# Ontology Versioning Framework for Representing Ontological Concept as Knowledge Unit

Archana Patel[a] and Sarika Jain[b]

[a] *Institute of Computer Science, Freie University Berlin, Berlin, Germany*
[b] *Department of Computer Applications, National Institute of Technology Kurukshetra, Haryana, India*

**Abstract**

Nowadays ontologies are used in everywhere and provide a reusable piece of knowledge about a specific domain. However, those pieces of knowledge are not static and change over the time in order to fulfil the requirements of the different task. So, it is essential that changes in ontologies should be managed very well. Ontology versioning mechanism is used to keep the track of the ontology changes via making the relationship with previous version of the ontologies. Many ontologies encode reality by representing ontological concept as a knowledge unit. Till the date, no work has been started towards to solve the ontology versioning problem when ontological concept store based on the idea of knowledge unit. To overcome this problem, we present an ontology versioning framework which is capable to maintain the relationship among different version of ontology explicit. We show operational analysis of the proposed work for the better understanding about ontology versioning framework.

**Keywords**

Versioning, knowledge Unit, Annotation, Ontology

## 1. Introduction

Ontology is widely used for sharing the information. A classical ontology comprises classes, instances, axioms and properties. These properties can be data property (relate a class with data value) and object property (relate two classes with each other) [1]. In opposite to classical ontology, a realistic ontology comes into the scenario where every concept is stored as a knowledge unit by comprising classes, set of defining properties (that define the concept uniquely or distinguish it with others), set of cancellable properties (that may or may not true for the concept), set of exceptions, UNK (used to complete the concept), instance and axioms [2]. Ontology changed over the time according to the need of the application that generates different version of the same ontology. Ontologies undergo changes due to one or all of the following reasons:

- Changes in the domain
- Adaptations to different applications.
- Changes in the conceptualization or understanding of the domain.
- To correct errors
- Catering the ontology to a new phenomenon

Ontology versioning implies that an ontology has various variants. In fact, these variants frequently drive from modifications to an existing variant of the ontology and thus build a derivation tree [3]. Klein et al. [4] describe ontology versioning as a process that manage the ontology changes and their effects by maintaining and creating diverse variant of the ontology. Ontology versioning maintains the synergy between different versions of the ontology that creates at the same time. Ontologies have a general tendency to have more changes the earlier they are in their lifecycle. Modularized ontologies generally change asynchronously, i.e., without changes in a module may begin waiting for the changes in some another module to commit. There are two categories of changes. One affecting the TBOX i.e., the ontology and the other affecting the ABOX i.e., the content. Table 1 lists some

*ISIC'21: International Semantic Intelligence Conference, February 25-27, 2021, New Delhi, India*

archanamca92@gmail.com (A. Patel); jasarika@nitkkr.ac.in (S.Jain)
0000-0002-7505-7226 (A.Patel); 0000-0002-7432-8506 (S.Jain)

example of Ontology changes and some example of content changes.

**Table 1**
Changes in Ontology

| **T-Box** |
|---|
| 1. Changes in Hierarchy: Adding/removing a class or property, Merging two classes or properties, Splitting a class into two classes.<br>2. Changes involving Classes: Renaming a class, Changing label, comment or cardinality of a class, Changing or removing parent, Adding/removing a child, Adding/removing a property to/from a class.<br>3. Changes involving Properties: Renaming a property, Changing the domain/range/sub-property, reference/label/comment of a property.<br>4. Other change types: Property characteristics, Equality or inequality, Restricted cardinality, Union or intersection. |
| **A-Box** |
| 1. Changes involving instances: Renaming an instance, Changing annotation of the instances, Adding or removing instances, Adding/removing properties and their values.<br>2. Changes involving properties: Renaming a property, Changing the domain/range/sub-property, reference/label/comment of a property |

When knowledge managers locate the changes between different versions of an ontology, we call it comparing the two ontologies as opposed to versioning the two ontologies. Why Versioning Systems are required?

- Implementing FAIR vocabularies: It is one of the best practices for implementing FAIR vocabularies and ontologies on the web.
- Backward Compatibility: The tools that work with older versions of the ontologies are functioning in new versions too.
- Resolving semantically (than syntactically): If conceptual relations between different versions are constructed, it becomes possible to re-interpret the data and knowledge under different ontology versions.
- Require changes in application logic: Applications need to update their logic to reference the new ontology. If the ontology change has a non-dynamic response, it may affect the use of these ontologies by higher level applications.

Klein et al. [4] describe the various requirements on an ontology versioning framework that are useful to create different versions of internal ontologies: Identification (a versioning framework should provide/identify the concept or relation in an unambiguous manner), Change specification (versioning framework should be able to make the relationship explicitly from one version of the concept with other versions), Transparent evolution (versioning methodology on the web should make clear which part of the data can still be valid interpreted), Task awareness (a framework should exhibit the behavior or task that helps in providing the transformations between different versions), Tackle to untraced changes (a versioning framework should able to determine whether two versions of an ontology are compatible or not). The key problems of ontology versioning are (1) how to check (track and detect) ontology changes, (2) how to distribute (release) new versions of ontologies, (3) how to merge different versions of an ontology [5].

In order to reduce these problems, the versioning information is encoded at meta level and term level but still there is a requirement to develop sophisticated versioning mechanisms to incorporate ontology changes. In this paper, we focus on "how to encode versioning information in an ontology when ontological concept is stored as a knowledge unit". The remaining paper is presented as given below: section 2 shows the literature of the versioning information, section 3 describes the ontology versioning framework for the realistic ontology, section 4 resembles the operational analysis for the storage of versioning information along with the comparison with existing work and last section concludes the proposed work.

## 2. Literature

Ontology versioning is required in order to handle ontology changes. The challenges and research opportunities of ontology versioning are:

- Two ontologies with different text serializations may be conceptually the same. The difference in text representations may be due to different storage syntax or due to different order of definitions.

- Distributed authoring and management (to identify versions of an ontology in distributed environments).
- Application-level dependencies need to be considered.
- To specify change logs between ontology versions explicitly.
- Identify additional ontology changes

The first approach for ontology versioning is proposed by Klein and Stojanovic [6] but the problem was unavailability of the standard ontology versioning system like CVS that use in software development field. The versioning information has been encoded at the meta level and term level. The meta level versioning information describes the meta detail of the ontology and term level versioning information describes the detail of every terms that are stored in ontology. The versioning information is stored by using different tags that available under the different namespace like /terms/, /elements/1.1/ etc [7]. The following tags are used at the meta level and term level for the storage of versioning information:

Meta Level Tags: Ontology uses IRI to identify the ontology and owl:versionIRI is used to identify the specify version of the ontology. dc:contributor is used to define the responsibility of the entity that make contribution to the resources. terms:license offers official permission to work with the resource. dc:description describes the resources. dc:title assigns the name to the resources. dc:creator describes the entity which is responsible for making the resources. dc:publisher offers the available resources. dcterms:modified updates the date according to the status of the resource. dc:language describes the language of the given resources. oboInOwl:date tells the date which is associated with the event. dcterms:issued describes the issuance date of the resources. dcterms:bibliographicCitation provides bibliographic reference of the resource.

Term Level Tags: The annotation property is used for the storage of the term level versioning information. hasVersion, Issued, Modified, Replaces, Status, date, created_by, versionInfo, creator, contributor, terms, author, priorVersion, backwardCompatibleWith and incompatibleWith etc can be created under the annotation property in order to store the appropriate detail of each entity. Deprecation is a feature which is used to deprecate the term (deprecating a term means that term will not use in new document). We can deprecate classes and properties according to the needs.

For example, Biological Collections Ontology (BCO) [8] has store owl:versionIRI, dc:contributor, terms:license at the meta level and annotation properties namely hasVersion, Issued, Modified, Replaces, Status has been created in order to stored term level versioning information. The versioning information of class Taxon has hasVersion: http://rs.tdwg.org/dwc/terms/history/#Taxon-2014-10-23, Replaces: http://rs.tdwg.org/dwc/terms/Taxon-2009-09-21, Status: recommended, Issued: 2008-11-19, Modified: 2014-10-23. Deprecated property is used to deprecate the class BCO-0000061.

Different portals stored the versioning information of the terminologies or ontologies by using different tag and annotation properties. The meta level versioning information is encoded under the <owl:ontology> tag that can be easily seen if an ontology file is opened into notepad. The term level versioning information can be easily seen if you open the ontology in the protege tool or any other tool. In the protege, after clicking the concept, all information of that concept is shown under the annotation properties. The most widely used ontology portals are Bio Portal, Agro Portal, OBO lib, AberOWL repository and ontology lookup services.

- Ontology Versioning in Bio Portal [9]: Bioportal uses the indexing mechanism in order to support ontology versioning. A stable ontology identifier is used to index the ontology and each versions of an ontology is indexed with version identifier. The version identifier changes from one version to another when new version of an ontology is derived. The web services use the ontology and its versions by ontology identifier and version identifier respectively.
- Ontology Versioning in OBO lib: The URI is used in the OWL language to identify all the entities of an ontology like classes, instances and ontology itself. The permanent URL (called PURL) of an ontology with standard base prefix [10] are used in OBO repository of ontologies in order to check if new versions of an ontology are updated and the tools are still

functioning that support older versions of an ontology. OBO uses versioning system where each version of an ontology has a unique identifier either in form of metadata tags and date or numbering system [11].

- Ontology Versioning in Agro Portal [12]: AgroPortal supports ontology versioning by utilizing the concept of 'submission'. A 'submission' object is attached with an ontology when the same ontology has been uploaded in the portal. Whenever an ontology is uploaded or pulled from its original location then every time a new submission object is created.
- AberOWL Repository: It is a framework that provides ontology-oriented access of the biological data [13]. The framework contains repository of the ontologies that are related to the biological data, set of web services, various frontends and provide reasoning over the stored ontologies. The versioning information is encoded at the meta and term level by using various ontology tags.
- Ontology Lookup Services (OLS): It provides single point access to the latest version of the biomedical ontologies from the repository [14]. OLS shows ontology history in order to describes the changes that occurs in different version of an ontology by calculating various parameters like add classes, add level, add synonyms, add definition, delete definition etc.

Available portals stored the classical ontologies and encode the versioning information inside itself. The main problem for the storage of the versioning information inside the classical ontology is how to deprecate the term/resources, how to use same syntax for creation of the versioning properties under the annotation. The process of storing the versioning information inside the realistic ontology is not cover yet. Here our focus is to present the versioning framework for the realistic ontology where every concept is represented as a whole. It is a first attempt to show the encoding of the versioning information inside realistic ontology.

## 3. Ontology Versioning Framework

The realistic ontology in accordance with the present subject matter represents rule, exception, and hierarchy of concepts to offer a realistic description of the real-world entities. A node or a unit of knowledge (UoK) to represent a knowledge packet takes the form of the following tuple [15]:

$$\mathbf{D}\,[\mathbf{TE},\mathbf{AE},\mathbf{VE},\mathbf{PE}](\boldsymbol{\omega}) = < \boldsymbol{DF}(\boldsymbol{\gamma}),\mathbf{CF},\mathbf{C}(\boldsymbol{\delta}),\mathbf{G},\mathbf{S},\mathbf{I} > \tag{1}$$

TE, AE, VE, PE are the textual encryption, audio encryption, video encryption and pictorial encryption of the class/concept/decision D respectively. DF, CF and C are the distinctive features, cancellable features and exceptions of the class/concept/decision D respectively. G and S are the general and specific class/concept/decision. The parameters $\gamma, \delta,$ and $\omega$ represents 0-degree, 1-degree and 2-degree of the strength of the class/concept/decision. I represents instances of the class/concept/decision D that takes following form [16]:

$$\mathbf{I}\,[\mathbf{TE},\mathbf{AE},\mathbf{VE},\mathbf{PE}] = < \boldsymbol{DF},\mathbf{CF},\mathbf{SD},\mathbf{TD} > \tag{2}$$

SD and TD are the spatial and temporal details of the instance I respectively. The below mentioned RDF/XML codes show the storage of distinctive feature (distinctive feature 'nature' of the class 'Emergency' with value 'sudden'), cancellable feature (cancellable feature 'hasWarning' of concept 'Emergency' with a default value 'no') and instance (spatial and temporal information of an instance 'Agartala–2008' of the concept 'Emergency') in the realistic ontology. All the distinctive and cancellable features are encoded by creating 'DistinctiveFeatures' and 'CancellableFeatures' properties; the SD and TD information about the instances are stored by creating 'SpatialInfo' and 'TemporalInfo' properties under the annotation properties.

The versioning framework for the realistic ontology is presented in figure 1. In the realistic ontology, we need to store the versioning information about the classes and properties (distinctive and cancellable features) but do not need to store this information for the instances because all the instances are already stored with TD and SD in realistic ontology. TD and SD show the temporal details (time and date) and spatial detail (space of the instance) of the Corresponding instances. In case, when we want to store more information about the instances like creator, contributor, saved-by and many more then we follow the same process as describes in section 4 for the classes. All the knowledge about the instances refer to the assertional knowledge and subject to the A-Box. The knowledge about the classes and relations refer to the terminological knowledge and subject to the T-Box. They both together form the knowledge base.

***The RDF/XML code for the storage of distinctive features***

```
<owl:AnnotationProperty rdf:about="http://www.semanticweb.org/lab/ontologies/2019/6/untitled-ontology-810
#DistinctiveFeatures"/>
<owl:AnnotationProperty rdf:about="http://www.semanticweb.org/lab/ontologies/2019/6/untitled-ontology-810
#nature">
<rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/lab/ontologies/2019/6/untitled-ontology-810
#DistinctiveFeatures"/>
</owl:AnnotationProperty>
<owl:Class rdf:about="http://www.semanticweb.org/lab/ontologies/2019/6/untitled-ontology-810#Emergency">
 <hasAim rdf:datatype="http://www.w3.org/2001/XMLSchema#string">nature </sudden></owl:Class>
```

***The RDF/XML code for storage of cancellable features with a default value***

```
<owl:AnnotationProperty rdf:about="http://www.semanticweb.org/lab/ontologies/2019/6/untitled-ontology-810
#CancellableFeatures"/>
<owl:AnnotationProperty rdf:about="http://www.semanticweb.org/lab/ontologies/2019/6/untitled-ontology-810
#hasWarning">
<rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/lab/ontologies/2019/6/untitled-ontology-810
#CancellableFeatures"/>
</owl:AnnotationProperty>
<owl:Class rdf:about="http://www.semanticweb.org/lab/ontologies/2019/6/untitled-ontology-810#Emergency">
<hasWarning rdf:datatype="http://www.w3.org/2001/XMLSchema#int">no</hasWarning></owl:Class>
```

***The RDF/XML code for the storage of SD and TD information of the instance Agartala_2008***

```
 <owl:AnnotationProperty rdf:about="http://www.w3.org/2000/01/rdf-schema#SpatialInfo"/>
<owl:AnnotationProperty rdf:about="http://www.w3.org/2000/01/rdf-schema#TemporalInfo"/>
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/lab/ontologies/2019/4/untitled ontology 788#Agartala_2008">
<rdfs:TemporalInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"> 2008-07-26T19:55:00</rdfs:TemporalInfo>
 <rdfs:SpatialInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Agartala</rdfs:SpatialInfo>
</owl:NamedIndividual>
```

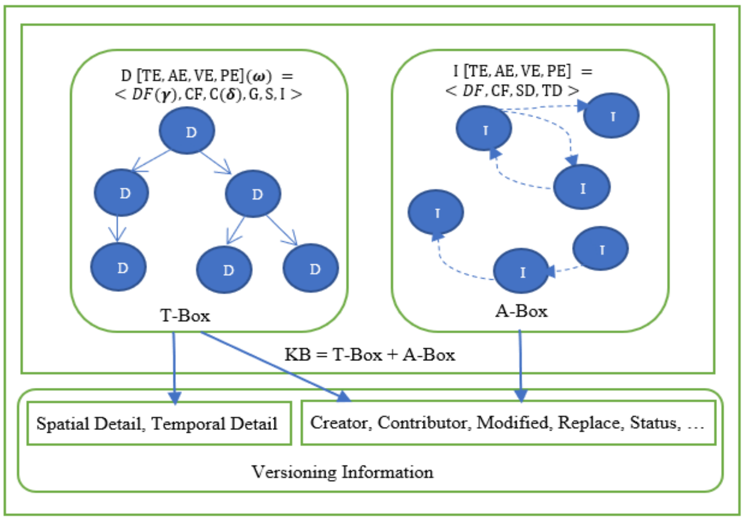

D [TE, AE, VE, PE]($\omega$) = < $DF(\gamma)$, CF, C($\delta$), G, S, I >
D
T-Box
I [TE, AE, VE, PE] = < $DF$, CF, SD, TD >
I
A-Box
KB = T-Box + A-Box
Spatial Detail, Temporal Detail
Creator, Contributor, Modified, Replace, Status, …
Versioning Information

**Figure 1:** Versioning Framework for the Realistic Ontology

## 4. Operational Analysis

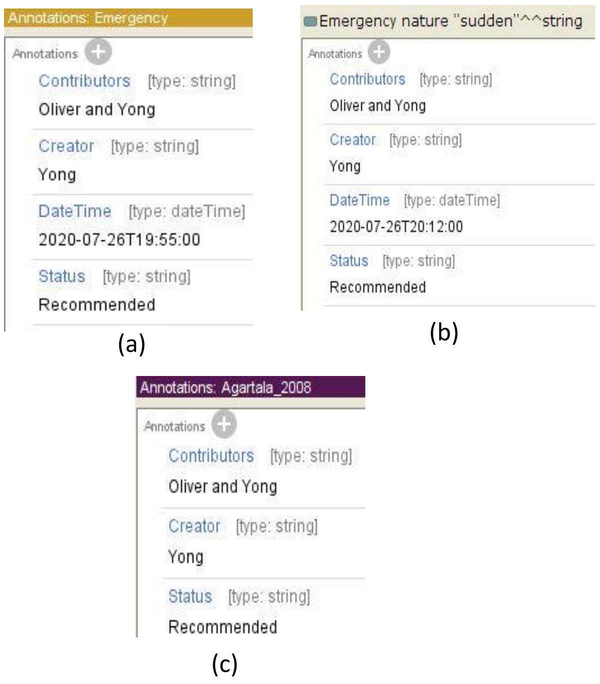
The entities of an ontology are subject to the change and these changes occur at the meta level and term level (within the ontology). The meta level change updates the meta information of an ontology like versionIRI, contributor, license and etc. The term level change includes classes, properties or features and instances. These changes provide different version of the same ontology. This section shows, how to add term level (classes, properties and instances) versioning information in the realistic ontology. The storage of meta level versioning information inside the realistic ontology is similar to the classical ontology.

A. **Storage of Versioning Information for the Classes:** We use annotation properties for the storage of versioning information about the classes as similar to the classical ontology. The below mentioned RDF/XML code shows the versioning information namely 'Contributors', 'Creator', 'DateTime' and 'Status' about the class 'Emergency'. The screenshot attached as figure 2 (a) shows the protégé tool for the storage of versioning information about the class 'Emergency'.

B. **Storage of Versioning Information for the Properties:** The distinctive and cancellable features (properties) of the classes are stored by creating properties 'DistinctiveFeatures' and 'CancellableFeatures' under annotation property. We annotate all the 'DistinctiveFeatures' and 'CancellableFeatures' properties for the storage of versioning information. The below mentioned RDF/XML code shows the versioning information namely 'Contributors', 'Creator', 'DateTime' and 'Status' about the distinctive feature 'nature' that value is 'sudden' for the class 'Emergency'. Figure 2(b) shows the screenshot of the protégé tool for the storage of versioning information about the distinctive feature 'nature' that value is 'sudden' for the class 'Emergency'.

C. **Storage of Versioning Information for the Instances:** All the instances are already stored with spatial and temporal details in the realistic ontology. There no need to store this information again. But for the storage of the rest of the versioning information like creator, contributors, status etc, we use annotation property. The below mentioned RDF/XML code shows the versioning information namely 'Contributors', 'Creator', and 'Status' about an instance 'Agartala_2008' of the class 'Emergency'. Figure 2 (c) shows the screenshot of the protégé tool for the storage of versioning information about the instance 'Agartala_2008' of the class 'Emergency'.

By the above-described way, we incorporate all the changes inside an ontology that create different version of an ontology. Now, every concept in the realistic ontology contains all the information about the entity that helps to understand the different version of an ontology. There is no need to store the spatial and temporal information about the instances as they already contained in the information while entering the systems.


Annotations: Emergency

Annotations

Contributors [type: string]
Oliver and Yong

Creator [type: string]
Yong

DateTime [type: dateTime]
2020-07-26T19:55:00

Status [type: string]
Recommended

(a)

Emergency nature "sudden"^^string

Annotations

Contributors [type: string]
Oliver and Yong

Creator [type: string]
Yong

DateTime [type: dateTime]
2020-07-26T20:12:00

Status [type: string]
Recommended

(b)

Annotations: Agartala_2008

Annotations

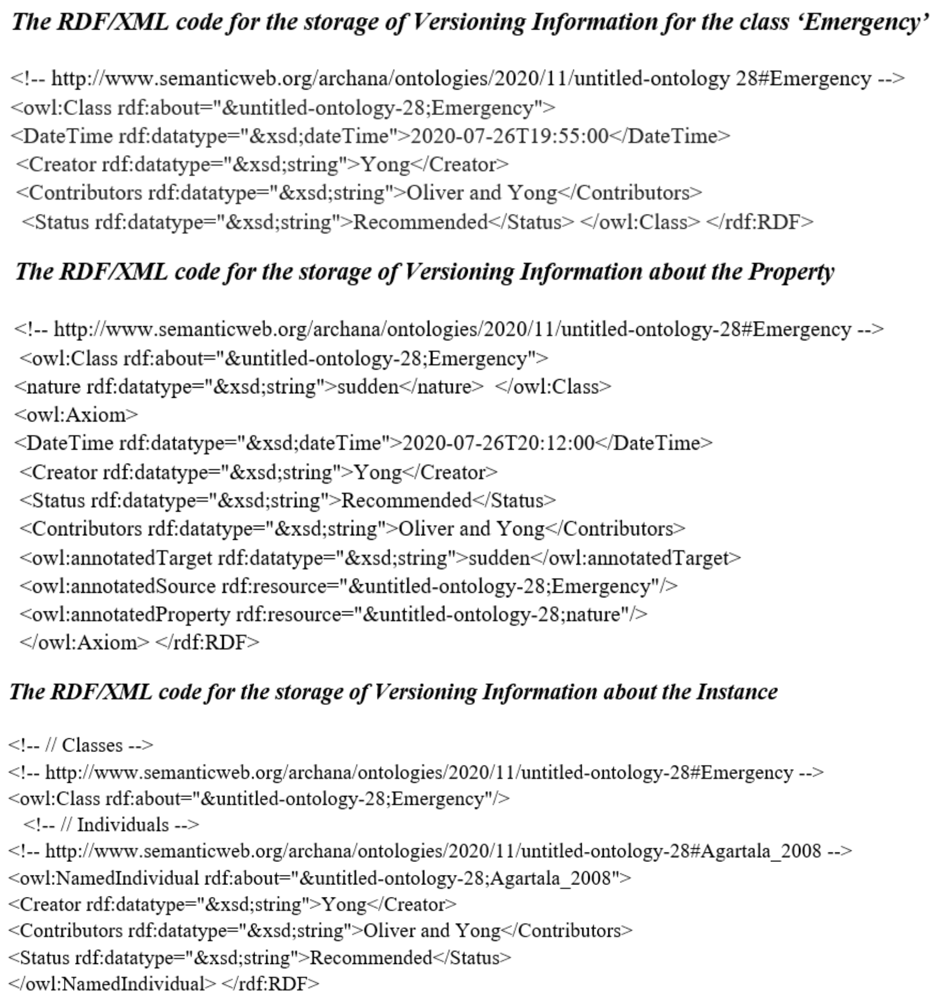
Contributors [type: string]
Oliver and Yong

Creator [type: string]
Yong

Status [type: string]
Recommended

(c)

**Figure 2:** Screenshot of the Protégé Tool for the Storage of Versioning Information (a) Class 'Emergency' (b) Distinctive Feature 'nature' (c) Instance 'Agartala_2008'



***The RDF/XML code for the storage of Versioning Information for the class 'Emergency'***

```
<!-- http://www.semanticweb.org/archana/ontologies/2020/11/untitled-ontology 28#Emergency -->
<owl:Class rdf:about="&untitled-ontology-28;Emergency">
<DateTime rdf:datatype="&xsd;dateTime">2020-07-26T19:55:00</DateTime>
 <Creator rdf:datatype="&xsd;string">Yong</Creator>
 <Contributors rdf:datatype="&xsd;string">Oliver and Yong</Contributors>
  <Status rdf:datatype="&xsd;string">Recommended</Status> </owl:Class> </rdf:RDF>
```

***The RDF/XML code for the storage of Versioning Information about the Property***

```
<!-- http://www.semanticweb.org/archana/ontologies/2020/11/untitled-ontology-28#Emergency -->
 <owl:Class rdf:about="&untitled-ontology-28;Emergency">
<nature rdf:datatype="&xsd;string">sudden</nature>  </owl:Class>
<owl:Axiom>
<DateTime rdf:datatype="&xsd;dateTime">2020-07-26T20:12:00</DateTime>
 <Creator rdf:datatype="&xsd;string">Yong</Creator>
 <Status rdf:datatype="&xsd;string">Recommended</Status>
 <Contributors rdf:datatype="&xsd;string">Oliver and Yong</Contributors>
 <owl:annotatedTarget rdf:datatype="&xsd;string">sudden</owl:annotatedTarget>
 <owl:annotatedSource rdf:resource="&untitled-ontology-28;Emergency"/>
 <owl:annotatedProperty rdf:resource="&untitled-ontology-28;nature"/>
 </owl:Axiom> </rdf:RDF>
```

***The RDF/XML code for the storage of Versioning Information about the Instance***

```
<!-- // Classes -->
<!-- http://www.semanticweb.org/archana/ontologies/2020/11/untitled-ontology-28#Emergency -->
<owl:Class rdf:about="&untitled-ontology-28;Emergency"/>
  <!-- // Individuals -->
<!-- http://www.semanticweb.org/archana/ontologies/2020/11/untitled-ontology-28#Agartala_2008 -->
<owl:NamedIndividual rdf:about="&untitled-ontology-28;Agartala_2008">
<Creator rdf:datatype="&xsd;string">Yong</Creator>
<Contributors rdf:datatype="&xsd;string">Oliver and Yong</Contributors>
<Status rdf:datatype="&xsd;string">Recommended</Status>
</owl:NamedIndividual> </rdf:RDF>
```

## 5. Conclusion

Ontology versioning is a mechanism to store and identify different versions of the same ontology. It can be achieved when user has complete information about the entities used in ontology. Ontology versioning information is encoded at the meta and term level by using different tags. The process to store versioning information inside the classical ontology is shown by various ontology portals/repositories. But how to store versioning information in the realistic ontology is not being covered yet. In this paper, we present the versioning framework for the realistic ontology that assists users to easily analyze the different version of a realistic ontology. We have shown RDF/XML code and screenshot of the protégé tool for the demonstration of the proposed versioning framework. In future, we will work to deprecate the entities and to reduce the problem of storing Versioning information related with the entity inside a realistic and classical ontology.

## References

[1] D. Kumar, A. Kumar, M. Singh, A. Patel, S. Jain, An online dictionary and thesaurus. Journal of Artificial Intelligence Research & Advances, (2019), 6(1), 32-38.

[2] S. Jain, A. Patel, Smart Ontology-Based Event Identification. In 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) (2019), pp. 135-142), IEEE.

[3] M. C. Klein, D. Fensel, Ontology versioning on the Semantic Web. In SWWS (2001), pp. 75-91.

[4] M. Klein, A. Kiryakov, D. Ognyanoff, D. Fensel, Finding and specifying relations between ontology versions. In Proceedings of the ECAI-02 (2002), Workshop on Ontologies and Semantic Interoperability. Lyon , pp. 442-456.

[5] M. Klein, D. Fensel, A. Kiryakov, D. Ognyanov, Ontology versioning and change detection on the web. In International Conference on Knowledge Engineering and Knowledge Management (2002), pp. 197-212. Springer, Berlin, Heidelberg.

[6] M. C. A. Klein, Change management for distributed ontologies, (2004).

[7] DCMI Metadata Terms, URL: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/#http://purl.org/dc/terms/bibliographicCitation

[8] R. L. Walls, J. Deck, R. Guralnick, S. Baskauf, R. Beaman, S. Blum, M. A. Gandolfo, Semantics in support of biodiversity knowledge discovery: an introduction to the biological collections ontology and related ontologies (2014), PloS one, 9(3), e89606.

[9] N. F. Noy, N. H. Shah, P. L. Whetze, B. Dai, M. Dorf, N. Griffith, M. A. Musen, BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic acids research, (2009), W170-W173.

[10] Version Control, URL: https://en.wikipedia.org/wiki/Version_control

[11] OBO Foundry, URL: http://www.obofoundry.org/principles/checks/fp_004

[12] C. Jonquet, A. Toulet, E. Arnaud, S. Aubin, E. D. Yeumo, V. Emonet, P. Larmande, AgroPortal: A vocabulary and ontology repository for agronomy. Computers and Electronics in Agriculture (2018), 144, 126-143

[13] Aber OWL, URL: http://aber-owl.net/about/

[14] OLS Ontology Search URL: https://www.ebi.ac.uk/ols/ontologies

[15] A. Patel, S. Jain, A Novel Approach to Discover Ontology Alignment, Recent Advances in Computer Science and Communications (2020,) 13: 1, Doi: https://doi.org/10.2174/2666255813666191204143256

[16] A. Patel, S. Jain, S. K. Shandilya, Data of Semntic Web as Unit of Knowledge. Journal of Web Engineering (2018), 17(8), 647-674.