of the ontology instance update process session of example 1. The tool displays all data property instances that have been previously specified by the ontology administrator for the new instance of the class “Person”.

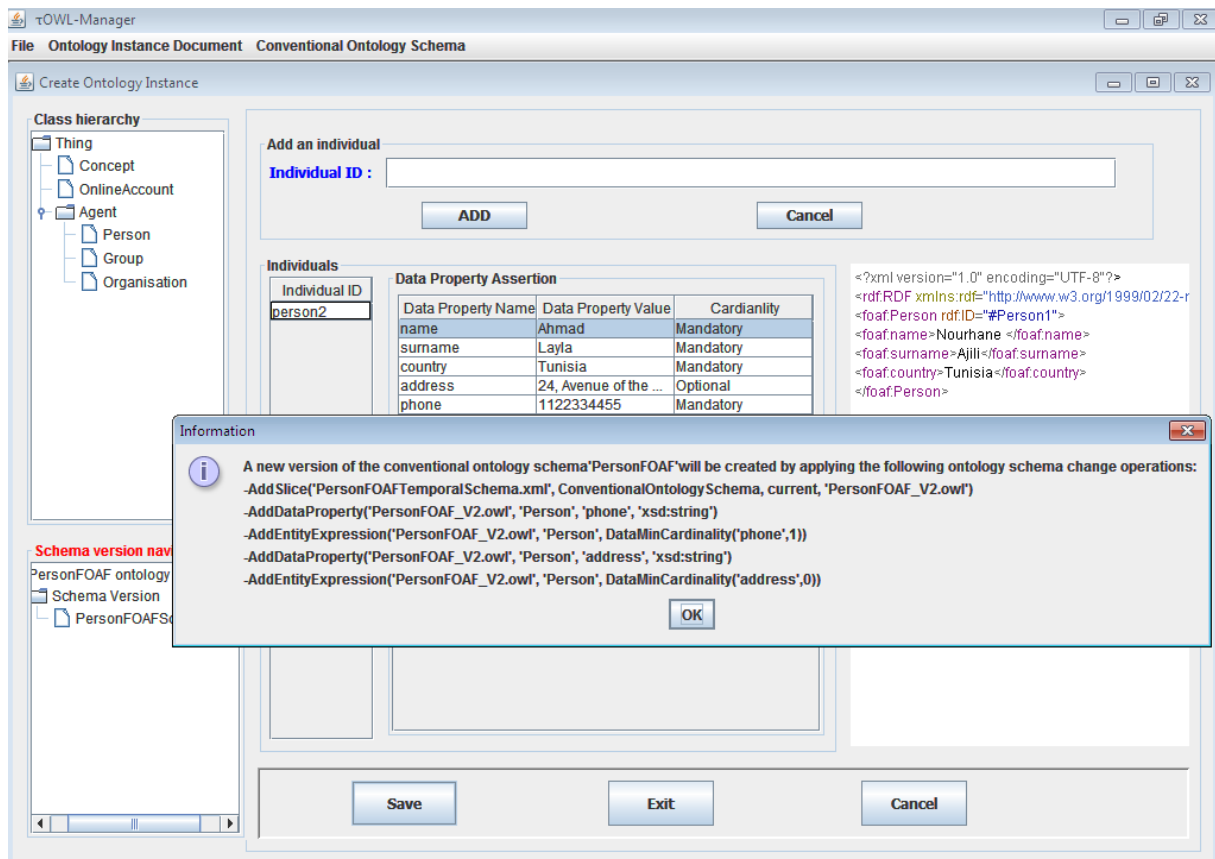


Figure 8. Step 6 of the ontology instance update process session of example 1. The tool informs the ontology administrator that a new version of the conventional ontology schema PersonFOAF will be created (as a result of the specification of a non-conservative insertion operation of a “Person” instance) and provides the full sequence of the ontology schema change operations that will be executed to produce this new schema version (by adding, to the class “Person”, the two new data properties “address” and “phone”).

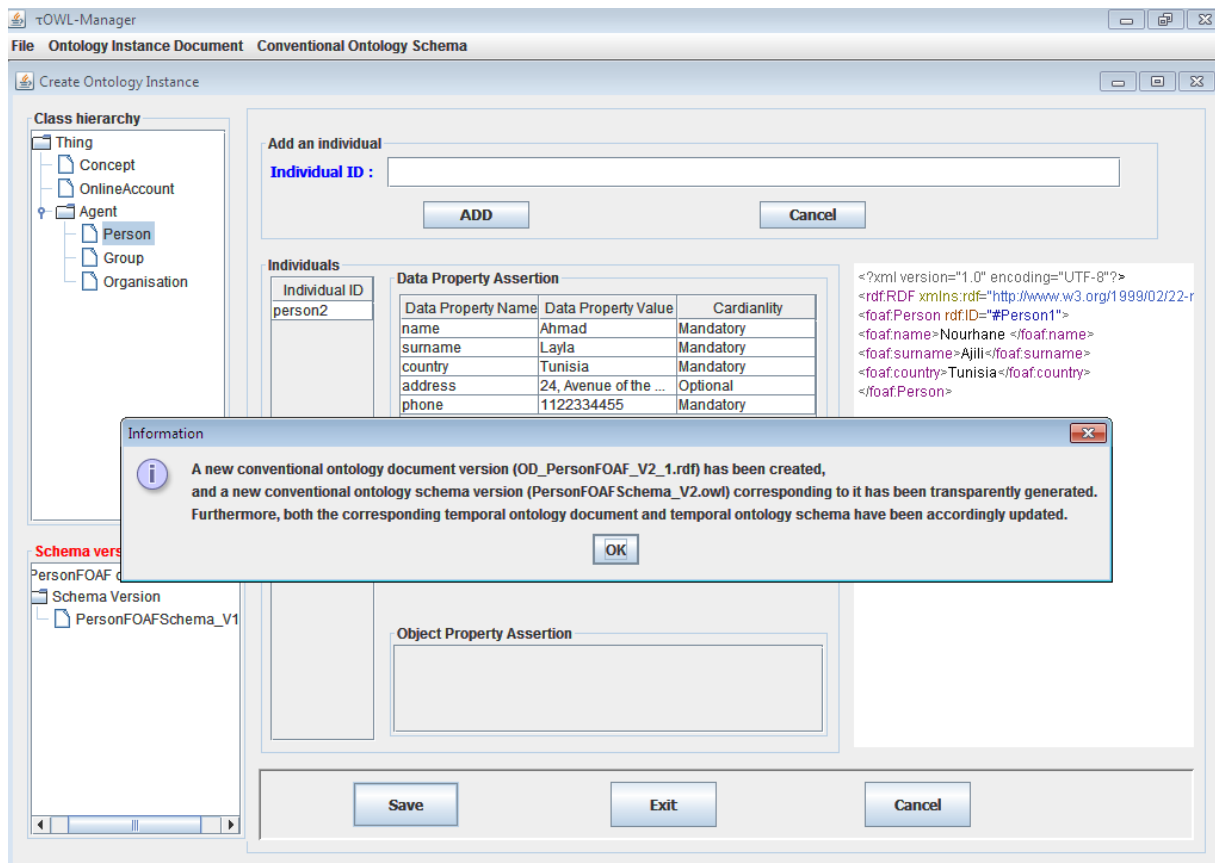


Figure 9. Step 7 of the ontology instance update process session of example 1. The tool informs the ontology administrator that (i) both a new conventional ontology instance document version (that stores the new “Person” instance) and a new conventional ontology schema version, corresponding to this instance document version, have been created, (ii) the temporal ontology document has been updated to include the new slice corresponding to the new conventional ontology instance document version, and (iii) the temporal ontology schema has been updated to take into account the new conventional ontology schema version.

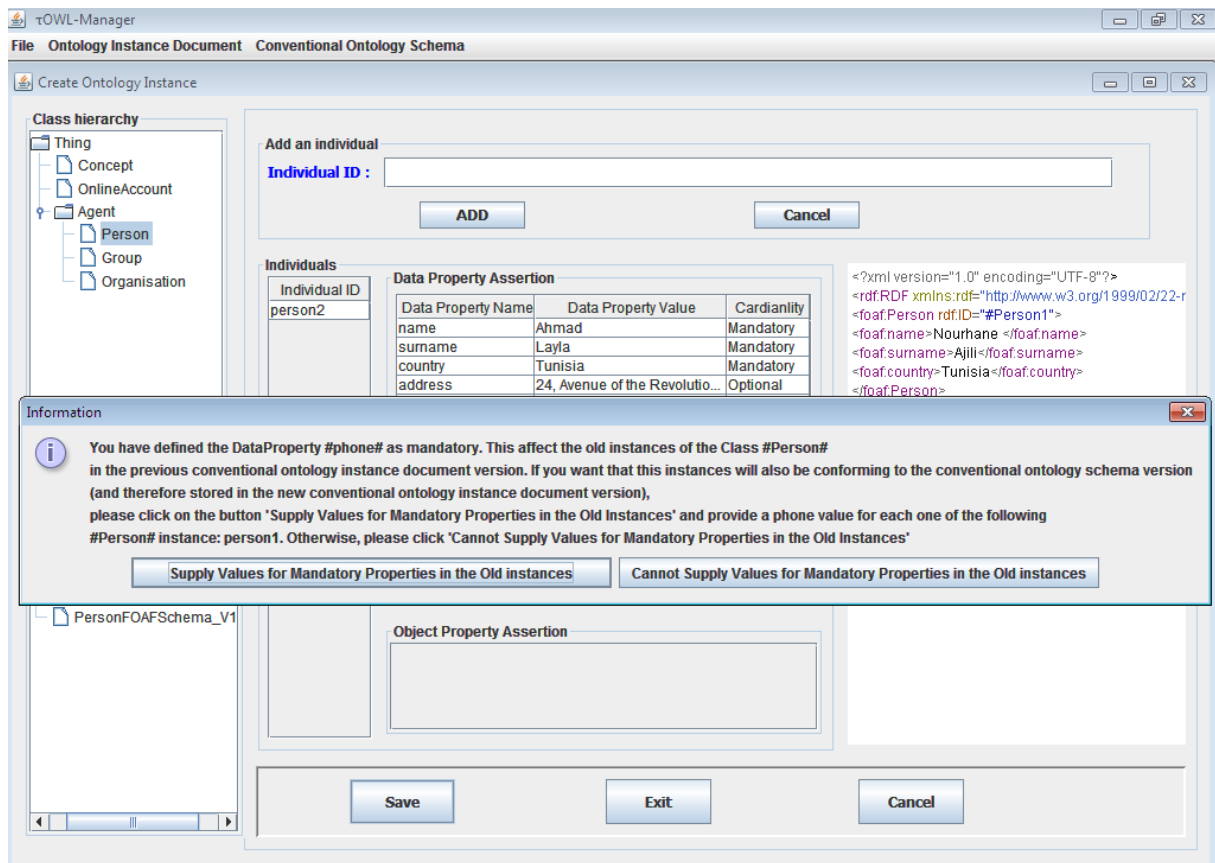


Figure 10. Step 8 of the ontology instance update process session of example 1. The tool informs the ontology administrator that the data property “phone” of the class “Person” is mandatory, and asks him/her to provide values for this data property in instances of the class “Person”, which have been defined according to the previous version of the conventional ontology schema (i.e., without the two data properties “address” and “phone”), if he/she would like that these old “Person” instances also conform to the new conventional ontology schema version. Notice that the ontology administrator has chosen not to supply values for these instances.

the temporal ontology document and the temporal ontology schema. In particular, in Figure 15 the tool provides this information and in Figure 16 the tool displays the code of the new ontology instance document version and the code of the new ontology schema version.

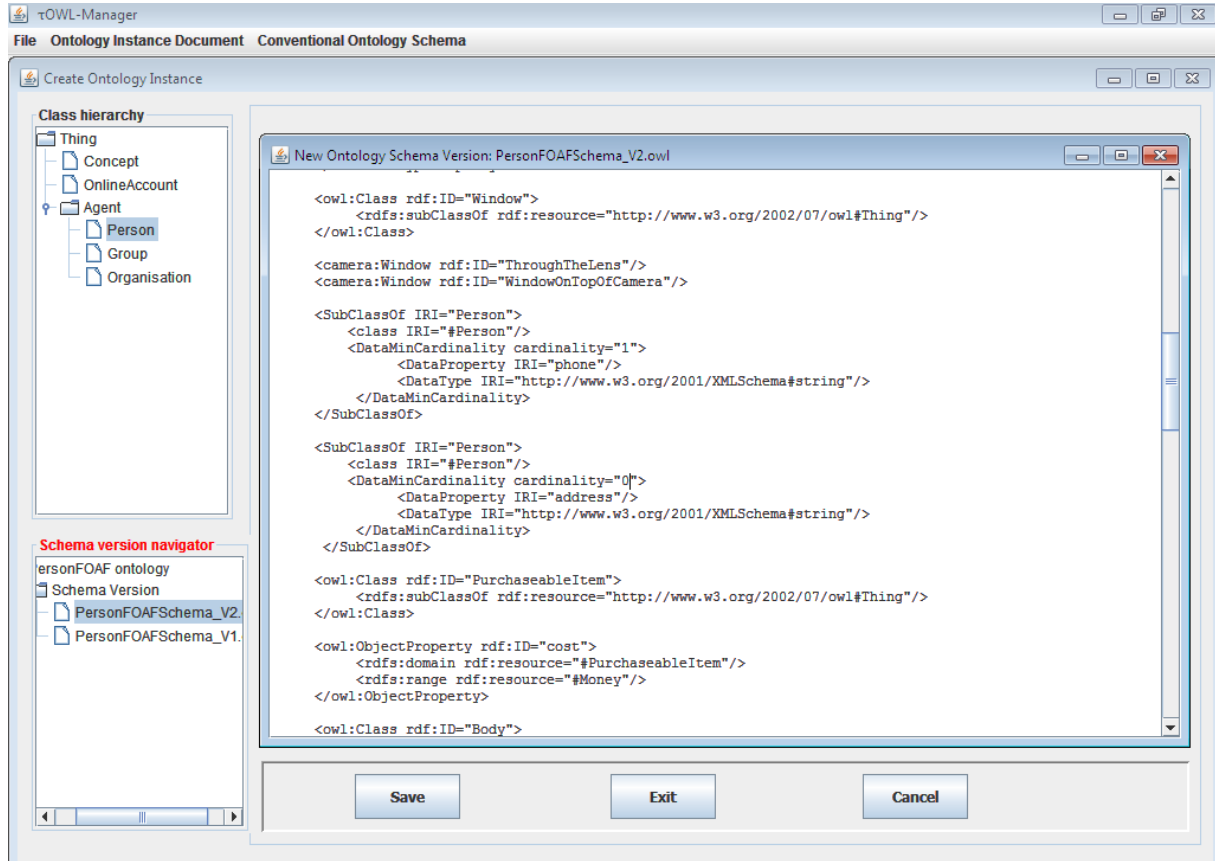


Figure 12. Step 1 of the ontology instance update process session of example 2. The tool shows that, in the code of new conventional ontology schema version, a class expression “DataMinCardinality”, with the value “1” assigned to the attribute “cardinality”, has been defined on the data property “phone” of the class “Person”, as a sub-class of this latter.

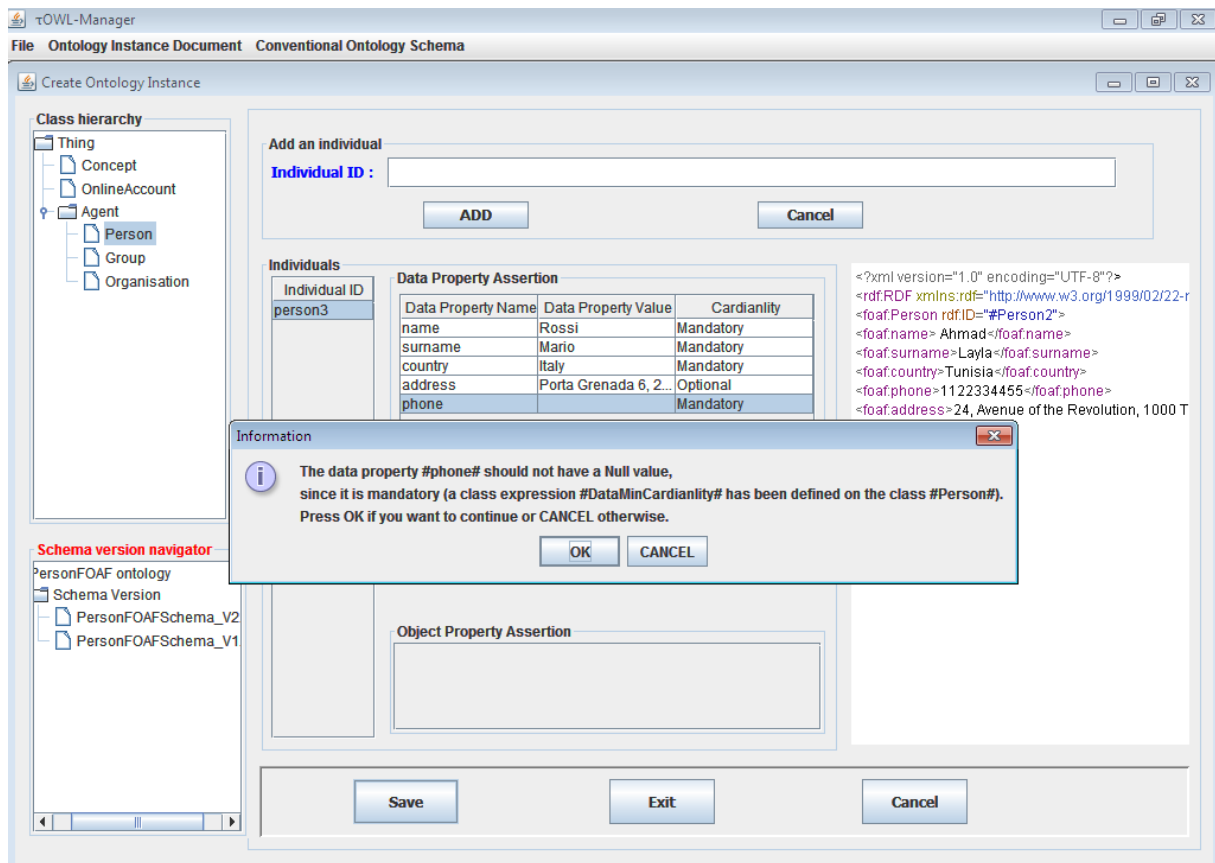


Figure 13. Step 2 of the ontology instance update process session of example 2. The tool (i) informs the ontology administrator that the data property “phone” of the class “Person” is mandatory, and therefore should not have a Null value in the new instance of the class “Person” (having the following data property values: "name"="Rossi", "surname"="Mario", "country"="Italy", and "address"="Porta Grenada 6, 20100 Milano"), and (ii) asks him/her to press either the button “OK”, if he/she wants to force the execution of this (non-conservative) insertion operation and to continue in this direction, or the button “CANCEL”, if he/she does not want to add this new instance.

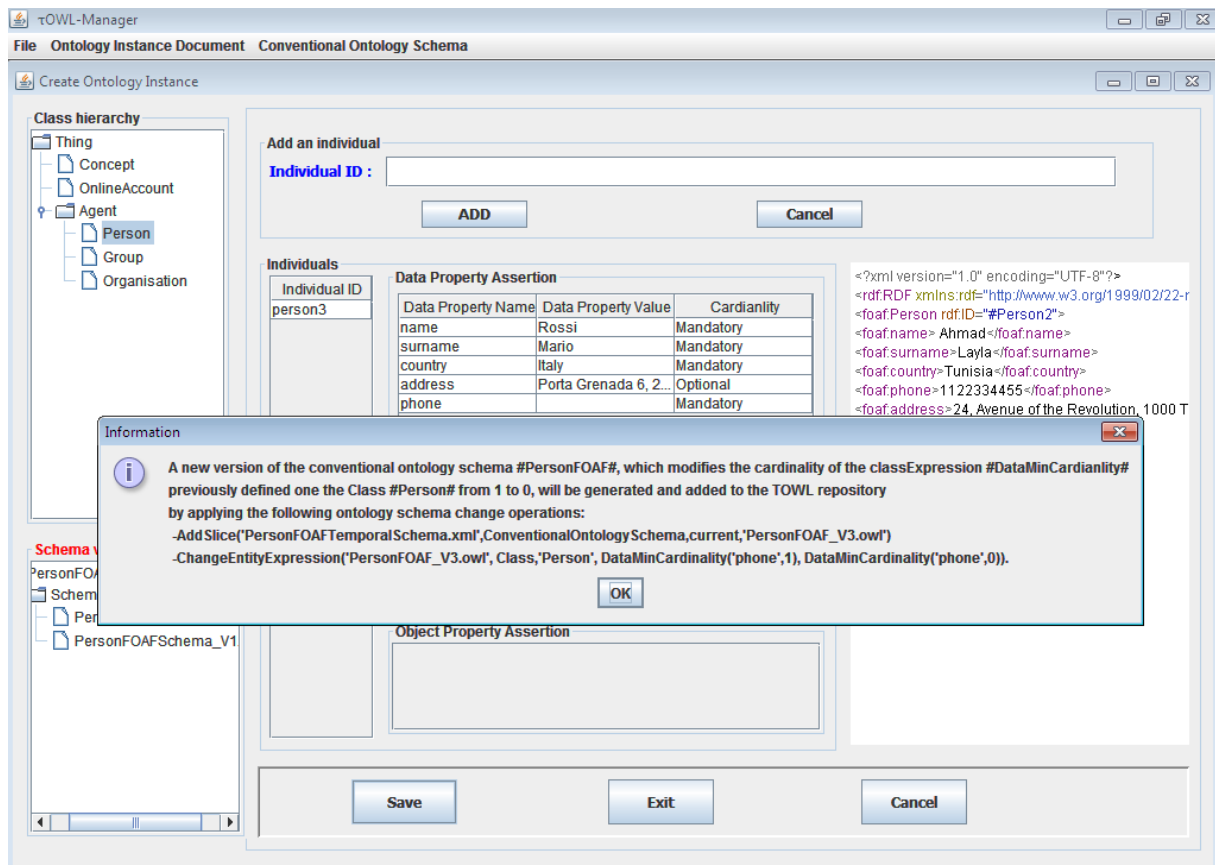


Figure 14. Step 3 of the ontology instance update process session of example 2. The tool (i) informs the ontology administrator that a new conventional ontology schema version will be created, corresponding to the new “Person” instance and changing the data property “phone” of the class “Person” from mandatory (cardinality = “1”) to optional (cardinality = “0”), and (ii) asks him/her to press either the button “OK” to continue the execution of the ontology instance insertion operation (which is implicitly accompanied with an ontology schema change), or the button “CANCEL” otherwise.

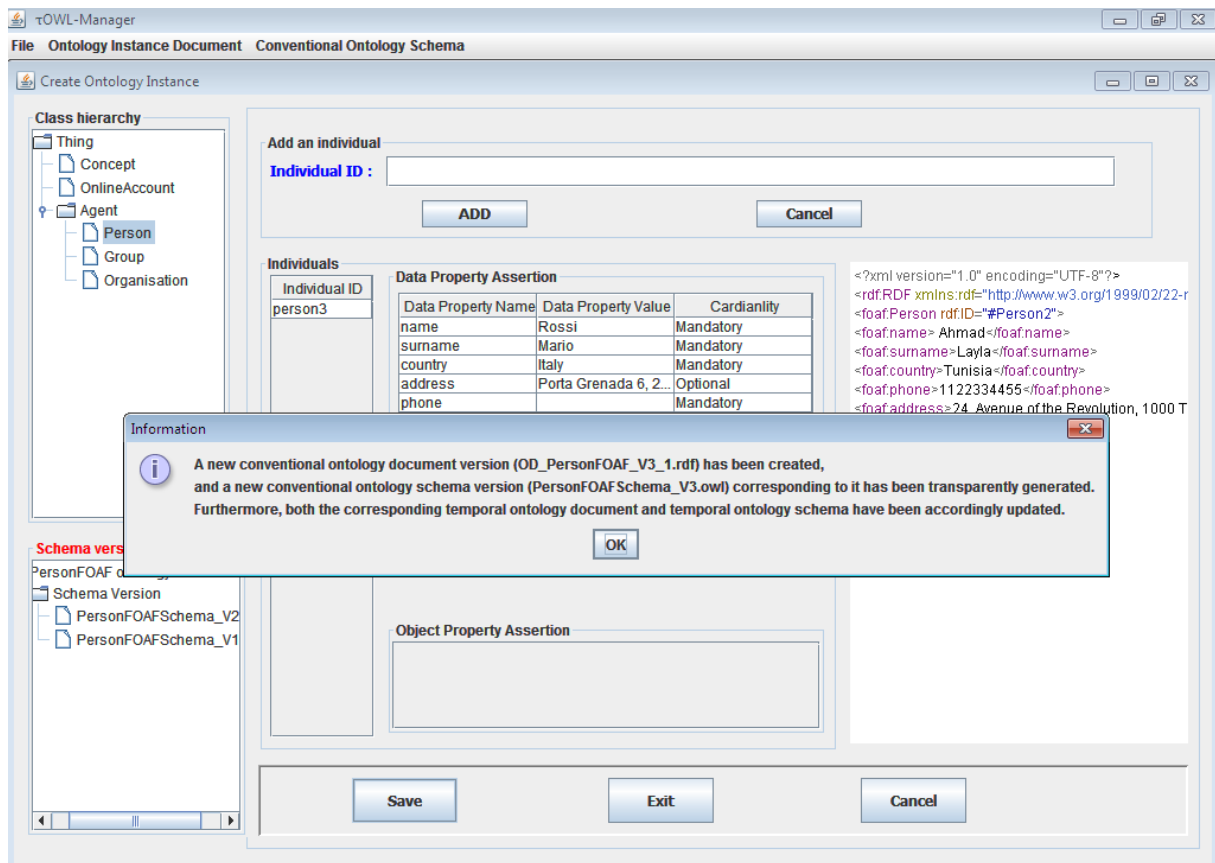


Figure 15. Step 4 of the ontology instance update process session of example 2. The tool informs the ontology administrator that (i) both a new conventional ontology instance document version (to store the new “Person” instance) and a new conventional ontology schema version, corresponding to this conventional ontology instance document version, have been created, (ii) the temporal ontology document has been updated to take into account the new slice corresponding to the new instance document version, and (iii) the temporal ontology schema has been also updated to include the new schema version.

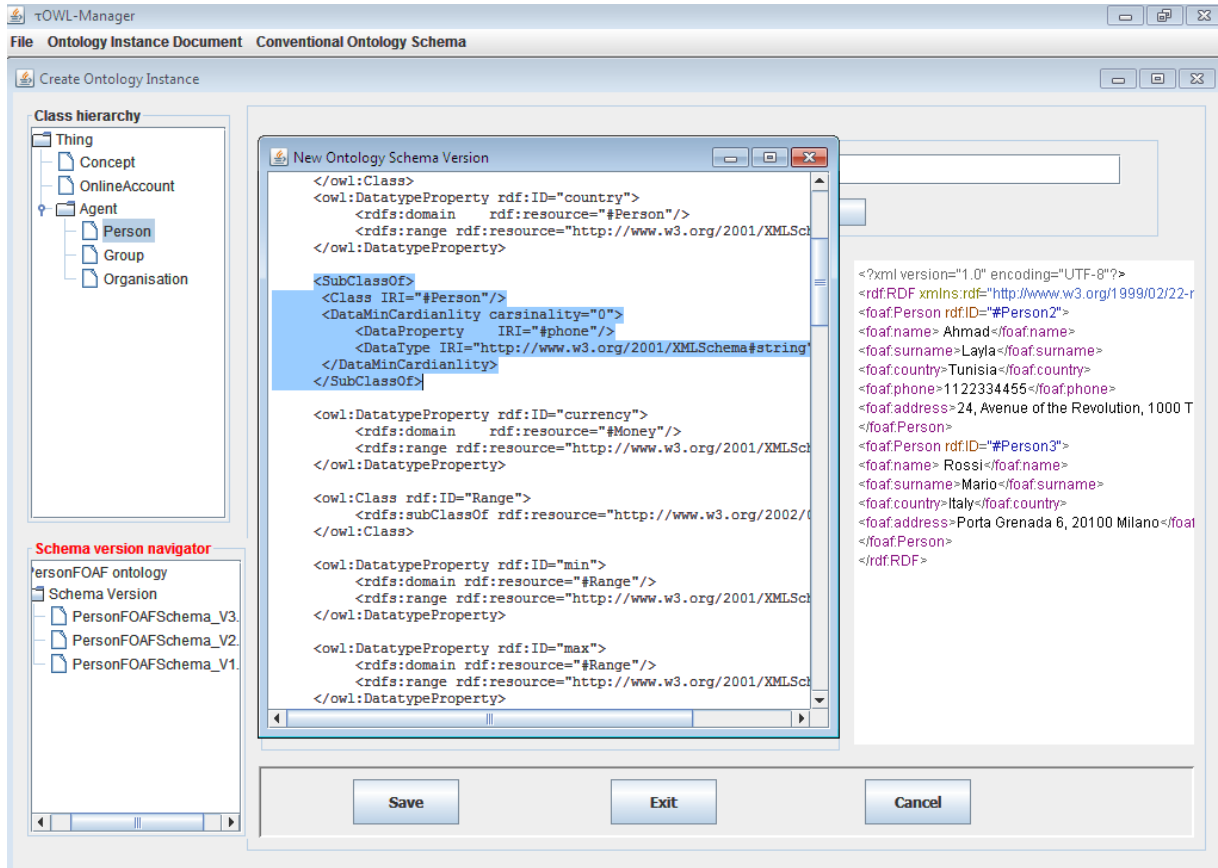


Figure 16. Step 5 of the ontology instance update process session of example 2. The tool displays both the code of the new conventional ontology instance document version and that of the new conventional ontology schema version.

Besides, it is worth mentioning that the effectiveness of our τOWL-Manager tool has also been evaluated by several colleagues, through several and different examples; it has given always the expected results. Therefore, we think that we achieved the objective stated in Section 1.3, and that our tool behaves as expected in order to *interactively accomplish implicit ontology versioning driven by the modifications to the ontology instances, effected by the ontology administrator*.

5. Related Work Discussion

Managing changes to ontologies is an important and challenging topic that has been widely studied in the literature. In (Flouris *et al.*, 2008), the authors provide a good survey of all aspects related to the ontology change issue. They first define this issue as the generic process of modifying an ontology in response to a certain need and managing the effects of such a modification on all depending components (data, services, applications, agents, other ontologies, etc.). After that they identify and study eleven subfields of ontology change: ontology mapping, morphism, matching, alignment, articulation, translation, evolution,

debugging, versioning, integration and merging. Our work is closely related to the subfields “ontology evolution”, which means keeping always a single ontology schema version (i.e., the last updated one) with its set of ontology instance versions (all ontology instance versions that were conforming to the previous ontology schema version are automatically adapted to the new ontology schema version), and “ontology versioning”, which consists in keeping all ontology schema versions with their corresponding ontology instance versions. Notice that in that survey, the authors have not mentioned any work that deals with non-conservative updates or with ontology schema changes triggered by ontology instance updates.

In a recent work, Zablith *et al.* (2015) have given an excellent literature review on the “ontology evolution” topic in the broad sense of this term: from ontology change/adaptation/evolution (e.g., (Noy & Musen, 2002; Klein, 2004; Plessers *et al.*, 2007; Papavassiliou *et al.*, 2013)) to ontology versioning (e.g., (Heflin & Hendler, 2000a; Klein & Fensel, 2001; Klein *et al.*, 2002; Redmond *et al.*, 2008; Allocca *et al.*, 2009a; Grandi, 2009)). The authors present and compare approaches for managing ontology updates, at ontology instance level and/or at ontology schema level, while maintaining or not ontology versions. However, none of the reviewed proposals has tackled ontology instance updates that lead to ontology schema changes or to ontology schema versions.

Hence, all works presented and discussed in (Zablith *et al.*, 2015) and dealing with ontology instance updates and/or ontology schema change/evolution/versioning could be considered as related to our work. In the following, we will discuss only works that have dealt with ontology instance/schema change/evolution/versioning and that either have not been covered by such a survey or have been published after 2015.

Grandi (2009) proposes a multi-temporal RDF database model and provides a set of manipulation operations which allow the management of temporal versions of an ontology. Grandi (2011) extends such a model to support temporal versioning of light-weight RDF(S) ontologies and introduces a set of change operations for defining and managing temporal schema versions of an RDF(S) ontology. Similarly to (Grandi, 2009) and to (Grandi, 2011), our present work handles temporal versioning of ontology instances and temporal versioning of ontology schemas, respectively, but in a different temporal Semantic Web framework. However, both (Grandi, 2009) and (Grandi, 2011) have not studied ontology instance updates that require ontology schema changes.

Grandi (2013) provides primitive operations that can be used for the maintenance of the class structure of a multi-version ontology (embodying a tree-shaped class hierarchy). Grandi (2016) 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 (Grandi, 2013) and (Grandi, 2016), we focus on

changes to ontology instance documents, which give rise to implicit and automatic ontology schema changes.

Jaziri *et al.* (2010) introduce a tool-supported approach for ontology evolution (i.e., keeping always the last ontology version) which allows adapting an ontology while preserving its consistency. This work has been extended by the authors of (Sassi *et al.*, 2016) who have proposed an ontology versioning approach and a tool, called Consistology, which support (i) the management of ontology versions based on their relevance, which is evaluated according to four criteria (i.e., conceptualization, usage frequency, abstraction, and completeness), and (ii) the definition of a process that “reduces” the number of stored ontology versions: when a maximum is reached, the least relevant ontology versions are removed. Notice here that the maximum number of ontology versions and the selection of relevance criteria to be applied are managed by the tool user. In our current paper, we do not study ontology version relevance or reduction of ontology version number. Besides, we think that removing ontology versions is a dangerous action since it leads to ontology loss and therefore to several problems in all components (applications, agents, services, other ontologies ...) depending on the removed ontology. Furthermore, in our environment, we consider that any ontology version has been created as a response to some actual requirement(s) and thus all ontology versions are important and have the same relevance.

Im *et al.* (2012) propose a framework for managing ontology versioning in RDF triple stores. Their approach is based on (i) storing the original RDF data version and the deltas between each two consecutive versions, in a relational database, and (ii) constructing any version, on the fly, through an SQL statement that applies the corresponding deltas to the original version. The main advantage of this approach is the reduction of the storage space. Contrarily to this work, we keep all versions of any manipulated ontology. Moreover, our proposal allows the ontology administrator/user performing any change to an ontology instance document whereas the framework proposed in (Im *et al.*, 2012) considers only insert and delete operations. The authors consider that a triple update operation could be effected through a deletion of the old triple followed by an insertion of the new triple. Notice that while this modeling of the triple update operation could be acceptable at instance level, as it is safe, it may give rise to some problems at schema level.

In (Khattak *et al.*, 2013), the authors propose a tool-supported framework for managing change history in evolving web ontologies. It supports features related to ontology change like ontology versioning, change provenance, ontology consistency, ontology recovery, change representation, and change visualization. Ontology changes are stored in a temporal triple store, named the Change History Log (CHL), which is a main component of the framework. Logged ontology changes allow recovering any previous ontology version in the case of unauthorized changes, version conflicts, or an inconsistent/incomplete ontology version

resulting e.g. from an accidental closing of the ontology editor. Our approach does not include an ontology change log but it stores all ontology instance/schema versions while timestamping them with their transaction times (thanks to the temporal ontology schema and the temporal ontology document components). Therefore, our proposal allows the ontology administrator to obtain any previous ontology instance/schema version that has been valid at a given instant or during a given time interval.

Taleb *et al.* (2014) propose and implement a method for comparing OWL ontology versions and extracting differences between them. Our approach compares both ontology instance versions and ontology schema versions, before recording them; if a new ontology instance/schema version is identical to the previous one, it is not accepted.

The paper of Flahive *et al.* (2015) has dealt with updating or adapting large ontologies in a semantic grid environment. The authors have formalized and validated the ontology update process while some sections of one ontology are replaced by a subset extracted from another ontology. However, the authors have not talked about ontology instance updates that require ontology schema changes.

In the context of ontologies that are defined under the RDF model, Kondylakis and Papadakis (2018) have proposed EvoRDF, a framework to explore ontology evolution through provenance queries. It is based on a high-level language for changing ontologies and allows answering three types of queries: (i) “when” queries, looking for the introduction time of a resource, (ii) “how” queries, seeking by which change operations a resource has been introduced, and (iii) “why” queries, searching the sequence of change operations that led to the creation of a resource in the current ontology version. Further, Taelman *et al.* (2019) have proposed a technique for indexing RDF archives, which allows storing datasets with a low storage overhead, through the compression of consecutive versions and the addition of metadata to reduce lookup times. They have also defined algorithms, which are based on this technique, to evaluate queries in such a multi-version environment.

Kozierkiewicz and Pietranik (2019) have studied revalidation of ontology alignment (i.e., mapping defined between two ontologies), which is triggered by evolution of one or both of the participating ontologies, and more precisely by changes performed on concepts of these ontologies. However, the environment in which the study has been effected does not support ontology versioning. Besides, it is worth mentioning that in our present work, we have not dealt with the impact of ontology changes on ontology alignment.

In (Bayoudhi *et al.*, 2019), the authors have proposed an approach and a Protégé plug-in for versioning of OWL 2 DL ontologies, while applying an *a priori* consistency management technique (i.e., checking ontology consistency before applying ontology changes) and storing ontology versions in a temporal object-oriented database whose (temporal) schema is based on both the direct model-theoretic semantics for OWL 2 (W3C, 2012c) and on the proposal of

Zhang *et al.* (2015) dealing with the storage of OWL 2 ontologies in object-oriented databases. However, the approach of Bayoudhi *et al.* (2019) does not support implicit schema changes triggered by non-conservative instance updates (not taken into account in this approach).

Bürger *et al.* (2020) have proposed an approach implemented in a system prototype for the detection of occurrences of semantic editing patterns, defined as graph transformation rules, between two versions of an OWL ontology. Moreover, they have evaluated their proposals in a medical information system, while considering ontologies for software security. Nevertheless, this approach supports neither ontology schema versioning (only the last version of the ontology is kept), nor ontology instance versioning.

In (Cardoso *et al.*, 2020), first a new technique, named the historical knowledge graph (HKG), has been proposed to store into one single knowledge graph all versions of an ontology along with a log of corresponding changes (which includes all changes applied on each version of this ontology). After that, the authors have experimentally evaluated the HKG through three use-cases in the biomedical area. Contrarily to the approach of Cardoso *et al.* (2020), ours stores ontology versions in files that are independent on those that store ontology change operations. We think that although the HKG could reduce the storage space associated to multi-versions ontologies and their changes logs, it introduces new difficulties like: (i) the necessity to define new efficient indexes for HKG files that are in general very large, and (ii) the need to revise specifications of existing ontology query/update languages and ontology tools, which were designed to work on files that only contain ontologies created with one of the existing ontology languages like OWL, OWL 2, RDF, RDF/XML or RDFS. Besides, Cardoso *et al.* (2020) have considered neither non-conservative ontology instance updates, nor implicit ontology schema changes.

In (Priya and Kumar, 2020), the topic of ontology merging has been dealt with while focusing on the ontology heterogeneity that does not allow interoperability between heterogeneous ontologies. More precisely, the authors have proposed an algorithm, named “pseudo-intent with backtracking-based FCA-Merge”, for ontology merging through the use of formal concept analysis (FCA) (Ganter and Wille, 2012). This algorithm executes four processes in order to merge two ontologies (inputs) into a single ontology (output). These processes are as follows: (i) identification of decision tree-based attributes, (ii) generation of linked lists to increase the sparse matrix size, (iii) performing backtracking to reduce the combinatorial problems, and (iii) execution of merging based on the generated linked lists. The proposed algorithm has been experimentally evaluated with respect to three existing methods for ontology merging (FCA-Merge, OntEx, and FCA-Map); it provides 97% of precision, 82% of recall, and 89% of accuracy, which are all higher than the other existing methods.

Recently, Santos *et al.* (2020) have proposed an experimental analysis concerning a set of

existing tools that are used for the management of ontology evolution. The tools are: PromptDiff (Noy *et al.*, 2002), WebProtégé (Horridge *et al.*, 2019), CODEX (Hartung *et al.*, 2012), KAON (Stojanovic *et al.*, 2002), OWL Diff (Redmond & Noy, 2011), OntoDiffGraph (Lara *et al.*, 2017), and OIM (Davidovsky *et al.*, 2011). The authors have evaluated them based on the following four criteria: change detection, complex change detection, change inspection, and extensibility. As for the “change detection” criterion, it deals with how primitive ontology changes are detected. Two techniques are identified: (i) keeping track of the history of ontology changes that are performed by the user using tool, and (ii) generating the difference between successive ontology versions. As for the “complex change detection”, it concerns the capability of a tool to (automatically) detect a complex ontology change from a sequence of primitive ontology changes applied by the user. As for the “change inspection” criterion, it concerns the representation of the applied ontology changes, which can be either textual (e.g., via a log file) or graphical (e.g., as a diagram). The “extensibility” criterion is dealing with the possibility to extend the provided tool (taking into account facts like the availability of its source code). Notice that, with regard to these four criteria, the τ OWL-Manager tool detects ontology changes through the history of actions done by the ontology administrator on his/her GUI used to update ontology instances or ontology schema. Furthermore, in its current release, τ OWL-Manager only allows to explicitly perform complex ontology changes at schema level, and is not able to detect a complex change from a list of primitive changes effected at instance/schema levels. As far as the ontology change inspection issue is concerned, our tool uses a textual format to represent the effected ontology changes. Considering the fourth criterion, τ OWL-Manager is extensible and its source code could be provided upon request.

In (Bayoudhi *et al.*, 2020), the authors have surveyed (among others) approaches for ontology versioning while focusing on three research topics: (i) ontology versions pertinence, like in (Sassi *et al.*, 2016), (ii) ontology versions relationship, like in (Allocca *et al.*, 2009b) and (Díaz *et al.*, 2011), and (iii) ontology versions storage and querying, like in (Grandi, 2013), (Grandi, 2016), (Meimaris, 2018), (Bayoudhi *et al.*, 2019), and (Taelman *et al.*, 2019).

The works, which are more strictly related with our approach, are (Zekri *et al.*, 2014), (Zekri *et al.*, 2015a), (Zekri *et al.*, 2015b), (Zekri *et al.*, 2016), and (Zekri *et al.*, 2017).

Zekri *et al.* (2014) introduce τ OWL, a τ XSchema-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 (i) separates conventional schema, logical annotations, and physical annotations, and (ii) allows each one of these three components to be changed independently and safely. The present work extends (Zekri *et al.*, 2014) by (i) proposing an approach for implicit schema versioning in τ OWL and (ii) focusing on non-conservative changes to conventional ontology

instance documents, which lead to the versioning of the corresponding conventional ontology schema. Zekri *et al.* (2016) extend the work presented in (Zekri *et al.*, 2014) by showing how, in the τ OWL framework, temporal ontology instance versioning could be managed in the presence of temporal ontology schema versioning, simultaneously and in a consistent manner. In (Zekri *et al.*, 2015a), 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 τ OWL context. Zekri *et al.* (2017) propose an approach, which extends the contribution of Zekri *et al.* (2016), for managing time-varying knowledge, since an ontology can be used for knowledge management. Although the proposals presented in (Zekri *et al.*, 2015a), (Zekri *et al.*, 2016), and (Zekri *et al.*, 2017) have focused on ontology schema versioning in τ OWL, none of them has dealt with ontology schema changes generated by ontology instance updates. Zekri *et al.* (2015b) present τ OWL-Manager, a prototype tool that allows defining temporal ontology schema and managing temporal versioning of temporal ontology instances, in the τ OWL framework, while supporting only conservative updates to ontology instances. Zekri *et al.* (2016) extend (Zekri *et al.*, 2015b) by adding schema versioning support to τ OWL-Manager and showing its functionalities. Our present work extends τ OWL-Manager to consistently support both non-conservative updates to conventional ontology instance documents and implicit conventional ontology schema changes triggered by these instance updates.

Notice that in the XML world, Bouchou *et al.* (2004) have talked about invalid updates to XML (instance) documents, which are XML data updates that give rise to XML documents not valid to their corresponding XML schemas. Invalid documents must be adapted to meet the changed XML schema, which is a dangerous operation as it could lead to some data loss when some updates require removing some XML schema components. Although non-conservative ontology instance updates are similar to invalid XML document updates, they are studied in different environments: invalid updates have been investigated in a non-temporal XML environment that supports only schema evolution (i.e., only the last XML schema version is kept with its XML instances), whereas non-conservative ontology instance updates have been considered in a temporal and multi-schema-version OWL 2 ontology environment (i.e., τ OWL). Furthermore, our approach does not lead to any data loss, as both instances and schemas are temporally versioned. Moreover, Bouchou and Duarte (2007) have talked about non-conservative schema evolution in an XML environment, which means that XML documents that are valid to an XML schema before its change could become not valid with respect to such a schema after its change. Similarly to the approach introduced by Bouchou *et al.* (2004), the authors of (Bouchou and Duarte, 2007) have also proposed that existing XML documents, which are no longer valid to the new XML schema, must be adapted to this latter. Contrarily to the approach of Bouchou and Duarte (2007), which has some disadvantages like data loss, our approach deals with non-conservative updates to

ontology instance documents, in an environment that supports ontology schema versioning, and therefore, it avoids several problems like loss of data and schemas, and system downtimes that are necessary to revise source codes of programs that are using the updated ontology instance documents and/or the changed ontology schemas.

In Table 1, we compare the ontology evolution/versioning approaches of the literature to our approach, while taking into account the following seven criteria:

- Ontology language (e.g., RDFS, OWL, OWL 2, OWL DL, UML), representing the ontology definition language on which the approach is based.
- Implementation (Yes or No), telling whether the approach has been implemented.
- Support of temporal aspects (Yes or No), telling whether the approach supports temporal aspects. If “Yes”, it is also mentioned at which level they are supported (instance level and/or schema level).
- Ontology instance change technique (Instance Evolution, if only the last instance version is kept, or Instance Versioning).
- Ontology schema change technique (Schema Evolution or Schema Versioning).
- Support of non-conservative updates to ontology instances (Yes or No).
- Support of implicit ontology schema changes (Yes or No).

Table 1. Comparison of approaches dealing with ontology evolution/versioning.

Approach	Ontology language	Implementation	Support of temporal aspects	Ontology instance change technique	Ontology schema change technique	Support of non-conservative updates to ontology instances	Support of implicit ontology schema changes
Grandi (2009)	RDF	No	Yes	Instance Versioning	-	No	No
Grandi (2011)	RDF(S)	No	Yes	Instance Versioning	Schema Versioning	No	No
Grandi (2013)	No specific language; tree-like ontology	No	No	Instance Versioning	Schema Versioning	No	No
Grandi (2016)	No specific language; general graph ontology	Yes	No	Instance Versioning	Schema Versioning	No	No
Jaziri <i>et al.</i> (2010)	UML	Yes	No	Instance Evolution	Schema Evolution	No	No
Sassi <i>et al.</i> (2016)	UML	Yes	No	Instance Versioning	Schema Versioning	No	No
Im <i>et al.</i> (2012)	RDF	No	No	Instance Versioning	-	No	No
Khattak <i>et al.</i>	RDFS,	Yes	No	Instance	Schema	No	No

(2013)	OWL			Versioning	Versioning		
Taleb <i>et al.</i> (2014)	OWL	Yes	No	Instance Versioning	-	No	No
Flahive <i>et al.</i> (2015)	Generic	No	No	Instance Versioning	-	No	No
Kondylakis and Papadakis (2018)	RDF	Yes	No	Instance Evolution	Schema Evolution	No	No
Taelman <i>et al.</i> (2019)	RDF	Yes	No	Instance Versioning	-	No	No
Kozierkiewicz and Pietranik (2019)	Generic	No	No	Instance Versioning	Schema Versioning	No	No
Bayoudhi <i>et al.</i> (2019)	OWL 2 DL	Yes	No	Instance Versioning	Schema Versioning	No	No
Bürger <i>et al.</i> (2020)	OWL	Yes	No	Instance Versioning	Schema Versioning	No	No
Cardoso <i>et al.</i> (2020)	Historical Knowledge Graph	Yes	No	Instance Versioning	Schema Versioning	No	No
Priya and Kumar (2020)	FCA	Yes	No	Instance Evolution	Schema Evolution	No	No
Zekri <i>et al.</i> (2016), Zekri <i>et al.</i> (2017)	OWL 2	Yes	Yes; at both instance and schema levels	Instance Versioning	Schema Versioning	No	No
Our approach	OWL 2	Yes	Yes; at both instance and schema levels	Instance Versioning	Schema Versioning	Yes	Yes

Moreover, in Table 2, we summarize the features of most popular existing tools for ontology evolution/versioning and of our τ OWL-Manager tool, taking into account the following six aspects:

- Ontology language (e.g., RDFS, OWL, OWL 2, OWL DL, UML), representing the ontology definition language on which the approach is based.
- Support of temporal aspects (Yes or No), telling whether the tool supports temporal aspects. If “Yes”, it is also mentioned at which level they are supported (instance level and/or schema level).
- Support of ontology instance versioning (Yes or No).
- Support of ontology schema versioning (Yes or No).
- Support of non-conservative updates to ontology instances (Yes or No).
- Support of implicit ontology schema changes (Yes or No).

Table 2. Comparison of research tools dealing with ontology evolution/versioning.

Tool	Ontology language	Support of temporal aspects	Support of Ontology instance Versioning	Support of Ontology Schema Versioning	Support of Non-conservative Updates to Ontology Instances	Support of Implicit Ontology Schema Changes
PromptDiff (Noy <i>et al.</i> , 2002)	OWL	No	No	No	No	No
OWL Diff (Redmond & Noy, 2011)	OWL2	No	No	No	No	No
OntoDiffGraph (Lara <i>et al.</i> , 2017)	VOWL	No	No	No	No	No
CODEX (Hartung <i>et al.</i> , 2012)	OBO	No	No	No	No	No
WebProtégé (Horridge <i>et al.</i> , 2019)	OWL 2, OWL 2 EL, OBO	No	No	No	No	No
KAON (Stojanovic <i>et al.</i> , 2002)	KAON language	No	No	No	No	No
OIM (Davidovsky <i>et al.</i> , 2011)	OWL	No	Yes	Yes	No	No
Consistology (Sassi <i>et al.</i> , 2016)	UML	No	No	Yes	No	No
Bayoudhi <i>et al.</i> (2019)’s tool	OWL 2 DL	No	Yes	Yes	No	No
Our τ OWL-Manager tool (in its last release)	OWL 2	Yes; at both instance and schema levels	Yes	Yes	Yes	Yes

6. Conclusion

In this paper, we have proposed a new general approach for managing ontology instance updates that require ontology schema changes in order to produce a consistent result, in an environment that supports temporal versioning of both ontology instances and ontology schemas. We have implemented this approach in the τ OWL framework, extending it to support two new aspects: non-conservative updates to ontology instances and ontology schema changes that are triggered by such instance updates. In order to show the feasibility of our approach, we have developed a new version of the τ OWL-Manager, which allows ontology administrators to perform ontology instance updates that require implicit ontology schema changes. With the developed tool, non-conservative updates can be executed in τ OWL-based Semantic Web repositories, while managing instance and schema changes consistently and guaranteeing a full history of evolving conventional ontology instances and

schemata. The required implicit ontology schema changes are automatically generated and, upon approval by the ontology administrator, executed.

To the best of our knowledge, we are the first to deal with these two interesting issues in an ontology environment that supports versioning at both instance and schema levels: non-conservative updates to ontology instances and implicit ontology schema changes. We have also applied it to an established framework (i.e., τ OWL) for managing evolving ontologies and implemented it within a tool prototype (i.e., τ OWL-Manager). We think that our proposal provides more flexibility in the management of ontology evolution and ontology versioning, guarantees safety of the updated ontology instance documents and the ontology schema documents, and makes it easy to ontology administrators, especially if they have no sufficient skills to extricate themselves with devising and applying ontology schema changes, the task of effecting non-conservative updates to conventional ontology instance documents. Moreover, from an ontology engineering viewpoint, the τ OWL-Manager provides a practical environment which can be used to support an incremental ontology construction method driven by the gradual availability of instance data.

As a part of our future work, we intend to study the interesting second method for updating ontology instances: the non-interactive one. Such a method is applied when the ontology administrator would like to integrate, into the τ OWL repository, a new conventional ontology instance document (e.g., which has been prepared offline or which is imported from an external source), that must be merged with the current ontology instance document to produce a new ontology instance document version. The works already done by Papavassiliou *et al.* (2009), on ontology change detection in knowledge bases, and by Gröner *et al.* (2010), on semantic recognition of ontology refactorings, could help us in this direction.

Furthermore, although retroactive and proactive updates to ontology instances are required (since we are in a temporal environment) to perform some instance updates with retroactive or proactive effect respectively, they are not allowed by the τ OWL-Manager in its current release, as only transaction-time versioning is supported at both instance and schema levels; consequently, only on-time updates are allowed. In our future work, we intend to extend both τ OWL and τ OWL-Manager to also support valid-time schema versioning and/or bitemporal schema versioning (De Castro *et al.*, 1997), so that they will also provide appropriate functionalities to execute such updates.

Moreover, we also plan to extend our current proposal to deal with propagating, in an automatic and safe manner, the effects of ontology schema changes to the possible other ontologies that are using or extending the evolved ontology, and to all artifacts (e.g., application programs, services, scripts, agents, web pages) that are relying on the ontology whose schema is changed; such an efficient propagation will be very helpful for ontology administrators.

Last but not least, in the future, we aim at studying the following aspect that is related to our present work and that we consider very interesting for today's semantic web applications: performing on-the-fly ontology schema changes which are necessary to the support of high-availability applications (like Internet of Things, cloud computing and e-health applications).

References

- Al-Jadir, L., Parent, C., & Spaccapietra, S. (2010). Reasoning with large ontologies stored in relational databases: The OntoMinD approach. *Data and Knowledge Engineering*, 69(11), 1158-1180.
- Allocca, C., d'Aquin, M., & Motta, E. (2009a). Detecting different versions of ontologies in large ontology repositories. In *Proceedings of the 3rd International Workshop on Ontology Dynamics (IWOD 2009)*. CEUR Workshop Proceedings, Vol-519. <http://ceur-ws.org/Vol-519/allocca.pdf>
- Allocca, C., d'Aquin, M., & Motta, E. (2009b). Door: Towards a formalization of ontology relations. In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development (KEOD 2009)*, pp. 13-20.
- Antoniou, G., Groth, P., Harmelen, F. V., & Hoekstra, R. (2012). *A Semantic Web Primer (3rd edition)*. The MIT Press, Cambridge, London, England.
- Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F., & Zakharyashev, M. (2017). Ontology-Mediated Query Answering over Temporal Data: A Survey. In *Proceedings of the 24th International Symposium on Temporal Representation and Reasoning (TIME'2017)*, Mons, Belgium, 16-18 October 2017, pp. 1:1-1:37.
- Bayoudhi, L., Sassi, N., & Jaziri, W. (2019). Efficient management and storage of a multiversion OWL 2 DL domain ontology. *Expert Systems*, 36(2), e12355.
- Bayoudhi, L., Sassi, N., & Jaziri, W. (2020). A Survey on Versioning Approaches and Tools. In *Proceedings of the International Conference on Intelligent Systems Design and Applications (ISDA 2020)*, pp. 1155-1164. Springer, Cham.
- Bergman, M. (2009). *The Fundamental Importance of Keeping an ABox and TBox Split - Ontology Best Practices for Data-driven Applications: Part 2*, 17 May 2009. <http://www.mkbergman.com/489/ontology-best-practices-for-data-driven-applications-part-2/>
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5), 34-43.
- Bouchou, B., Duarte, D., Alves, M. H. F., Laurent, D., & Musicante, M. A. (2004). Schema evolution for XML: A consistency-preserving approach. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science (MFCS'2004)*, pp. 876-888. Springer, Berlin, Heidelberg.
- Bouchou, B., & Duarte, D. (2007). Assisting XML Schema Evolution that Preserves Validity. In *Proceedings of the 22nd Brazilian Symposium of Databases (SBBD'2007)*, pp. 270-284.
- Brahmia, S., Brahmia, Z., Grandi, F., & Bouaziz, R. (2017). Temporal JSON Schema Versioning in the τ JSchema Framework. *Journal of Digital Information Management*, 15(4), 179-202.
- Brahmia, Z., Grandi, F., Oliboni, B., & Bouaziz, R. (2014a). Schema Change Operations for Full Support of Schema Versioning in the τ XSchema Framework. *International Journal of Information Technology and Web Engineering*, 9(2), 20-46.

- Brahmia, Z., Grandi, F., Oliboni, B., & Bouaziz, R. (2014b). High-level Operations for Creation and Maintenance of Temporal and Conventional Schema in the τ XSchema Framework. In *Proceedings of the 21st International Symposium on Temporal Representation and Reasoning (TIME'2014)*, Verona, Italy, 8-10 September 2014, pp. 101-110.
- Brahmia, Z., Grandi, F., Oliboni, B., & Bouaziz, R. (2018). Schema Versioning in Conventional and Emerging Databases. In: Mehdi Khosrow-Pour, (Ed.), *Encyclopedia of Information Science and Technology (4th edition)*, IGI Global, Hershey, PA, USA, 2018, pp. 2054-2063.
- Bürger, J., Kehrer, T., & Jürjens, J. (2020). Ontology Evolution in the Context of Model-Based Secure Software Engineering. In *Proceedings of the 14th International Conference on Research Challenges in Information Science (RCIS'2020)*, pp. 437-454. Springer, Cham.
- Burns, T., Fong, E., Jefferson, D., Knox, R., Mark, L., Reedy, C., Reich, L., Roussopoulos, N., & Truszkowski, W. (1986). Reference Model for DBMS Standardization, Database Architecture Framework Task Group (DAFTG) of the ANSI/X3/SPARC Database System Study Group. *SIGMOD Record*, 15(1), 19-58.
- Cardoso, S. D., Da Silveira, M., & Pruski, C. (2020). Construction and exploitation of an historical knowledge graph to deal with the evolution of ontologies. *Knowledge-Based Systems*, 194, 105508.
- Davidovsky, M., Ermolayev, V., & Tolok, V. (2011). Instance migration between ontologies having Structural Differences. *International Journal on Artificial Intelligence Tools*, 20(06), 1127-1156.
- De Castro, C., Grandi, F., & Scalas, M. R. (1997). Schema Versioning for Multitemporal Relational Databases. *Information Systems*, 22(5), 249-290.
- Díaz, A., Motz, R., & Rohrer, E. (2011). Making ontology relationships explicit in a ontology network. In *Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2011)*. CEUR Workshop Proceedings, Vol-749, paper14. <http://ceur-ws.org/Vol-749/paper14.pdf>
- Etzioni, O., Golden, K., & Weld, D. S. (1997). Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, 89(1-2), 113-148.
- Flahive, A., Taniar, D., Rahayu, J. W., & Apduhan, B. O. (2015). A methodology for ontology update in the semantic grid environment. *Concurrency and Computation: Practice and Experience*, 27(4), 782-808.
- Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., & Antoniou, G. (2008). Ontology change: classification and survey. *Knowledge Engineering Review*, 26(2), 117-152.
- Ganter, B., & Wille, R. (2012). *Formal concept analysis: mathematical foundations*. Springer Science & Business Media.
- Grandi, F. (2009). Multi-temporal RDF ontology versioning. In *Proceedings of the 3rd International Workshop on Ontology Dynamics (IWOD 2009)*, Washington, D.C., USA, 26 October 2009. CEUR Workshop Proceedings, Vol-519. <http://ceur-ws.org/Vol-519/grandi.pdf>
- Grandi, F. (2010). T-SPARQL: a TSQ2-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 2010, pp. 21-30.
- Grandi, F. (2011). Light-weight Ontology Versioning with Multi-temporal RDF Schema. In *Proceedings of the 5th International Conference on Advances in Semantic Processing (SEMANTAPRO 2011)*, Lisbon, Portugal, 20-25 November 2011, pp. 42-48.

- 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 2013, pp. 224-232.
- Grandi, F. (2016). Dynamic class hierarchy management for multi-version ontology-based personalization. *Journal of Computer and System Sciences*, 82(1), 69-90.
- Gröner, G., Parreiras, F. S., & Staab, S. (2010). Semantic Recognition of Ontology Refactoring. In *Proceedings of the 9th International Semantic Web Conference (ISWC 2010)*, Shanghai, China, 7-11 November 2010, Revised Selected Papers, Part I, pp. 273-288.
- Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5-6), 907-928.
- Guarino, N., (Ed.) (1998). *Formal Ontology in Information Systems*, IOS Press, Amsterdam, Netherlands.
- Hartung, M., Groß, A., & Rahm, E. (2010). Rule-based generation of diff evolution mappings between ontology versions. *arXiv preprint arXiv:1010.0122*.
- Heflin, J., & Hendler, J. (2000a). Dynamic ontologies on the web. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI 2000)*, Austin, Texas, USA, 30 July - 3 August 2000, pp. 443-449.
- Heflin, J., & Hendler, J. (2000b). Searching the Web with SHOE. In: *Artificial Intelligence for Web Search. Papers from the 2000 AAAI Workshop WS-00-01*, 2000, pp. 35-40. Menlo Park, CA: AAAI Press.
- Heflin, J., & Muñoz-Avila, H. (2002). LCW-based agent planning for the semantic web. In: *Ontologies and the Semantic Web. Papers from the 2002 AAAI Workshop WS-02-11*, pp. 63-70, Menlo Park, CA, 2002.
- Heymans, S., Ma, L., Anicic, D., Ma, Z., Steinmetz, N., Pan, Y., Mei, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Feier, C., Hench, G., Wetzstein, B., & Keller, U. (2008). Ontology Reasoning with Large Data Repositories. In: M. Hepp, P. De Leenheer, A. de Moor, and Y. Sure (Eds.), *Ontology Management*, CHE 7, Springer-Verlag, Berlin, Germany, 2008, pp. 89-128.
- Horridge, M., & Bechhofer, S. (2011). The OWL API: A Java API for OWL Ontologies. *Semantic Web*, 2(1), 11-21.
- Horridge, M., Gonçalves, R. S., Nyulas, C. I., Tudorache, T., & Musen, M. A. (2019). WebProtégé: A Cloud-Based Ontology Editor. In *Companion Proceedings of The 2019 World Wide Web Conference* (pp. 686-689).
- Im, D. H., Lee, S. W., & Kim, H. J. (2012). A version management framework for RDF triple stores. *International Journal of Software Engineering and Knowledge Engineering*, 22(1), 85-106.
- Jaziri, W., Sassi, N., & Gargouri, F. (2010). Approach and tool to evolve ontology and maintain its coherence. *International Journal of Metadata, Semantics and Ontologies*, 5(2), 151-166.
- Khattak, A. M., Latif, K., & Lee, S. (2013). Change management in evolving web ontologies. *Knowledge Based Systems*, 37, 1-18.
- Klein, M. C. A., & Fensel, D. (2001). Ontology versioning for the semantic web. In *Proceedings of the 1st International Semantic Web Working Symposium (SWWS'2001)*, Stanford University, Stanford, CA, USA, 30 July – 1 August 2001, pp. 75-91.
- Klein, M. C. A., Fensel, D., Kiryakov, A., & Ognyanov, D. (2002). Ontology Versioning and Change Detection on the Web. In *Proceedings of the 13th International Conference on Knowledge Engineering*

- and Knowledge Management (EKAW 2002), Siguenza, Spain, 1-4 October 2002, pp. 197-212.
- Klein, M. C. A. (2004). *Change Management for Distributed Ontologies*, PhD thesis, Vrije University, Amsterdam, Netherlands.
- Kondylakis, H., & Papadakis, N. (2018). EvoRDF: evolving the exploration of ontology evolution. *The Knowledge Engineering Review*, 33, e12. doi:10.1017/S0269888918000140
- Konev, B., Ludwig, M., & Wolter, F. (2012). Logical Difference Computation with CEX2.5. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR 2012)*, Manchester, UK, 26-29 June 2012, pp. 371-377.
- Konys, A. (2016). Ontology-Based Approaches to Big Data Analytics. In *Proceedings of the 20th International Multi-conference on Advanced Computer Systems (ACS 2016)*, Międzyzdroje, Poland, 19-21 October 2016, pp. 355-365.
- Kozierkiewicz, A., & Pietranik, M. (2019). Triggering ontology alignment revalidation based on the degree of change significance on the ontology concept level. In *Proceedings of the 22nd International Conference on Business Information Systems (BIS'2019)*, pp. 137-148. Springer, Cham.
- Kuiler, E. W. (2014). From big data to knowledge: an ontological approach to big data analytics. *Review of Policy Research*, 31(4), 311-318.
- Lambrix, P., Dragisic, Z., Ivanova, V., & Anslow, C. (2016). Visualization for Ontology Evolution. In *Proceedings of the 2nd International Workshop on Visualization and Interaction for Ontologies and Linked Data co-located with the 15th International Semantic Web Conference (VOILA@ISWC 2016)*, Kobe, Japan, 17 October 2016, pp. 54-67.
- Lara, A., Henriques, P. R., & Gançarski, A. L. (2017). Visualization of Ontology Evolution Using OntoDiffGraph. In *Proceedings of the 6th Symposium on Languages, Applications and Technologies (SLATE 2017)*, Article No. 14, pp. 14:1-14:8.
- Liu, Y., Chen, R., Song, Y., & Deng, W. (2014). A fast approach for querying multiple ontology versions based on a concept lattice. *Journal of Web Engineering*, 13(1&2), 97-113.
- Lutz, C., Seylan, I., & Wolter, F. (2012). Mixing Open and Closed World Assumption in Ontology-Based Data Access: Non-Uniform Data Complexity. In *Proceedings of the 2012 International Workshop on Description Logics (DL-2012)*, Rome, Italy, 7-10 June 2012. CEUR Workshop Proceedings, Vol. 846, paper 17.
- Meimaris, M. (2018). *Managing, Querying and Analyzing Big Data on the Web*. Ph.D. Thesis, University of Thessaly, Greece.
- Merrill, E., Corlosquet, S., Ciccarese, P., Clark, T., & Das, S. (2014). Semantic Web repositories for genomics data using the eXframe platform. *Journal of Biomedical Semantics*, 5(S1), Article no. S3.
- Noy, N. F., & Musen, M. A. (2002). PROMPTDIFF: A fixed-point algorithm for comparing ontology versions. In *Proceedings of the 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI 2002)*, 28 July - 1 August 2002, Edmonton, Alberta, Canada, pp. 744-750.
- Noy, N. F., Kunnatur, S., Klein, M. C. A., & Musen, M. A. (2004). Tracking changes during ontology evolution. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, 7-11 November 2004, pp. 259-273.

- O'Connor, M. J., & Das, A. K. (2009). SQWRL: a query language for OWL. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009)*, Chantilly, VA, USA, 23-24 October 2009. CEUR Workshop Proceedings, Vol. 529, pp. 208-215.
- O'Connor, M. J., & Das, A. K. (2011). A Method for Representing and Querying Temporal Information in OWL. In: Fred A, Filipe J, and Gamboa H (Eds.), *Biomedical Engineering Systems and Technologies (Selected Papers)*, CCIS 127, Springer-Verlag, Berlin, Germany, 2011, pp. 97–110.
- O'Connor, M. J. (2016). *SQWRL: a query language for OWL*, GitHub Wiki, 12 August 2016. <https://github.com/protegeproject/swrlapi/wiki/SQWRL>
- Papavassiliou, V., Flouris, G., Fundulaki, I., Kotzinos, D., & Christophides, V. (2009). On Detecting High-Level Changes in RDF/S KBs. In *Proceedings of the 8th International Semantic Web Conference (ISWC 2009)*, Chantilly, VA, USA, 25-29 October 2009, pp. 473-488.
- Papavassiliou, V., Flouris, G., Fundulaki, I., Kotzinos, D., & Christophides, V. (2013). High-level change detection in RDF(S) KBs. *ACM Transactions on Database Systems*, 38(1), 1-42.
- Patel-Schneider, P. F., & Horrocks, I. (2007). A comparison of two modelling paradigms in the Semantic Web. *Journal of Web Semantics*, 5(4), 240-250.
- Plessers, P., De Troyer, O., & Casteleyn, S. (2007). Understanding ontology evolution: A change detection approach. *Journal of Web Semantics*, 5(1), 39-49.
- Preventis, A., Petrakis, E. G. M., & Batsakis, S. (2014). CHRONOS Ed: A Tool for Handling Temporal Ontologies in Protégé. *International Journal on Artificial Intelligence Tools*, 23(4), 1460018.
- Priya, M., & Kumar, C. A. (2020). A novel approach for merging ontologies using formal concept analysis. *International Journal of Cloud Computing*, 9(2-3), 189-206.
- Redmond, T., & Noy, N. (2011). Computing the changes between ontologies. In *Proceedings of the Joint Workshop on Knowledge Evolution and Ontology Dynamics*, pp. 1-14. <http://ceur-ws.org/Vol-784/evodyn6.pdf>
- Redmond, T., Smith, M., Drummond, N., & Tudorache, T. (2008). Managing change: an ontology version control system. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED-08)*. http://ceur-ws.org/Vol-432/owled2008eu_submission_33.pdf
- Roddick, J. F. (2018). Schema Versioning. In: Liu L, and Özsu M. T (Eds.), *Encyclopedia of Database Systems (2nd edition)*, Springer, New York, NY, USA, 2018. DOI: 10.1007/978-1-4614-8265-9
- Santos, J. S., Silva, V. T., Azevedo, L. G., Soares, E. F., & Thiago, R. M. (2020). An Experimental Analysis of Tools for Ontology Evolution Management. In *Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS'2020)*, Volume 2, pp. 111-121.
- Sassi, N., Jaziri, W., & Alharbi, S. (2016). Supporting ontology adaptation and versioning based on a graph of relevance. *Journal of Experimental and Theoretical Artificial Intelligence*, 28(6), 1035-1059.
- Seylan, I., Franconi, E., & De Bruijn, J. (2009). Effective query rewriting with ontologies over DBoxes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, Pasadena, CA, USA, 11-17 July 2009, pp. 923-929.
- Snodgrass, R. T., Dyreson, C. E., Currim, F., Currim, S., & Joshi, S. (2008). Validating Quicksand: Schema Versioning in τ XSchema. *Data and Knowledge Engineering*, 65(2), 223-242.
- Stojanovic, L., Maedche, A., Motik, B., & Stojanovic, N. (2002). User-driven ontology evolution

- management. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management* (pp. 285-300). Springer, Berlin, Heidelberg.
- Taelman, R., Vander Sande, M., Van Herwegen, J., Mannens, E., & Verborgh, R. (2019). Triple storage for random-access versioned querying of RDF archives. *Journal of Web Semantics*, 54, 4-28.
- Taleb, N., Tighiouart, B., & Laiche, S. (2014). A method based on OWL schema for detecting changes between Ontology's versions. *Intelligent Decision Technologies*, 8(1), 45-52.
- Tzitzikas, Y., Theoharis, Y., & Andreou, D. (2008). On storage policies for semantic web repositories that support versioning. In *Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*, Tenerife, Canary Islands, Spain, 1-5 June 2008, pp. 705-719.
- 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 2006, Vol. 1, pp. 195-202.
- W3C (2004a). *XML Schema Part 0: Primer Second Edition*, W3C Recommendation, 28 October 2004. <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>
- W3C (2004b). *RDF/XML Syntax Specification (Revised)*, W3C Recommendation, 10 February 2004. <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>
- W3C (2012a). *OWL 2 Web Ontology Language – Primer (Second Edition)*, W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-primer/>
- W3C (2012b). *OWL 2 Web Ontology Language – Document Overview (Second Edition)*, W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-overview/>
- W3C (2012c). *OWL 2 Web Ontology Language – Direct Semantics (Second Edition)*, W3C Recommendation, 11 December 2012. <http://www.w3.org/TR/owl2-direct-semantics/>
- Zablith, F., Antoniou, G., d'Aquin, M., Flouris, G., Kondylakis, H., Motta, E., Plexousakis, D., & Sabou, M. (2015). Ontology evolution: a process-centric survey. *The Knowledge Engineering Review*, 30(1), 45-75.
- Zekri A, Brahmia Z, Grandi F, & Bouaziz R (2014). τ OWL: 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 2014, pp 33-41.
- Zekri A, Brahmia Z, Grandi F, & Bouaziz R (2015a). Temporal Schema Versioning in τ OWL. In *Proceedings of the 2nd International Conference on Knowledge Management, Information and Knowledge Systems (KMIKS 2015)*, Hammamet, Tunisia, 16-18 April 2015, pp. 81-92.
- Zekri A, Brahmia Z, Grandi F, & Bouaziz R (2015b). τ OWL-Manager: A Tool for Managing Temporal Semantic Web Documents in the τ OWL Framework. In *Proceedings of the 9th International Conference on Advances in Semantic Processing (SEMAPRO 2015)*, Nice, France, 19-24 July 2015, pp. 56-64.
- Zekri A, Brahmia Z, Grandi F, & Bouaziz R (2016). τ OWL: A Systematic Approach to Temporal Versioning of Semantic Web Ontologies. *Journal on Data Semantics*, vol. 5, no. 3, 2016, pp. 141-163.
- Zekri A, Brahmia Z, Grandi F, & Bouaziz R (2017). Temporal Schema Versioning in τ OWL: A Systematic Approach for the Management of Time-varying Knowledge. *Journal of Decision Systems*, vol. 26, no. 2, 2017, pp. 113-137.
- Zhang, F., Ma, Z. M., & Li, W. (2015). Storing OWL ontologies in object-oriented databases. *Knowledge-*

Based Systems, 76, 240-255.