International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 13 (2021) pp. 080-090

Received: 15 Jan 2021; Accepted: 21 May 2021; Published: 13 July 2021

Towards a Semantic Querying Approach For a Multi-Version OWL 2 DL Ontology

Leila Bayoudhi[1], Najla Sassi[2] and Wassim Jaziri[3]

[1] MIRACL Laboratory, University of Sfax, Sfax, Tunisia,
bayoudhi.leila@gmail.com

[2] CCSE, Taibah University, Medina, Saudi Arabia,
MIRACL Laboratory, University of Sfax, Sfax, Tunisia
sassinajla@yahoo.fr

[3] CCSE, Taibah University, Medina, Saudi Arabia,
MIRACL Laboratory, University of Sfax, Sfax, Tunisia
jaziri.wassim@gmail.com

***Abstract*: The ontology formalizes a given domain of interest along a specific structure for disambiguating and conveying the knowledge semantics. This helps to create an explicit shared consensus on a given domain knowledge. When updating a domain ontology, it is useful to store ontology versions. Indeed, this helps to retrieve old critical knowledge through the different query types which require retrieving knowledge from different versions. Nonetheless, not all versioning approaches support conventional queries for OWL 2 DL ontologies such as cross-version queries. More importantly, they overlook semantic queries which expect retrieving complete answers with implicit knowledge. Our paper aims to present our proposed approach to support ontology engineers in querying an OWL 2 DL ontology and its evolution history.**

***Keywords*:** OWL 2 DL Ontology, Semantics, Cross-version, Querying, Versioning.

I. Introduction

In the literature, the ontology has been defined along various definitions. Nonetheless, all researchers concur that it is undoubtedly the chief specification used for modelling consensual domain knowledge, and fixing its heterogeneity and its semantics ambiguities in a formal way. The formal character of the ontology stems from an ontology language which turns the modelled knowledge into a machine understandable one. The ontology formalizes a given domain of interest along a specific structure which consists of concepts, properties, instances, relations and axioms. Particularly, the latter two components are which merely responsible for the intended use of ontology, i.e. disambiguating and conveying the knowledge semantics. This helps to create an explicit shared consensus on a given domain knowledge.

Like everything in the world, the ontology undergoes changes over time. According to Klein [1], an ontology change stems from a change in the domain, a change in the conceptualization or a change in the specification. A change in the domain is a modification of the world that the ontology models. For example, an ontology may change in response to the merge of two university departments having distinct administrative structures [2]. A change in the conceptualization is a modification of the concepts, properties and relations used to model a domain of interest. It results from a new usage or from the change of the standpoint from which the world is seen. For example, a university ontology from the perspective of students is not as the same as that from the ministry perspective. Indeed, the university ontology from the latter viewpoint may include additional knowledge, such as teachers' degrees, their identifiers, their hiring date, etc. A change in the specification corresponds to a change in the language in which ontology formalizes a domain knowledge.

When updating a domain ontology, it is useful to store ontology versions [3][4][5]. Indeed, this helps to: (i) keep track of ontology changes, (ii) retrieve old critical knowledge, (iii) recover previous domain states, and (iv) study the evolution history through the different query types which require retrieving knowledge from different versions, etc. Specifically, the state-of-the-art storage strategies of ontology versions perform well some typical queries at the cost of a high storage space [6]. More importantly, strategies with low space overhead lose the main specificity of the ontology: the knowledge semantics modelling. Hence, these strategies become inconvenient for the queries which require retrieving not only explicitly but also implicitly stated knowledge.

The major purpose of this paper is to propose an approach which supports ontology engineers in querying the ontology and its evolution history, based on the proposed storage strategy in [7].

The remaining of this paper is structured as follows. Section II describes the ontology evolution cycle. Section III outlines and studies the different versioning approaches for storing and querying ontology versions. Section IV summarizes the main conclusions drawn from Section III. Section V outlines the versioned queries considered in our work and describes the proposed query process. Before concluding, Section VI describes the architecture of the proposed query system.

II. Background

In the literature, several frameworks have been proposed to cope with both the ontology evolution and versioning. Each of these frameworks has its own step-based process which focuses mainly on a specific aspect, e.g. change identification, change effects, versions comparison, etc. To this end, Zablith et al. [8] have identified the commonalities among these frameworks and proposed a unified one. The proposed framework is a five-step cyclic process in such a way that each of its steps has been gaining the attraction of researchers (see Figure 1). In what follows, we outline the different identified stages and refer the reader to the evolution process-centric survey of Zablith et al. [8] for more details.

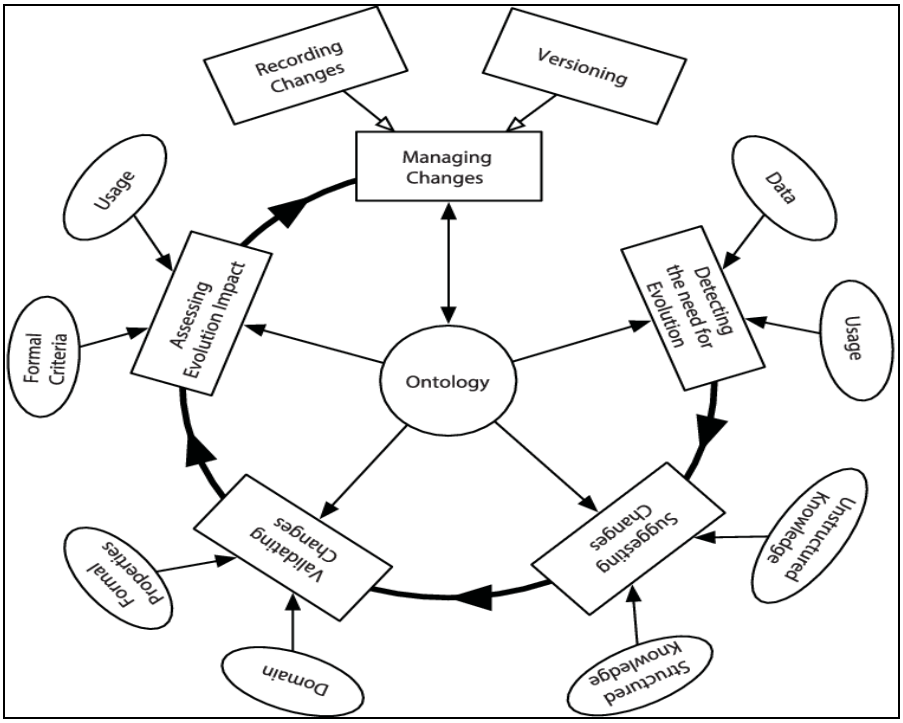

Figure 1. The ontology evolution cycle (adopted from [8]).

Validating changes: According to Zablith et al. [8], this step has as a purpose validating the suggested changes with regard to specific domain and formal properties. The domain-based validation allows determining the relevance of the changes to the domain. The second kind of validation ensures the satisfaction of the suggested changes to the DL-related properties, such as consistency, coherence, or to some custom validity rules.

Assessing evolution impact: It is referred to as "change propagation" in the evolution process proposed by Stojanovic [9]. This step intends to assess the change impacts on the artifacts which depend on the ontology in question (e.g. instances, ontologies, applications, etc). This is conducted using formal or usage-based criteria. The formal criteria allow evaluating a quantifiable cost of the change impact. The usage-based criteria allow checking whether a change invalidates ontology entities (e.g. instances) or other systems (e.g. applications used to answer queries that depend on the changed ontology).

Managing changes: This phase consists in managing changes by recording ontology changes and versions. This step aims at keeping track of both changes and versions, to retrieve old critical knowledge and to recover previous versions.

Our previous work [12][13] studied the "validating changes" stage, whereas the present work aims at finding solutions mainly to the "managing changes" stage.

Detecting the need for evolution: It is referred to as "change capturing" in the evolution process proposed by Stojanovic [9]. This phase aims at identifying the change to be applied, either from explicit requirements or from implicit ones. The explicit requirements are dictated by either ontology engineers or by ontology users. The implicit requirements stem from the change discovery methods which include three changes' types, namely structure-driven (e.g. deleting a concept without properties), usage-driven (e.g. deleting concepts that were never queried), and data-driven (e.g. deleting a concept without instances).

Suggesting changes: It corresponds to the "change representation" phase in the evolution process proposed by Stojanivic [9], and to the "relation discovery" phase in that proposed by Zablith [10]. This step allows expressing and formalizing a change in an explicit way which heavily depends on the ontology language. A research direction towards automating this step consists in suggesting appropriate changes based on external knowledge sources which may be structured (e.g. online ontologies [10], etc) or unstructured (e.g. text documents [11], etc).

III. Storage and Querying Approaches

Ontology versioning is a key solution to store ontology versions and to recover previous states of the domain [14]. In the literature, four main strategies can be distinguished to store ontology versions [6]:

Independent Copies (IC): It is also referred to as state-based versioning, version-based strategy or full materialization [3]. This strategy materializes complete ontology versions. If an ontology change is introduced, a new version having a new version IRI is created and stored next to previous ones.

Change-Based (CB): It is also referred to as edit-based, diff-based, delta-based strategy or operation-based versioning. It just computes and stores the differences (i.e. deltas) between two consecutive versions next to the root or the current ontology version.

Timestamp-Based (TB): It stores ontology components while assigning a temporal validity for each of them.

Hybrid (HB): It combines two or more of the three aforementioned strategies to keep track of the ontology evolution history.

It is of crucial interest to store ontology versions along one of the aforementioned strategies. Indeed, this helps to access previous versions and to satisfy various retrieval needs. Based on both a query type (i.e. materialization or structured) and its focus (i.e. version or delta), Fernandez et al. [6] identified the following retrieval needs (see Table 1):

Version materialization: This query type reflects a world state at a given time. It is about retrieving a certain version V_i which is valid at a given time t_i.

Single-version and cross-version structured queries: These queries require the retrieval of knowledge from a given version V_i or across several versions, respectively.

Delta materialisation: The result of this query type reflects the changes performed between two given versions which are not necessarily consecutive.

Single-delta and cross-delta structured queries: They are structured queries that are satisfied on a given delta or across several deltas, respectively. In the latter case, they are mainly used to study the knowledge evolution over time.

Type / Focus	Materialization	Structured queries	
		Single time	**Cross time**
Version	Version materialisation e.g. *get snapshot at time t_i.*	Single-version structured queries e.g. *lectures given by certain teacher at time t_i.*	Cross-version structured queries e.g. *subjects who have played the role of student and teacher of the same course.*
Delta	Delta materialisation e.g. *get delta at time t_i.*	Single-delta structured queries e.g. *students leaving a course between two consecutive snapshots, i.e. between t_{i-1} and t_i.*	Cross-delta structured queries e.g. *largest variation of students in the history of the archive.*

Table 1. A categorization of queries (adopted from [6]).

The remainder of the present section is devoted to present and discuss related work to the Independent Copies (IC), the Timestamp-Based (TB), the Change-Based (CB) and Hybrid strategies (HB). It also aims to scrutinize the adopted querying approaches to answer the aforementioned queries.

A. Independent copies approaches

This subsection sheds light on the most important state-of-the-art work which adopted the Independent Copies (IC) strategy.

Heflin and Hendler [15] proposed the ontology language SHOE (Simple HTML Ontology Extensions language). It allows managing multiple ontology versions by introducing some useful tags, such as the "use-ontology" and the "backward compatible-with" tags.

Klein et al. [16] coped with both ontology versions identification and relationship issues. To address the identification issue, the authors distinguished logical changes from non-logical ones. They also distinguished backward compatible revisions from non-backward compatible ones. For the ontology versions relationship issue, the authors proposed to define a set of transformations between two arbitrary ontology versions in terms of change operations. They also proposed to define a set of conceptual relationships between the components of two ontology versions. These functionalities were implemented by the Ontoview tool.

Völkel and Groza [17] proposed an approach which is based on the separation between the management aspects and the versioning functionality. Regarding the versioning functionality, the authors coped with structural and semantic deltas for ontology versions in RDF-based languages. In addition, they used the blank node enrichment technique for versioning RDF blank nodes. These functionalities were implemented by the SemVersion tool.

Sassi et al. [18] defined four criteria to assess the relevance of ontology versions, namely conceptualization, usage frequency, abstraction, and completeness. Once a predefined maximal number of versions is reached, a graph of relevance is proposed to show the ontology versions relationship after relevance scores computing. According to this graph, an optimization process is carried out to remove the least relevant version.

B. Change-based approaches

The present subsection exposes the literature work which adopted the change-based strategy for storing ontology versions.

Cassidy and Ballantine [19] proposed to store revisions as a sequence of patches (i.e. deltas) for RDF data. Each patch is made up of two sub-graphs, to represent both the added and the deleted triples. In addition, contextual information about revisions is stored. The proposed system is inspired by the theory of patches implemented in Darcs [1] which is a

[1] https://hub.darcs.net/darcs/darcs-reviewed

distributed version control system devoted for versioning software source codes.

Dragoni and Ghidini [20] proposed to inject transformation patterns into an OWL ontology, to keep track of changes. Moreover, these patterns promote the queries mapping among all ontology versions. This approach is positioned in the context of the evaluation of ontology changes on the effectiveness of information retrieval systems.

Im et al. [21] proposed a version management framework for RDF triple stores. It is based on storing intermediate deltas and a snapshot of the current version. To overcome the query time overhead for reconstructing a given version, the authors proposed the aggregated delta approach that is based on a compression algorithm. This solution eliminates duplicate RDF triples that are issued by both the sequential delta and all snapshots storage policies. The aggregated delta strategy performs well cross-delta queries at the cost of a higher storage space if it is compared with the sequential deltas strategy. The proposed approach has just considered asserted triples and not inferred ones while storing RDF deltas.

Kondylakis and Plexousakis [22] coped with query answering in evolving ontology-based integration systems. In such environments, the authors proposed a solution based on rewriting queries among ontology versions rather than redefining mappings between each ontology version and the corresponding data source. To this end, they used a high-level language of changes between the different ontology versions. The proposed approach is implemented by the web-based platform exelixis.

Graube et al. [23] proposed a semantic version control system called R43ples (Revision for triples). It consists in storing a full copy of a named graph and a Revision Management Ontology (RMO) in a triple store. Particularly, the RMO extends the PROV-O ontology [24], by including the added and the deleted triples and other additional tags characterizing revisions. To query such an ontology, the authors extended the SPARQL language using new key words, such as REVISION, BRANCH, and TAG. Nevertheless, the authors acknowledged that their solution is efficient for only medium-sized datasets.

Frommhold et al. [25] conceived a version control system for RDF datasets and blank nodes. After issuing a SPARQL update query, a patch (i.e. delta) is created while including the added and the deleted triples, and the change provenance information, as well. To do so, they used an RDF versioning vocabulary based on both the Delta ontology [26] and the PROV-O ontology [24]. They are both stored in a triple store. The authors have also implemented the proposed system.

C. Timestamp-based approaches

To keep track of the ontology evolution history, most of versioning approaches have adopted the timestamp-based strategy. They are as follows:

Eder et al. [27] used a directed graph to formalize ontology versioning. This graph represents all versions of ontology entities. This is mainly carried out by assigning valid time intervals to each node or edge in this graph. As far as the implementation for OWL ontologies is concerned, they proposed three techniques, namely a meta-ontology, a standard extension, or simply a standard use. However, each of these solutions has its limitations since it does not include a temporal semantics that can be supported by existing reasoning algorithms and other applications.

Bedi et al. [28] proposed a temporal tag-based technique for versioning OWL ontologies. They augmented every "rdf:Resource" and "rdf:Id" statement of an OWL document with new tags, i.e. "rdf:Validity" and "rdf:Timestamp" which have to be checked and updated with every ontology change. The authors acknowledged the limits of such an approach since it produces a crowded document.

In [29], the authors aim at assisting ontology engineers in understanding and detecting ontology changes. To this end, they proposed the Change Definition Language (CDL) and the use of a version log. The CDL is an OWL-based language that allows users to represent and to understand the meaning of changes. The version log helps to keep track of all concepts versions and to detect changes. Two protégé plugins were developed: the "Version Log Generator" and the "change detection plugin".

Chen and Mathews [30] proposed an evolutionary log which tracks the lifeline of an axiom/annotation in an evolving ontology. An evolutionary log is made up of axioms logs each of which characterizes an axiom/annotation by an anode. The latter represents some metadata, such as the author, the group to which the author belongs, and timestamps of creation and retirement. Regarding the implementation, the authors proved that both the document-centric and the rich axiom annotation-based approach are not relevant to the implementation of the proposed framework. Therefore, they suggested the temporal database, as a potential storage technique, for its scalability and its efficiency.

Klarman et al. [31] coped with changing definitional concepts over time in legal domains. To this end, the authors proposed a versioning strategy which consists in a DL-based representation. It has a three-layer structure consisting of a temporal framework, stamps and temporal restrictions. It allows modelling and switching the different versions of a concept definition within a single OWL ontology. The description logic character of the representation promotes a reasoner support. Nevertheless, an inference system may have low performance when switching between the versions of concepts' definitions.

In [32], The τ–SPARQL language was proposed as a temporal extension to SPARQL to support both time-point and temporal queries on RDF data. Two versioning strategies were used to evaluate such queries namely versioned snapshots and time-stamped RDF data.

Grandi and Scalas [33] proposed "The Valid Ontology" approach to cope with the temporal versioning issue of OWL/XML ontologies. Specifically, this ontology intends to manage the versions of both classes and property definitions. The approach consists mainly in augmenting an OWL/XML ontology document with custom XML markups.

Kirsten et al. [34] proposed a timestamp-based approach for the efficient storage and management of large biomedical ontology versions. Such an approach reduces redundancy by just storing the changed entities. Regarding the implementation, the authors used a MYSQL database repository for storing the different versions of ontology concepts, attributes, and relationships. They stated that their approach is efficient in terms of storage space and query

performance compared with the native approach where ontology versions are entirely stored.

In [35], the authors have proposed SPARQL-ST as a SPARQL extension for executing semantic queries on RDF datasets containing both spatial and temporal data.

Grandi [36] proposed a set of primitive changes that can be applied to an RDFS ontology. To cope with temporal versioning, he augmented RDFS ontology triples with temporal timestamps, based on the temporal data model proposed in [37]. To support temporal queries, the author proposed a SPARQL extension language called T-SPARQL [38] which reuses some temporal constructs of the temporal query language TSQL2. Nevertheless, the author did not provide any implementation to execute the formulated queries.

Liu [39] stored ontology versions in a relational database. The motivation behind this choice is avoiding redundancy and detecting changes between ontology versions. However, the proposed approach does not support all OWL constructs.

Bereta et al. [40] proposed the stSPARQL language for querying valid times of linked geospatial data which change over time.

Grandi [41] proposed a storage schema for multi-version directed graph-shaped ontologies. The schema consists in a temporal relationship, i.e. TreeRelation(Id, Pre, Post, Lev, From, To) which is stored in a temporal relational database. Such a storage schema allows executing temporal and personalized queries. In addition, he proposed a list of maintenance operations. Nonetheless, only ontologies with a class hierarchy are considered.

For managing OWL 2 ontology versions, Zekri [42] proposed the τOWL (Temporal OWL 2) framework which is based on different documents. A conventional schema document stores an OWL 2 ontology in the RDF/XML syntax. A conventional ontology instance document is an RDF document. A temporal schema document is a conventional schema document which is augmented by physical and logical annotations. These annotations allow the specification of which and how ontology components evolve over time. A temporal document ties the different versions of the other documents and specifies their relationships. Whenever a change is applied, a new time-stamped instance document version is generated. The τOWL-Manager prototype was developed to implement such an approach.

In [43], the authors constructed the Historical Knowledge Graph (HKG) to store ontology versions. This graph is made up of set of vertex V and a set of edges E. V consists of concepts, their validity periods and attributes. E consists of both hierarchical and evolutionary relationships. HKG is applied in three use-cases to support two tasks, namely information retrieval and maintenance of semantic annotations. To assess the validity of HKG, the authors used four medical datasets.

D. Hybrid approaches

So far, there have been few approaches which adopted the hybrid strategy, to store ontology versions. They are as follows:

Vander Sande et al. [44] proposed a distributed triple version control system called R&Wbase (Read-and-Write base). It is based on a hybrid storage strategy which combines both the TB and the CB ones. It consists in storing triples annotated by a context value indicating the version and the change type, i.e. addition or deletion.

Meinhardt et al. [45] developed the platform TailR, for storing and accessing the evolution history of linked data resources. Particularly, the underlying storage approach combines both the IC and the CB strategies. Indeed, the proposed storage system distinguishes three types of change sets, namely snapshot, delta, and delete. To decide on storing each of them, a set of rules was defined. As far as the implementation is concerned, two main HTTP APIs have been used: Push API and a read-only Memento API. The former is devoted to submitting and storing revisions, whereas the latter is devoted to accessing information about revisions on the linked data resources. A relational database management system was used to store the linked datasets and revisions information. The platform was implemented as a Python web service. Nevertheless, the authors acknowledged that their platform does not support cross-version queries.

Meimaris [4] focused mainly on proposing a new query language as an extension of SPARQL called the DIACHRON Query Language. It is used to query data, metadata and changes over a multi-version RDF dataset. Regarding the adopted storage strategy, it consists in storing dataset versions and deltas. This incurs a time overhead while computing and storing deltas, and a space overhead while storing complete snapshots. Furthermore, the author used the reification technique [46] to timestamp ontology entities and axioms, which slows down the query process [3].

Taelman et al. [47] proposed a hybrid storage approach for RDF versioned datasets, by combining the different state-of-the-art storage strategies to ensure querying efficiency while gaining storage space. Specifically, their storage strategy lies in keeping the first version and a delta chain. To fix the redundancies caused by the inherited changes in the strategy of Im et al. [21], Taelman et al. [47] proposed to compress these redundancies using the TB techniques. However, they rather annotated triples using addition and deletion flags [44] and stored them in B+trees-based indexes. Furthermore, a local change flag is added to discern the changes that have been performed with regard to the stored snapshot. To ensure a more efficient execution of versioned queries, the authors pre-processed and stored additional metadata during the ingestion step (i.e. the step of encoding and storage of triples). Offset and limits-enabling algorithms were also implemented to execute three types of queries: Version Materialization (VM), Version Query (VQ), and Delta Materialization (DQ) to retrieve triples at, across, and between different dataset versions, respectively. As a proof of concept, the authors implemented their approach in the OSTRICH (Offset enabled STore for TRIple CHangesets) tool. Taelman et al. evaluated their tool, its underlying storage strategy and its querying algorithms using the BEAR benchmark [48]. The results of this evaluation showed that their tool introduces a new trade-off between three dimensions: storage space, ingestion time and querying efficiency.

In both [7] and [49], the authors proposed hybrid storage strategy. It combines both the IC (i.e. Independent Copies) and the TB (i.e. Timestamp–based) strategies. The IC strategy

is adopted for the permanent storage (respectively, temporal storage) of a full copy of the root domain ontology version (respectively, of the current ontology version). It is also adopted for storing the reference ontology version. Regarding the TB strategy, it is used for storing and retrieving the evolution history. The Change Management tool for OWL 2 DL Ontologies "CAMO" was also developed to implement the proposed storage strategy.

IV. Discussion and Synthesis

In the light of the above discussions and according to Table 2, it is obviously noticeable that all the aforementioned basic strategies (i.e. IC, TB and CB) provide a trade-off between storage space efficiency and versioned queries processing overhead [3]. Particularly, the TB strategy can answer conventional versioned queries with low or medium complexities. For the sake of a better compromise between these two dimensions (i.e. space and query overheads), Stefanidis et al. [3] recommended hybrid approaches over pure storage ones. In the literature, few attempts have been proposed in this respect [45][4][47][49].

Another drawn conclusion from Table 3 is that not all versioning approaches support conventional queries such as cross-version queries (e.g. [45]). More importantly, they overlook semantic queries which expect retrieving complete answers with implicit knowledge. This is explained by the use of the implementation techniques (e.g. XML, database) which hinder the support of inference functionalities. Some works support these queries, but they are either limited to lightweight ontologies, such as RDFS ontologies [36] or RDF data [47]. We rather consider ontologies in a standard and expressive language, i.e. OWL 2 DL [50].

It is also worth mentioning that not all works provide a tool as a proof of concept of their approaches [51][52], or the proposed tool does not manage ontology versions [42][47]. Thus, we thought about developing a tool which implements the proposed querying approach.

In our work, we are interested in two main inter-dependent issues which are involved in the evolution history management: the versions storage and the support of conventional versioned queries. Indeed, we adopted a hybrid storage strategy which gains space and fosters the efficient execution of versioned queries which are not considered in related work. The proposed storage strategy has already been detailed in both [7] and [49]. The present paper aims to focus on the querying aspect of our versioning approach.

Strategies / Retrieval need	Independent Copies (IC)	Change-Based (CB)	Timestamp-based (TB)
Version materialization	Low	Medium	Medium
Delta materialization	Medium	Low	Low
Single-version structured queries	Medium	Medium	Medium
Cross-version structured queries	High	High	Medium
Single-delta structured queries	High	Medium	Medium
Cross-delta structured queries	High	High	Medium

Table 2. Complexity level of versioned queries (adopted from [53]).

V. Our Querying Approach

The proposed storage strategy in [7] is devoted to trace OWL ontology versions and to retrieve pertinent information. The present section aims to emphasize the utility of this strategy by outlining the versioned queries considered in our work (see Section A). Thereafter, Section B describes the proposed query process to answer the different versioned query types.

A. Versioned queries

In the same line as Papakonstantinou et al. [54] and Fernandez et al. [6], versioned queries are categorized along the type and focus. According to the first dimension, three main query categories are distinguished: Materialization, Single-version and Cross-version queries. Regarding the second dimension, the focus of a given query may be either a delta between two ontology versions or an ontology version which may be the current (i.e. modern) or any anterior version (i.e. historical). These query types have already been defined in Section III.

Figure 2 is an extended version of that of Papakonstantinou et al. [54]. It shows how so various s query types can be distinguished based on one type and focus. In the literature, structured queries usually refer to issuing interrogations on a given delta, version, or on more than one version, without considering inferred knowledge (i.e. the grey-colored query boxes in Figure 2). This has motivated us to extend the conventional versioned query types by semantic queries (i.e. the green-colored boxes in Figure 2).

Semantic queries are interrogations about both asserted and entailed knowledge. They are mainly enabled using semantic inference engines (i.e. reasoners). In the present work, beyond the conventional versioned query types, we also consider semantic queries that are modern single-version, historical and cross-version queries.

- Modern single-version semantic queries: They are semantic queries that allow retrieving knowledge from the current ontology version.
- Historical single-version semantic queries: They are semantic queries that allow retrieving knowledge from an anterior ontology version. In the present work, only the root ontology version may be targeted for these queries.
- Cross–version semantic queries: They are semantic queries that expect an answer from more than an ontology version. In the present work, these queries are processed by considering all ontology versions. Nevertheless, we intend to propose a new query language, for specifying only some ontology versions.

We note that these query types target the current, root and reference ontology versions, respectively. Other semantic query types (e.g. single-delta semantic queries, cross-delta semantic queries) can be defined by developing an inference layer on top of the database to take into consideration inferred knowledge. For more details about our developed hybrid storage strategy, we refer the reader to our previous work [7].

Work	Ontology language	Storage strategy	Physical storage technology	Querying functionality	Tool
[15]	SHOE	IC	OWL files	-	-
[16]	DAML+OIL	IC	OWL files	Querying changes.	+
[27]	Generic	TB	OWL files	-	-
[17]	RDFS	IC	Triple store	Structural and semantic diffs.	+
[29]	OWL DL	TB	OWL files	Querying a version log to detect changes.	+
[28]	OWL	TB	OWL files	-	-
[19]	RDF	CB	Relational database and triple store	-	-
[34]	Generic	TB	Relational database	Non-semantic queries on the ontology structure.	-
[55]	OWL DL	TB	OWL files	-	+
[36]	RDFS	TB	Triple store	A temporal SPARQL extenstion: T-SPARQL.	-
[39]	OWL Lite	TB	Relational database	Detecting changes using SQL queries.	-
[21]	RDF	CB	Relational database	Different query types, except semantic ones, are formulated using SQL.	-
[20]	OWL	CB	OWL file	Queries mapping among ontology versions	-
[22]	RDFS	CB	Triple store	Queries rewriting and supporting cross-version structured queries.	+
[44]	RDF	HB :TB+CB	quad-store	SPARQL queries to reconstruct versions.	-
	RDF	CB	triple store	Extending SPARQL using new keywords.	+
[51]	OWL DL	CB	OWL files	-	-
[45]	RDF	HB: IC+CB	Relational database	Basic queries on resources history, but no focus on cross-version queries.	+
[18]	UML	IC	Relational database	-	+
[25]	RDF	CB	Triple store	Retrieving patches via the LUCID endpoint using predefined parameters.	+
[56]	OWL DL	TB	OWL files	-	+
[41]	Generic	TB	Relational database	Personalized queries using SQL.	-
[42]	OWL 2	TB	RDF and XML files	-	+
[4]	OWL DL	HB:IC+CB	Triple store	DIACHRON QL, dataset and version listing, data queries, longitudinal queries, queries on changes, mixed queries on changes and data.	+
[47]	RDF	HB:TB+CB	HDT files	Versioned queries: VM, DM, and VQ.	+
[43]	OWL	TB	OWL file	Querying medical documents annotated with different datasets versions.	-

Table 3. A comparison of versioning approaches.

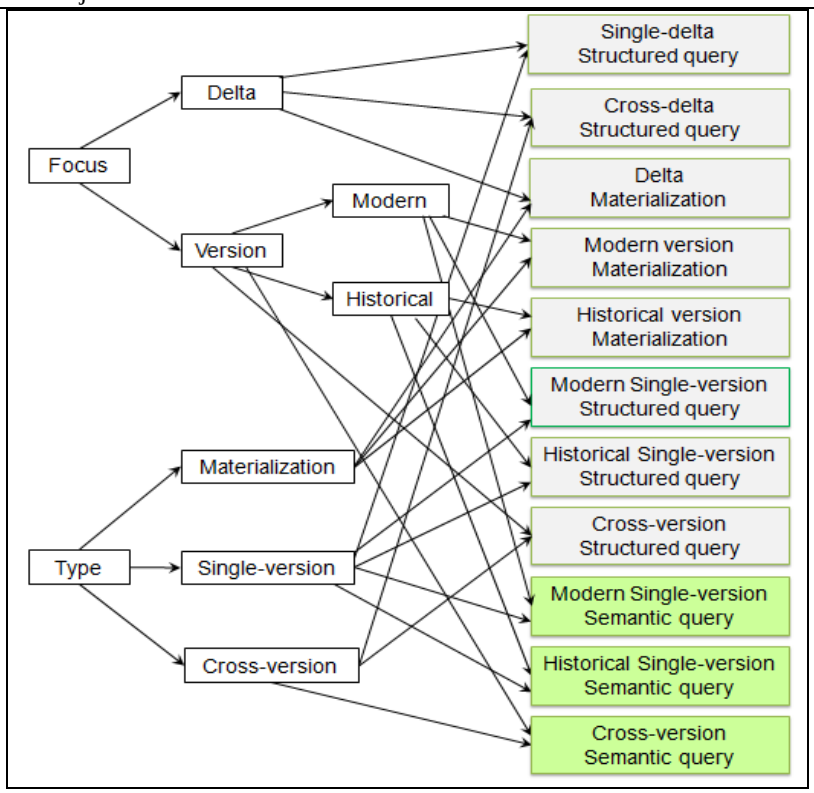

Figure 2. The considered versioned queries.

B. Versioned queries answering process

In the present work, we adopt a three-step query process which is depicted by Figure 3.

i. First, a user formulates a versioned query among those listed in Figure 2.
   In this work, we rely mainly on graphic user interfaces to express the predefined versioned queries. A future research is directed towards proposing a new string-based query language to explicitly formulate a versioned query.
ii. Second, the issued query is analysed to determine its type (i.e. materialization, single-version, or cross-version), its focus (i.e. delta or version).
   In this work, a user is guided by formulating queries in convenient and dedicated interfaces to each query type. Nevertheless, our reflection is directed towards proposing a query parser to determine the query type and focus.
iii. Third, the query is executed and its answer is delivered to the user.
   In this work, the query execution is delegated to the "state-of-the-art query APIs" (see Section IV). Nonetheless, we envisage proposing query plans and optimizers to carry out an efficient answering.

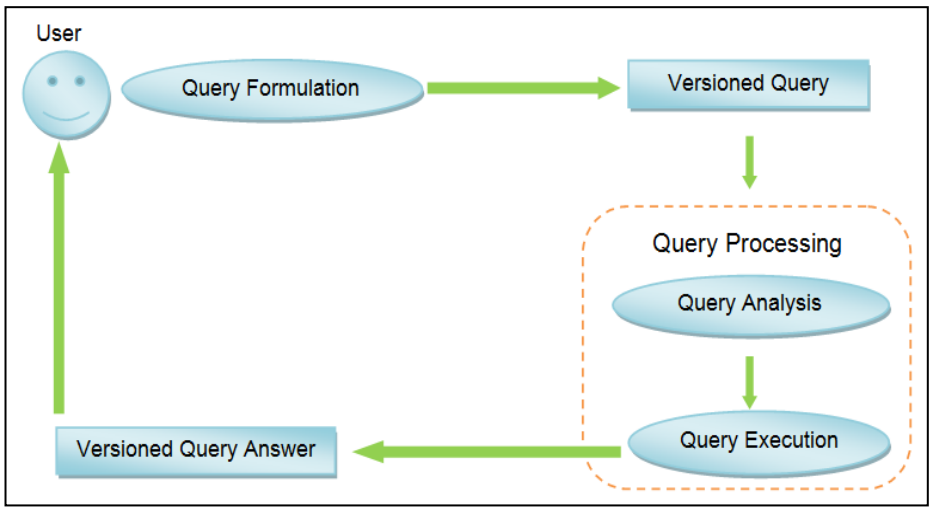

Figure 3. The adopted query process to answer a versioned query.

VI. Implementation

The architecture of the proposed query system is more detailed in Figure 4. A semantic query is formulated either in the DL query tab or in the Snap-SPARQL [57] plugin of the protégé ontology editor [58]. A structured or materialization query is formulated through dedicated developed user interfaces. This query is executed by the query systems, i.e. native query and query by example of the database management system DB4O [59].

DB4O (DataBase For Objects) is an open source and easy to use database management system for object-oriented databases. DB4O does not have a standard String-based query language such as SQL for RDBMSs. Nonetheless, it has three query systems: Query By Example (QBE), Native Query and SODA (Simple Object Data Access) Query API. Although SODA is a fast API for processing queries, the first two query options are the most recommended for developers [59]. Hence, they are used in the present work to query an ontology evolution history.

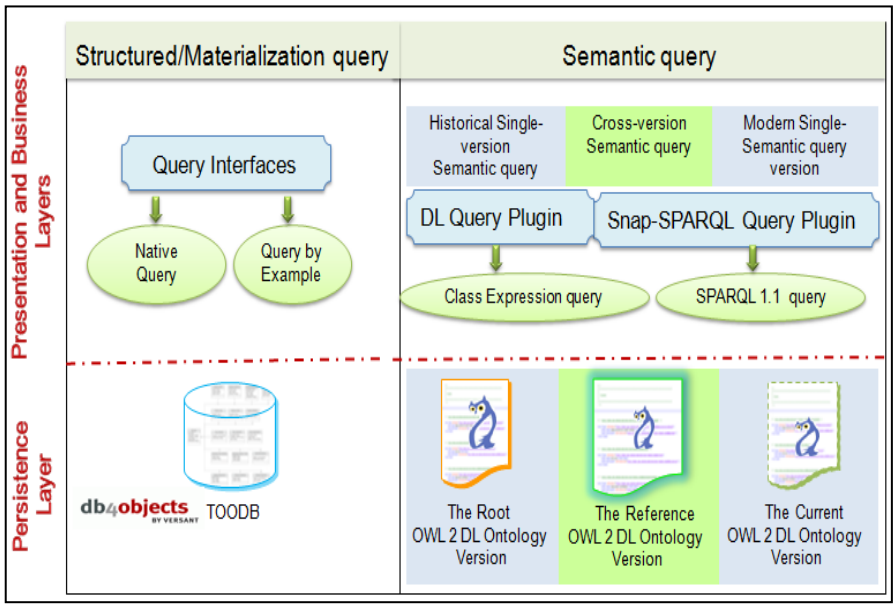

Figure 4. The query system architecture.

VII. Conclusion

An important related issue to ontology versions storage [7] is their access and the retrieval of old knowledge. In this respect, three main research streams can be distinguished:

i. Approaches which overlooked the querying issue [56][18][42]. They were rather limited to proposing a storage strategy to keep track of the ontology evolution history.
ii. Approaches which proposed specific storage strategies while using standard query languages, such as SPARQL for ontologies and SQL for relational databases [21][25][41] .
iii. Approaches that proposed new query languages as extensions to standard ones, to support various conventional versioned query types [36][4][35][40] .

Our work is in the same line as the second research stream. Indeed, the ontology evolution history is queried using a database management system, whereas cross-version and semantic queries are retrieved from a reference ontology using a reasoner or a semantics-enabling query language such as SPARQL 1.1 [60].

In this paper, we outlined some typical versioned queries (e.g. semantic and multi-version queries). A prototype tool was also developed to support knowledge engineers in retrieving the evolution history of an OWL 2 DL ontology.

In the near future, the query tool will be extended by other querying functionalities. We also plan to make it available online and to carry out a user study evaluation thereof. In the same scope, we intend to propose a new query language for answering multi-version queries.

References

[1] M. Klein, « Change Management for Distributed Ontologies », PhD Thesis, VRIJE University, Amesterdam, 2004.
[2] N. F. Noy et M. Klein, « Ontology Evolution: Not the Same as Schema Evolution », *Knowledge and Information Systems*, vol. 6, nº 4, p. 428-440, 2004, doi: 10.1007/s10115-003-0137-2.
[3] K. Stefanidis, I. Chrysakis, et G. Flouris, « On Designing Archiving Policies for Evolving RDF Datasets on the Web », in *Proceedings of the International Conference on Conceptual Modeling*, 2014, p. 43-56. doi: 10.1007/978-3-319-12206-9_4.
[4] M. Meimaris, « Managing, Querying and Analyzing Big Data on the Web », PhD Thesis, University of Thessaly, Greece, 2018.
[5] F. Zhang, Z. Li, D. Peng, et J. Cheng, « RDF for temporal data management – a survey », *Earth Sci Inform*, vol. 14, nº 2, p. 563-599, 2021, doi: 10.1007/s12145-021-00574-w.
[6] J. D. Fernández, J. Umbrich, A. Polleres, et M. Knuth, « Evaluating Query and Storage Strategies for RDF

Archives », *Semantic Web*, vol. 10, nº 2, p. 247-291, 2019, doi: http://dx.doi.org/10.3233/SW-180309.

[7] L. Bayoudhi, N. Sassi, et W. Jaziri, « A Hybrid Storage Strategy to Manage the Evolution of an OWL 2 DL Domain Ontology », in *Proceedings of the 21st International Conference KES-2017*, 2017, vol. 112, p. 574-583. doi: 10.1016/j.procs.2017.08.170.

[8] F. Zablith *et al.*, « Ontology evolution: a process-centric survey », *The Knowledge Engineering Review*, vol. 30, nº 01, p. 45-75, 2015, doi: 10.1017/S0269888913000349.

[9] L. Stojanovic, « Methods and tools for ontology evolution », PhD Thesis, University of Karlsruhe, Germany, 2004.

[10] F. Zablith, « Harvesting Online Ontologies for Ontology Evolution », PhD Thesis, The Open University, United Kingdom, 2011.

[11] Z. Sellami, « Gestion Dynamique d'Ontologies à partir de Textes par Systèmes Multi-agents Adaptatifs », PhD Thesis, University of Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), France, 2012.

[12] L. Bayoudhi, N. Sassi, et W. Jaziri, « Overview and reflexion on OWL 2 DL ontology consistency rules », in *Proceedings of the Second International Conference on Internet of things and Cloud Computing*, 2017, vol. Part F1348. doi: 10.1145/3018896.3036376.

[13] L. Bayoudhi, N. Sassi, et W. Jaziri, « How to Repair Inconsistency in OWL 2 DL Ontology Versions? », *Data and Knowledge Engineering*, vol. 116, p. 138-158, 2018, doi: 10.1016/j.datak.2018.05.010.

[14] L. Bayoudhi, N. Sassi, et W. Jaziri, « A Survey on Versioning Approaches and Tools », in *Intelligent Systems Design and Applications*, 2021, vol. 1351, p. 1155-1164. doi: 10.1007/978-3-030-71187-0_107.

[15] J. Heflin et J. Hendler, « Dynamic Ontologies on the Web », in *Proc. of the 17th Nat. Conf. on Artificial Intelligence*, 2000, p. 443-449.

[16] M. Klein, D. Fensel, A. Kiryakov, et D. Ognyanov, « Ontology versioning and change detection on the web », *Knowledge Engineering and Knowledge Management, Proceedings: Ontologies and the Semantic Web*, vol. 2473, p. 197-212, 2002.

[17] M. Völkel et T. Groza, « SemVersion: RDF-based ontology versioning system », in *Proceedings of the IADIS International Conference on WWW/Internet*, 2006, nº Section 6.

[18] N. Sassi, W. Jaziri, et S. Alharbi, « Supporting ontology adaptation and versioning based on a graph of relevance », *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 28, nº 6, p. 1035-1059, 2016, doi: 10.1080/0952813X.2015.1056239.

[19] S. Cassidy et J. Ballantine, « Version Control For RDF Triple Stores », in *ICSOFT (ISDM/EHST/DC)*, 2007, p. 5-12.

[20] M. Dragoni et C. Ghidini, « Evaluating the impact of ontology evolution patterns on the effectiveness of resources retrieval », 2012.

[21] D.-H. Im, S.-W. Lee, et H.-J. Kim, « a Version Management Framework for Rdf Triple Stores », *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, nº 01, p. 85-106, 2012, doi: 10.1142/S0218194012500040.

[22] H. Kondylakis et D. Plexousakis, « Ontology evolution without tears », *Journal of Web Semantics*, vol. 19, p. 42-58, 2013, doi: 10.1016/j.websem.2013.01.001.

[23] M. Graube, S. Hensel, et L. Urbas, « R43ples: Revisions for Triples An Approach for Version Control in the Semantic Web », in *Proceedings of the 1st Workshop on Linked Data Quality co-located with 10th International Conference on Semantic Systems (SEMANTiCS 2014)*, 2014, vol. 1215.

[24] World Wide Web Consortium, « PROV-O: The PROV Ontology », *W3C Recommendation*, 2013. https://www.w3.org/TR/prov-o/

[25] M. Frommhold, R. N. Piris, N. Arndt, S. Tramp, N. Petersen, et M. Martin, « Towards Versioning of Arbitrary RDF Data », in *Proceedings of the 12th International Conference on Semantic Systems (SEMANTICS 2016)*, 2016, p. 33-40. doi: https://10.1145/2993318.2993327.

[26] T. Berners-Lee et D. Connolly, « Delta: an ontology for the distribution of differences between RDF graphs ». 2004. [En ligne]. Disponible sur: https://www.w3.org/DesignIssues/lncs04/Diff.pdf

[27] J. Eder et C. Koncilia, « Modelling changes in ontologies », in *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops*, 2004, p. 662-673.

[28] P. Bedi et S. Marwaha, « Versioning OWL ontologies using temporal tags », *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 1, nº 3, p. 332-337, 2007.

[29] P. Plessers, O. De Troyer, et S. Casteleyn, « Understanding ontology evolution: A change detection approach », *Web Semantics*, vol. 5, nº 1, p. 39-49, 2007, doi: 10.1016/j.websem.2006.11.001.

[30] C. Chen et M. M. Matthews, « A New Approach to Managing the Evolution of OWL Ontologies », in *Proceedings of The 2008 International Conference on Semantic Web and Web Services*, 2008, nº D, p. 57-63.

[31] S. Klarman, R. Hoekstra, et M. Bron, « Versions and applicability of concept definitions in legal ontologies », in *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions*, 2008, vol. 496, nº April.

[32] J. Tappolet et A. Bernstein, « Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL », in *The Semantic Web: Research and Applications*, 2009, p. 308-322.

[33] F. Grandi et M. R. Scalas, « The Valid Ontology: A simple OWL temporal versioning framework », in *Proceedings of the 3rd International Conference on Advances in Semantic Processing - SEMAPRO 2009*, 2009, p. 98-102. doi: 10.1109/SEMAPRO.2009.12.

[34] T. Kirsten, M. Hartung, A. Groß, et E. Rahm, « Efficient Management of Biomedical Ontology Versions », in *Proceedings of the 4th International Workshop on Ontology Content (OnToContent)*, 2009, p. 574-583.

[35] M. Perry, P. Jain, et A. P. Sheth, « SPARQL-ST: Extending SPARQL to Support Spatiotemporal Queries », in *Geospatial Semantics and the Semantic Web: Foundations, Algorithms, and Applications*, vol. 12, N. Ashish et A. P. Sheth, Éd. Boston, MA: Springer US, 2011, p. 61-86. doi: 10.1007/978-1-4419-9446-2_3.

[36] F. Grandi, « Light-weight Ontology Versioning with Multi-temporal RDF Schema », in *Proceedings of The*

Fifth International Conference on Advances in Semantic Processing (SEMAPRO2011), Lisbon, Portugal, 2011, nº c, p. 42-48.

[37] F. Grandi, « Multi-temporal RDF Ontology Versioning », in *Proceedings of the 3rd International Workshop on Ontology Dynamics (IWOD@ISWC 2009)*, Washington DC, USA, 2009, vol. 519.

[38] F. Grandi, « T-SPARQL: A TSQL2-like temporal query language for RDF », in *Local Proceedings of the Fourteenth East-European Conference on Advances in Databases and Information Systems*, Novi Sad, Serbia, 2010, vol. 639, p. 21-30.

[39] K. Liu, « A Method Based on RDB for Detecting Changes Between Multi-version Ontologies », *Journal of Computational Information Systems*, vol. 8, nº 8, p. 3293-3300, 2012.

[40] K. Bereta, P. Smeros, et M. Koubarakis, « Representation and Querying of Valid Time of Triples in Linked Geospatial Data », in *The Semantic Web: Semantics and Big Data*, 2013, vol. 7882, p. 259-274.

[41] F. Grandi, « Dynamic class hierarchy management for multi-version ontology-based personalization », *Journal of Computer and System Sciences*, vol. 82, nº 1, p. 69-90, 2016, doi: 10.1016/j.jcss.2015.06.001.

[42] A. Zekri, « Intégration du Temps et du Versionnement des Schémas dans le Web Sémantique », PhD Thesis, University of Sfax, Tunisia, 2018.

[43] S. D. Cardoso, M. Da Silveira, et C. Pruski, « Construction and exploitation of an historical knowledge graph to deal with the evolution of ontologies », *Knowledge-Based Systems*, vol. 194, p. 105508, 2020, doi: 10.1016/j.knosys.2020.105508.

[44] M. Vander Sande, P. Colpaert, R. Verborgh, S. Coppens, E. Mannens, et R. Van de Walle, « R&Wbase: Git for triples », in *Proceedings of the 6th Workshop on Linked Data on the Web*, 2013, vol. 996.

[45] P. Meinhardt, M. Knuth, et H. Sack, « TailR: A Platform for Preserving History on the Web of Data », in *Proceedings of the 11th International Conference on Semantic Systems (SEMANTiCS 2015)*, 2015, p. 57-64. doi: https: //dx.doi.org/10.1145/2814864.2814875.

[46] C. Gutierrez, C. A. Hurtado, et A. Vaisman, « Introducing time into RDF », *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, nº 2, p. 207-218, 2007, doi: 10.1109/TKDE.2007.34.

[47] R. Taelman, M. V. Sande, J. Van Herwegen, E. Mannens, et R. Verborgh, « Triple Storage for Random-Access Versioned Querying of RDF Archives », *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 54, p. 4-28, 2019, doi: https://doi.org/10.1016/j.websem.2018.08.001.

[48] J. D. Fernández, A. Polleres, et J. D. Fern, « BEAR: Benchmarking the Efficiency of RDF Archiving Archiving », Vienna University of Economics and Business, Vienna, 2015.

[49] L. Bayoudhi, N. Sassi, et W. Jaziri, « Efficient management and storage of a multiversion OWL 2 DL domain ontology », *Expert Systems*, vol. 36, nº 2, p. e12355, 2019, doi: 10.1111/exsy.12355.

[50] B. Motik *et al.*, « OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (Second Edition) », *Online*, 2012. https://www.w3.org/TR/owl2-syntax/

[51] P. Pittet, C. Cruz, et C. Nicolle, « An ontology change management approach for facility management », *Computers in Industry*, vol. 65, nº 9, p. 1301-1315, 2014, doi: 10.1016/j.compind.2014.07.006.

[52] D. Zheleznyakov, E. Kharlamov, W. Nutt, et D. Calvanese, « On Expansion and Contraction of DL-Lite Knowledge Bases », *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 57, p. 100484, 2019, doi: https://doi.org/10.1016/j.websem.2018.12.002.

[53] J. D. Fernandez, A. Polleres, et J. Umbrich, « Towards efficient archiving of dynamic linked open data », in *Proceedings of the First DIACHRON Workshop on Managing the Evolution and Preservation of the Data Web co-located with 12th European Semantic Web Conference (ESWC 2015)*, 2015, vol. 1377, p. 34-49.

[54] V. Papakonstantinou, I. Fundulaki, et G. Flouris, « Deliverable 5.2.2: Second Version of the Versioning Benchmark », The Institute of Computer Science, FORTH, Greece, Technical Report, 2018.

[55] R. Jedidi, « Approche d'évolution d'ontologie guidée par des patrons de gestion de changement », PhD Thesis, University of Paris-Sud XI Orsay, France, 2009.

[56] F. Chamekh, « L'évolution du web de données basée sur un système multi-agents », PhD Thesis, University of Lyon, France, 2016.

[57] M. Horridge et M. Musen, « Snap-SPARQL: A Java Framework for working with SPARQL and OWL », in *Revised Selected Papers of the 12th International Experiences and Directions Workshop on Ontology Engineering*, 2015, vol. 9557, p. 154-165. doi: https://doi.org/10.1007/978-3-319-33245-1_16.

[58] M. A. Musen, « The protégé project », *AI Matters*, vol. 1, nº 4, p. 4-12, 2015, doi: 10.1145/2757001.2757003.

[59] J. Paterson, S. Edlich, H. Hörning, et R. Hörning, *The Definitive Guide to db4o*. Apress, 2006. [En ligne]. Disponible sur: https://link.springer.com/book/10.1007%2F978-1-4302-0176-2

[60] World Wide Web Consortium, « SPARQL 1.1 Overview », 2013. https://www.w3.org/TR/sparql11-overview/ (consulté le mai 14, 2019).

Author Biographies

Leila Bayoudhi is a member of the MIRACL laboratory (Multimedia, InfoRmation Systems and Advanced Computing Laboratory), the University of Sfax, Tunisia. She received her computer engineer degree in 2012 and her PhD degree in 2020 from the National Engineering School of Sfax, the University of Sfax. Her major research interests include ontology engineering and its related challenges.

Najla Sassi is an assistant professor in computer science at the College of Computer Science and Engineering, Taibah University, KSA. She received her master's degree in computer science from the University of Rouen, France in 2004 and a PhD degree from the University of Sfax in 2011. Her main interests are artificial intelligence, ontology, and information systems modelling. Dr Najla Sassi was involved in several projects and has published several research papers in international journals and conferences.

Wassim Jaziri received his PhD degree in Computer Science in 2004 from INSA - Rouen, France. He received an Accreditation to supervise research (French HDR, a required grade to be a full professor) in computer science in 2010 from Sfax University - Tunisia. Currently, He is a professor in computer science at the College of Computer Science and Engineering, Taibah University, KSA. His main interests are geographic information systems, spatio-temporal databases, spatial decision aid, data and knowledge modelling, ontology, and optimization.