

Managing Modular Ontology Evolution Under Big Data Integration

Hanen Abbas^(✉) and Faiez Gargouri

MIRACL Laboratory, Higher Institute of Computer Science and Multimedia,
Sfax University, Sfax, Tunisia

abbes.hanen@gmail.com, faiez.gargouri@isims.usf.tn

Abstract. Big Data integration frameworks provide unified view of the data available from heterogeneous data sources. These data sources are continuously evolving, forcing systems that integrate them to adapt their global schema after each change. This gets more challenging when aiming to maintain the global schema always reflecting data sources content. To cope with such complexity, in this paper we describe evolution scenarios and manage modular ontology evolution within Big Data integration framework in an a priori way according to changes performed against the data sources.

Keywords: Ontology evolution · Big Data integration · Data source evolution · Modular ontology

1 Introduction

According to [1], “Big Data can be defined as data that exceed the processing capacity of conventional database systems. This implies that the data count is too large, and/or data values change too fast, and/or it does not follow the rules of conventional database management systems”. Big Data are characterized along three important dimensions, namely volume, variety and velocity [2, 3] known as 3Vs.

Despite this complexity, users usually look for a unified view of the data available from heterogeneous data sources. Consequently, several Big Data integration systems were proposed. Nevertheless, they do not cope with the evolutionary aspect of Big Data sources. Indeed, maintaining an integrated view over such evolving and heterogeneous set of data sources is a challenging problem which current systems fail to address.

Regardless of the great amount of work done in ontology-based Big Data integration, an important problem that most of the systems are likely to ignore is that ontologies are living artifacts and are subject to change and evolution as well. Ontologies are frequently changed to reflect the new knowledge that is acquired. The problem that occurs is the following: when data sources change, the mappings may become invalid and should be updated. Ontology evolution is defined as the “timely adaptation of an ontology to the arisen changes and the consistent management of these changes” [4].

In this paper, we address the problem of ontology evolution under Big Data integration. We argue that data sources changes should be considered when designing ontology-based Big Data integration systems. A distinctive solution would be to update

the mappings and then regenerate the dependent ontologies each time a data source evolves. We propose an a priori method to correct incoherencies caused by a change and rely on ontology learning and ontology merging tools to manage the evolution process.

This paper is organized as follows. Section 2 exposes our research context. In the third section, we describe scenarios that drive to evolve the ontology in a Big Data integration context. Our approach to manage modular ontology evolution in Big Data integration is detailed in Sect. 4. We start by describing the evolution process, then we give examples of coherence constraints that we respect to evolve the ontology. Inspired by previous research work, we adapt existing notions to our needs and provide an illustrative example. Section 5 examines related work. A discussion comparing our research to previous ones encloses this section. Finally Sect. 6 draws conclusions and suggests further research.

2 Research Context

This work joins within the scope of a Big Data integration approach [5] such that the departure corpus is formed by Big Data, whereas the target schema is an OWL¹ (Ontology Web Language) ontology. We are interested particularly to OWL-DL since it supports the maximum expressiveness while retaining computational completeness and decidability.

The original approach aims to build an ontology for Big Data integration where Big Data are seen as data from many sources having different formats, each source contains a very big amount of data and grows and evolves independently from the other ones [5]. This approach is based on three main steps (Fig. 1):

- *Wrapping data sources to MongoDB² databases*: the content of each data source is converted to a MongoDB database,
- *Mapping MongoDB databases to ontology modules*: each MongoDB database is mapped to an OWL ontology module by means of transformation rules [6, 7]. The first phase is the creation of the ontology skeleton. It consists of defining ontology classes and detecting subsumption relationships between them. The second phase is to learn concepts properties (dataTypeProperties and objectProperties). Individuals are identified in the third phase. In the fourth phase, class axioms (equivalence and disjoining), property axioms (inverseOf) and constraints (cardinality constraints, value constraints) are deduced. Finally, in the fifth phase, the ontology is enriched with classes' definition operators (union, intersection, complement).
- *Merging ontology modules to get a global one*: the modules obtained in the previous step are merged together in order to get a global ontology [8]. Our algorithm is based on three main actions. The first action is to detect overlaps between the two modules to be merged. The second action is to compute similarities between concepts belonging to the two ontology modules and the third action is to update the reference ontology module with concepts, attributes, as well as relationships from

¹ <https://www.w3.org/TR/owl-features/>.

² <http://www.mongodb.org/>.

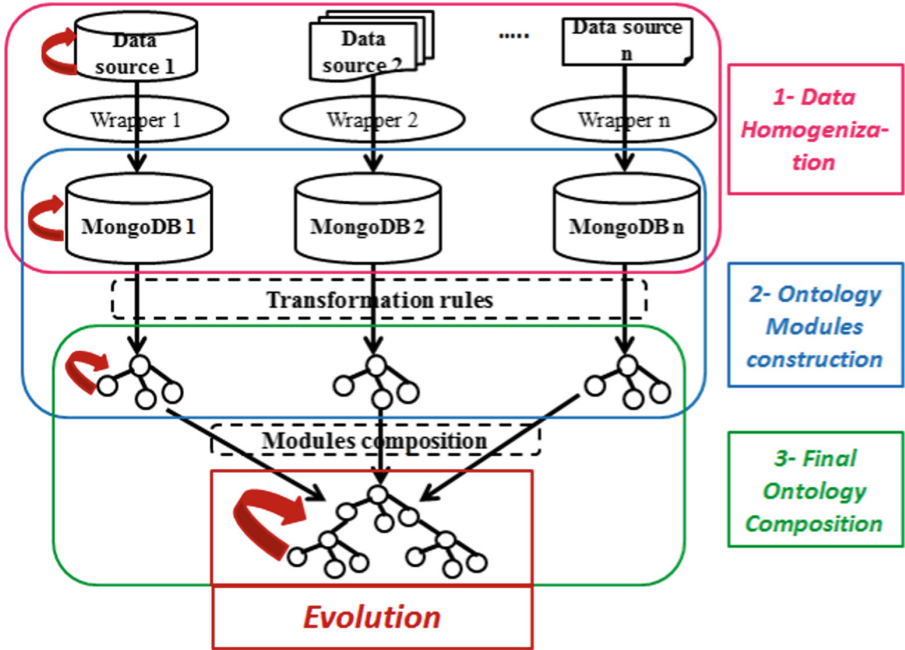


Fig. 1. Ontology-based Big Data integration.

the input ontology module. To measure the similarity between two concepts, one from the input ontology module and the other from the reference ontology module, we combine syntactic matching and semantic matching. The syntactic matching compares strings characterizing concept names as well as concept attributes whereas the semantic matching is based on relationships similarity. To compute the syntactic matching, we apply a distance function over a pair of strings. We adopted the Levenshtein distance [9] which returns the number of character changes needed to transform one string into another (LEV). The smaller this dissimilarity is (i.e. the less character changes needed), the more similar are the strings.

The methodology followed to build the global ontology follows a modular conceptualization since the beginning of the ontology development cycle where an ontology module represents a point of view covered by a data source containing data about the modeled domain.

Considering that Big Data are dynamic by nature, they are exposed to different updates. These updates must be sent up to the global ontology and evolution issues have to be dealt with. Indeed, new sources of data may appear and data about a domain may change according to different manners. On the one hand, new data may appear, and this leads to establish new concepts. On the other hand, some data may become obsolete, and so, some concepts must be removed from the global ontology. Besides, modeling a domain may necessitate concepts redefinition. New concepts that are more specific than the pre-existent ones can be defined if the domain needs to be more precise, or more abstract if we want to simplify the domain and facilitate its comprehension.

3 Evolution Triggers

Considering the increasing number of Big Data integration frameworks emerging nowadays with different features and objectives, evolutionary issues are critical for the contributors involved in the integration process. Conventional data modeling approaches occasionally consider the data model evolution issue. To fill this gap, our Big Data integration framework (c.f. Fig. 1) is based on three main fields of ontology engineering to manage our global data model, namely ontology learning, ontology merging and ontology evolution (Fig. 2).

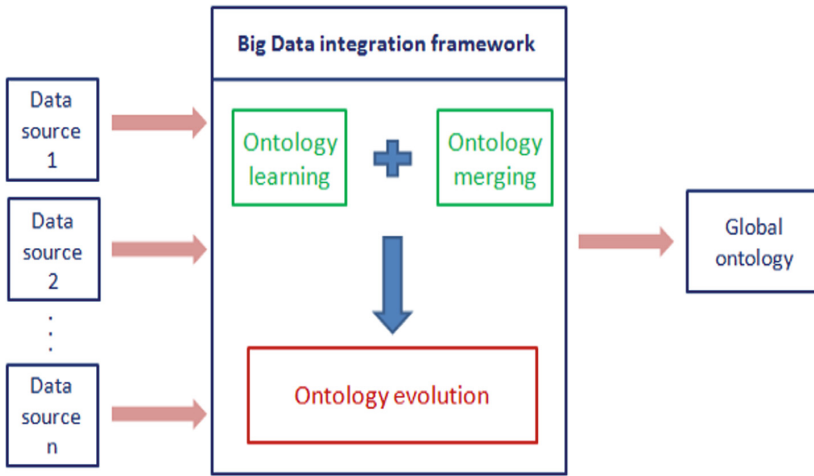


Fig. 2. Big Data integration evolution.

Accordingly, the derived data model represents a shared data model for the various data sources it integrates and should be always up to date in compliance with the changes met by the data sources. However, to deal with the evolution of data models over time, three scenarios may trigger the evolution process as depicted in Fig. 3.

We consider that ontology modules corresponding to each data source are stored separately in addition to the global ontology for subsequent use.

- *Scenario 1* (Introducing a new data source): A new data source may be added to the framework. It is required to acquire its corresponding ontology module by means of ontology learning according to the process defined in Fig. 1 (wrapping the data source to a MongoDB database then mapping the MongoDB database to an ontology module) [6, 7]. The learned knowledge of the new data source is represented in a separate ontology module, i.e. Module (New), and then the similarities between Module (New) and the global ontology are calculated [8]. At the end, Module (New) is integrated into the global ontology according to the ontology merging results.

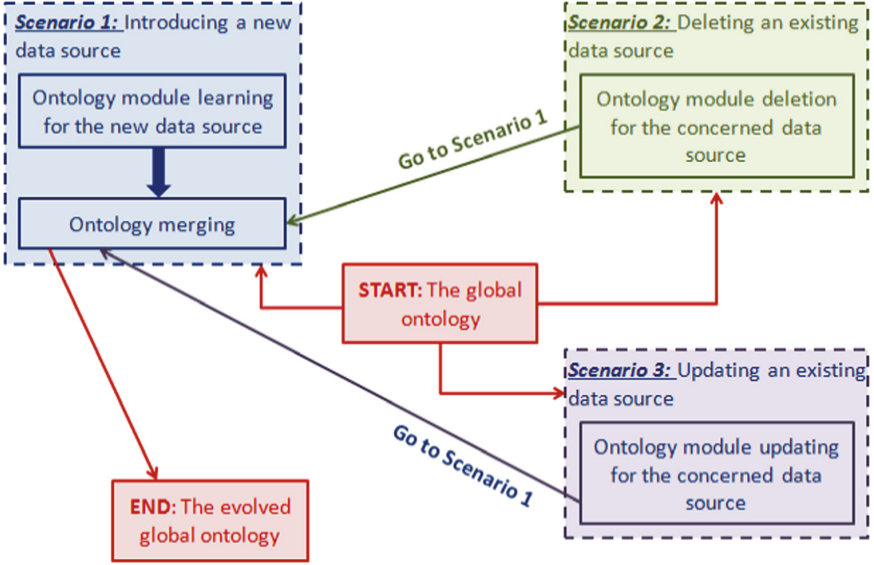


Fig. 3. Evolutionary scenarios for Big Data integration.

- *Scenario 2* (Deleting an existing data source): A current data source may leave the framework in case the data it contains become obsolete or due to reasons coming from the outside of the integration framework. Consequently, the corresponding ontology module (Module (i)) is deleted from the modules database and the other ontology modules have to be re-merged to constitute the new version of the global ontology.
- *Scenario 3* (Updating an existing data source): When an existing data source undergoes a modification of one of its entities (addition, deletion, renaming or modification), the corresponding ontology module must be updated accordingly and the global ontology as a consequence. We consider two levels of evolution: inter-modular evolution and intra-modular evolution. The inter-modular level concerns the global ontology update and is managed by the merging process. After evolving the corresponding ontology module, the ontology merging process has to be re-iterated to produce the new global ontology. The intra-modular evolution concerns the update of the concerned module itself. Detailed characterization of the intra-modular evolution is described in the next section.

4 Intra-modular Evolution Approach

4.1 Evolution Process

To carry out the ontology module evolution task, we propose an a priori process to address changes composed of three main steps as depicted in Fig. 4. For each change that occurs in the database, the corresponding mapping must be updated, since the

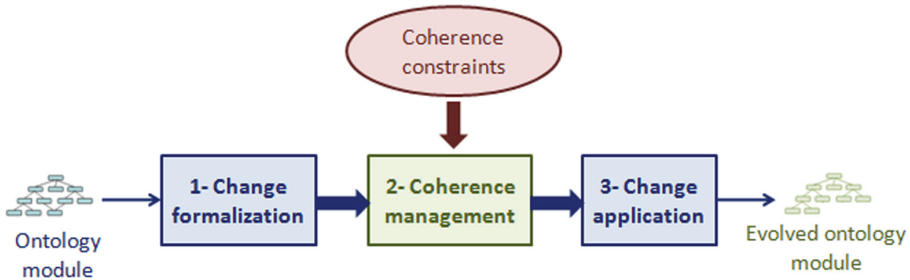


Fig. 4. Ontology evolution process.

ontology module has to always reflect the structure of the database and subsequently the data source.

Step 1: Change Formalization. This step consists of representing the change expressed over the database entities according to their correspondents into the ontology module. We define four types of changes similarly to the changes undergone in the database level, namely insertion, deletion, renaming and updating. Thus, the list of changes is given in Table 1. The list of changes is performed in compliance with the transformation rules presented in [6, 7].

Table 1. Changes performed against the ontology module analogously to those performed against the database.

| | Database level | Ontology module level |
|-----------|-------------------------------------|--|
| Insertion | -Collection | -Insert concept |
| | -Document into collection | -Insert individual |
| | -Basic field into document | -Insert dataTypeProperty |
| | -DBList into document | -Insert cardinality constraint |
| | -Embedded document into document | -Insert objectProperty |
| | -Reference with DBRef into document | -Insert objectProperty |
| | -Parent reference into document | -Insert hierarchy relationship |
| Deletion | -Collection | -Delete class |
| | -Document from collection | -Delete individual |
| | -Basic field from document | -Delete dataTypeProperty |
| | -DBList from document | -Delete cardinality constraint OR -Update cardinality constraint |
| | -Embedded document from document | -Delete objectProperty |
| | -Reference with DBRef from document | -Delete objectProperty |
| | -Parent reference from document | - Update hierarchy |
| Renaming | -Collection | -Rename class |
| | -Basic field into document | -Rename dataTypeProperty |
| | -Embedded document into document | -Rename objectProperty |
| | -Reference with DBRef into document | -Rename objectProperty |
| Updating | -Document | -Update individual |
| | -DBList | -Update cardinality constraint |
| | -Parent reference into document | -Update hierarchy |

Step 2: Coherence Management. This step consists of studying the impact of the selected change on the ontology module coherence. To do this, we reused the notion of “change kit” of [10, 11] to maintain a priori coherence constraints defined over the ontology module. Coherence constraints are discussed in Sect. 4.2 and the specification of change kit is given in Sect. 4.3.

Step 3: Change Application. In this step, changes are implemented over the ontology module. Hence, we obtain the evolved ontology module which already respects coherence constraints.

4.2 Coherence Constraints

We classify the coherence constraints that must be respected into three main categories as in [4]. Structural constraints represent constraints imposed by the ontology representation language which is OWL in our context. Logical constraints refer to checking the semantic correctness of the ontological entities. User-defined constraints describe specific user requirements. We give examples of these constraints.

OWL Language Constraints

- Isolated classes are not allowed
- A class which is a sub-class of another class must be defined in the ontology
- Each objectProperty relationship must link two classes that are defined in the ontology
- Each individual must be linked to a class that is defined in the ontology

Logical Constraints

- A class can not be disjoint with its super-class
- Two disjoint classes can not have common sub-classes

User’s Requirements

- Redundant information is not allowed
- Redundant links between two directly linked classes are not allowed
- The resulting ontology must respect the conventional requirements of the transformation rules described in [6].

4.3 Change Kit

A kit of change associates to each change the definition of its pre-conditions, its role, the mandatory additional changes, the optional additional changes and post-conditions. The specification of a kit of change is as follows:

- Pre-conditions: a set of predications that must be checked and controlled before applying the change
- Role: Description of the change

- Required additional changes: a set of changes that are necessarily attached to the current change to avoid coherence constraints violation
- Optional additional changes: a set of changes that may extend the current change
- Post-conditions: a set of predications that must be verified after the application of the change

4.4 Illustrative Example

As an example, we suppose that a modification in data source leads to insert an imbedded document in the corresponding MongoDB database. We describe the effect of inserting an embedded document (document B) into an existing document (document A) in the MongoDB database. According to the correspondences described in Table 1, this insertion leads to update the corresponding ontology module by inserting an objectProperty. The kit of change associated to the change “insert objectProperty” is as follows (Table 2).

The global evolution process monitors the following steps. The corresponding ontology module is updated according to the intra-modular ontology evolution process described in Fig. 4 and the change kit specified in Table 2 while respecting coherence constraints. Then the global ontology is updated according to the merging process performed against the updated module and the other previously existing modules that are related to the other data sources. The evolved global ontology is consequently structurally and logically consistent according to the researches developed in [6–8].

Table 2. Example of a change kit

| Kit of change “insert objectProperty” | |
|---------------------------------------|---|
| Role | The kit of change “insert objectProperty” serves to link two classes A and B with an objectProperty relationship |
| Pre-conditions | -The class corresponding to the document A exists already in the ontology module -The class corresponding to document B doesn’t exist in the ontology module -The objectProperty relationship doesn’t exist in the ontology module |
| Required additional changes | -Insert class corresponding to document B |
| Optional additional changes | – |
| Post-conditions | -The class corresponding to the document A belongs to the ontology module -The class corresponding to the document B belongs to the ontology module -The name of the objectProperty is the concatenation of the word “has” and the name of the class corresponding to document B -The domain of the inserted objectProperty is the class corresponding to document A -The range of the inserted objectProperty is the class corresponding to document B |

5 Related Work

To situate our research in the area of ontology evolution, we focus on a priori evolution approaches and concentrate on recent solutions addressing ontology evolution in Big Data integration frameworks.

In [10], a preventive approach that manages the inconsistencies generated by each change is described and a set of rules that must be maintained during the evolution of an ontology is defined. Authors define kits of changes to a priori manage the inconsistencies generated by each change. They rely on the UML specifications to take into account the maximum of evolution cases, independently of the ontology representation language and consider all conceptual relationships supported by the UML language, such as n-ary relationships, but do not define explicitly the type of coherence which is considered.

Authors in [11] present an evolution process composed of three main steps. The first step consists of presenting all the possible changes for the naRyQ ontological and terminological resource (OTR) evolution to the ontology engineer, from which he chooses the ones to be applied. The second step consists of preserving the coherence constraints (CC-coherence) of naRyQ during its evolution. To do this, authors adapted the notion of kit of changes of [10] to their needs, thus an additional set of changes is added automatically to maintain a priori the CC-coherence of the OTR before the application of the requested changes. In the third and last step, requested and additional changes are applied to the OTR.

Authors in [12] focused on addressing the need to reflect the evolution of ontologies used as global schemata onto the underlying data integration systems. They consider that when ontologies evolve, the changes should necessarily be rendered and used by the pre-existing data integration systems. They propose to answer query in data integration systems under evolving ontologies without mappings redefinition. This is ensured by rewriting queries among ontology versions and then sending them to the underlying data integration systems to be answered. Initially, the changes among ontology versions are automatically detected and described using a high level language of changes. These changes are then interpreted as sound global-as-view (GAV) mappings, used to produce equivalent rewritings among ontology versions.

In [13], authors present an approach that enables to integrate situational data coming from external providers, and to facilitate the co-evolution of data and analytical processes preserving backward compatibility. They introduce the Big Data Integration ontology that allows the isolation of analytical queries and applications from the technological details of the sources and accommodates syntactic evolution from the sources. Its goal is to model and integrate, in a machine-readable format, semi-structured data while preserving data independence regardless of the source formats or schema. The introduced ontology incorporates two layers to provide to the analysts an integrated and format-agnostic view of the sources. The global level provides a unified schema to query and relevant metadata about the attributes, while the source level deals with the physical details of each data source. This structure is exploited to handle the evolution of source schema via semi-automated transformations on the ontology upon service releases. Aided by semi-automatic techniques, a data

steward is responsible for, first incorporating to the source level the triple-based representation of the schema of newly incoming events (Ei) produced by APIs, and second make such data available for data analysts to query (Qi) by creating mappings from the source level to the global level. To semi-automatically adapt the BDI ontology to such evolution, authors present an algorithm to aid the data steward to enrich the ontology upon new releases to shield analytical processes, implemented on top of the global level, so that they do not crash upon new API version releases. This aims to adapt the source level to schema evolution in the events, so that the global level is not affected.

The following table summarizes advantages and drawbacks of these works (Table 3).

Table 3. Comparison between ontology evolution approaches.

| Approaches | Advantages | Drawbacks |
|------------|---|---|
| [10] | Anticipatory approach that takes into account the maximum of evolution cases | Does not define the type of coherence which is considered |
| [11] | Anticipatory approach that is based on a clear definition of ontology coherence | Does not propagate changes to the related artifacts |
| [12] | The proposed architecture can be placed on top of any traditional ontology-based data integration system, enabling ontology evolution | Does not consider local schema evolution, thus the ontology used as a global schema may contain inconsistencies |
| [13] | The proposed method handles schema evolution using a metadata-driven approach in the context of Big Data integration | Focuses only on enrichment and does not cover other change types (deletion, renaming, updating) |

From these perspectives, we notice the following remarks against the studied approaches.

Authors of [10, 11], although they develop preventive ontology evolution approaches, the latters do not fit evolution in data integration frameworks. On the other hand, evolution approaches developed in [12, 13] in the context of Big Data integration are not preventive and suffers from some limitations.

Authors of [13] aim to aid the data steward to only enrich the ontology upon new releases and do not consider other aspects of evolution such as deletion and modification. Moreover, they address the evolution locally in the source level and do not propagate changes to the global level. Conversely, we consider all evolution types namely insertion, deletion, update and renaming and we propagate changes to the global ontology by means of merging re-iteration.

Authors of [12] are interested to ontology evolution in data integration like us. But, while they focus on propagating changes from the ontological level to the data sources level, we make the opposite and try to communicate changes over the data sources to the global ontology.

The notion of change kit was initially proposed by [10] but relying on UML specification, authors of [11] adapted it to the specificities of the OWL language, and in our work, we adapted it to the context of ontology-based Big Data integration founded on data sources evolution.

Our approach has three main advantages. It firstly covers the entire ontology evolution cycle and manages incoherencies that are likely to occur in a priori manner, secondly relies on the use of background releases i.e. previously developed components [6–8] to potentially decrease, or even eliminate, user involvement, and finally fits all evolution scenarios.

6 Conclusions and Future Work

Ontologies need to be updated across their life cycle to reflect new requirements and must remain coherent. We are interested in this work to the ontology evolution in the context of Big Data integration. The majority of existing works about ontology-based Big Data integration ignores evolution issues.

When an ontology is used as a component of an advanced information system, its evolution is a complex process and raises several challenges such as the formal representation of ontology changes, the verification of ontology consistency when applying the ontology changes, and the propagation of these changes to the ontology related artifact. We discussed related work relative to a priori ontology evolution and Big Data integration evolution. We presented evolution scenarios in the context of Big Data integration and proposed a solution to deal with modular ontology evolution while considering changes performed against the data sources.

There are many interesting future directions. A prominent one is to explore how to manage change history and to record changes performed against the ontology. As a short-term goal, we plan to integrate such functionality to our approach. Other avenue of research would be to propose an approach to enhance the change propagation step to the global ontology.

References

1. Gupta, R., Gupta, H., Mohania, M.: Cloud computing and big data analytics: what is new from databases perspective? In: Srinivasa, S., Bhatnagar, V. (eds.) BDA 2012. LNCS, vol. 7678, pp. 42–61. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-35542-4_5](https://doi.org/10.1007/978-3-642-35542-4_5)
2. Zikopoulos, P., Eaton, C.: Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. McGraw–Hill/Osborne Media, New York City (2011)
3. Boden, C., Karnstedt, M., Fernandez, M., Markl, V.: Large-scale social-media analytics on stratosphere. In: Proceedings of the 22nd International Conference on World Wide Web Companion, pp. 257–260 (2013)
4. Haase, P., Stojanovic, L.: Consistent evolution of OWL ontologies. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 182–197. Springer, Heidelberg (2005). doi:[10.1007/11431053_13](https://doi.org/10.1007/11431053_13)

5. Abbes, H., Gargouri, F.: Big data integration: a MongoDB database and modular ontologies based approach. *Procedia Comput. Sci.* **96**, 446–455 (2016)
6. Abbes, H., Boukettaya, S., Gargouri, F.: Learning ontology from Big Data through MongoDB database. In: *Proceedings of IEEE/ACS 12th International Conference of Computer Systems and Applications*, pp. 1–7 (2015)
7. Abbes, H., Gargouri, F.: M2Onto: an approach and a tool to learn OWL ontology from MongoDB database. In: *Madureira, A.M., Abraham, A., Gamboa, D., Novais, P. (eds.) ISDA 2016. AISC*, vol. 557, pp. 612–621. Springer, Cham (2017). doi:[10.1007/978-3-319-53480-0_60](https://doi.org/10.1007/978-3-319-53480-0_60)
8. Abbes, H., Gargouri, F.: Structure based modular ontologies composition. In: *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, Agadir, Morocco (2016)
9. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.* **10**(8), 707–710 (1966)
10. Jaziri, W., Sassi, N., Gargouri, F.: Approach and tool to evolve ontology and maintain its coherence. *Int. J. Metadata Semant. Ontol.* **5**(2), 151–166 (2010)
11. Touhami, R., Buche, P., Dibie, J., Ibanescu, L.: Ontology evolution for experimental data in food. In: *Garoufallou, E., Hartley, R.J., Gaitanou, P. (eds.) MTSR 2015. CCIS*, vol. 544, pp. 393–404. Springer, Cham (2015). doi:[10.1007/978-3-319-24129-6_34](https://doi.org/10.1007/978-3-319-24129-6_34)
12. Kondylakis, H., Plexousakis, D.: Ontology evolution without tears. *Web Semant.: Sci. Serv. Agents World Wide Web* **19**, 42–58 (2013)
13. Nadal, S., Romero, O., Abelló, A., Vassiliadis, P., Vansummeren, S.: An integration-oriented ontology to govern evolution in big data ecosystems. In: *Proceedings of the EDBT/ICDT 2017 Joint Conference. Published in the Workshop OLAP* (2017)

Information Systems

14th European, Mediterranean, and Middle Eastern
Conference, EMCIS 2017, Coimbra, Portugal, September
7-8, 2017, Proceedings

Themistocleous, M.; Morabito, V. (Eds.)

2017, XV, 680 p. 144 illus., Softcover

ISBN: 978-3-319-65929-9