Towards Dynamic Ontology

Integrating Tools of Evolution and Versionning in Ontology

Perrine Pittet, Christophe Cruz, Christophe Nicolle
LE2I, UMR CNRS 5158
University of Bourgogne
Dijon, France
{perrine.pittet, christophe.cruz, christophe.nicolle}@u-bourgogne.fr

***Abstract*—Since Gruber's definition, a lot of works focused on evolution or versioning issues. Not enough attention has been paid to integrated solutions which resolve both these two purposes. In this paper we present a new semantic architecture that combines versioning tools with the evolution process. This architecture called VersionGraph is integrated in the source ontology since its creation in order to make it possible to evolve and to be versioned.**

Keywords-evolution; versioning; Versiongraph; ontology lifecycle;change operations;

I. INTRODUCTION

Many works have been published about the definition of ontology to bridge the gap of semantic heterogeneity. Literature now generally agrees on the Gruber's terms to define an ontology: explicit specification of a shared conceptualization of a domain [1]. The domain is the world that the ontology describes. It can be a general domain or a more specific one. This description uses a vocabulary of concepts which is understandable and agreed by people of the domain; here is the meaning of "shared conceptualization". The ontology can be implemented in several languages with a different level of formalization and expressivity, with no ambiguity that's why ontology is an "explicit specification".

The development of ontology is becoming a common task and an inescapable support for information systems interoperability [2]. This research domain is mature and the first feedbacks arise. New scientific deadlocks are identified concerning the lifecycle of ontology especially the evolution phase. Discussing about those issues leads us to first ask what part of the ontology definition is concerned by this lifecycle and where the evolution can be situed.

Regarding to [3] Ontology lifecycle depends from changes occurring in the domain, conceptualization and/or specification of the ontology. Moreover, as depicted in Figure 1 (red dotted arrows), a change on one of this identified sources can impact a change in the other sources. Figure 1 shows the causes of changes related to the domain (a), the conceptualization (b) and the specification (c). We can notice that a change cause in (a) and (b) can have a change consequence in (b) and (c). Proposition a new classification of the identified changes in the state of art of [26] we have identified two types of change interaction:

Firstly, the domain can impact conceptualization. These changes are similar to changes in database schemas [5]. For example new concepts/relationships must be considered or existing concepts/relationships must be improved or deleted. That's the role of Domain Evolution [6] or Domain Fusion (Ontology Integration [7], Ontology Merging [8]) proposals. The domain can also affect specification. For example a complete translation to a new specification corresponds to Ontology Translation_1[9] proposals.

Secondly, the conceptualization can impact the specification. For instance new models in the domain are introduced and require a change in the concept/relationships organization, formalization and expressivity. That's the purpose of Conceptualization Evolution [10] and Conceptual Revision (Ontology Debugging [11]) proposals. Nevertheless, we note that four types of change, used to resolve conceptual heterogeneity (conceptualization part), don't impact the ontology itself: Ontology Mapping [12], Ontology Matching [13], Ontology Articulation [13] and Ontology Morphism [14]. These last ones add an external mapping to bridge the semantic gap. We argue that a change in the specification doesn't impact nor the conceptualization nor the domain when the specification language is enough rich to express this change. It's the case of Description Logics languages [15] which display different levels of expressivity by holding different ontology constructors. So, we can choose one of them depending on the level of expressivity we need.

From this discussion, we deduce that the evolution phase concerns the domain and the conceptualization of the ontology. The Conceptualization Evolution is a direct consequence of the Domain Evolution

The new research area which aims at resolving the impact of change management on ontology is known as Ontology Dynamics [16] Ontology Dynamics deals with all issues concerning changes impacting the ontology (change of the domain, change in the conceptualization, or change in the specification), especially maintenance and evolution. The ontology development is a dynamic and incremental process starting with the creation of a brute ontology which has to be revised, refined and populated [17]. In the literature, a lot of papers have addressed the problem of managing the lifecycle of the existing ontology [18]. Most of them propose tools dealing with the different causes of change as depicted in figure 1. The major part put the emphasis on the evolution issues [19]. Some articles cope with versioning solutions to

handle different versions of evolved ontologies [20]. Nevertheless not enough attention has been paid to the characterization of an ontology which integrates in its definition the mechanisms to evolve and being versioned.

This paper focuses on a generic architecture make it possible to combine the definition of ontology with evolution and versioning operators. This architecture can be used with any type of ontology based on description logics and especially OWL-DL formalism [21]. An implementation of this architecture is presented at the end of this paper. It is an extension of the Jena's library [22] by override of the existing ontology handling operators.

This paper is articulated in three parts. The first part presents a background on ontology evolution and versioning. The second part describes the VersionGraph Architecture. The last part is an example of evolution versioning based on the Wine Ontology [23].

II. Evolution and Versionning Background

According to [24], ontology lifecycle is divided in seven steps: needs detection, conception, management and planning, evolution, diffusion, use, and evaluation. The needs detection phase starts with a detailed inventory of the domain and the various purposes. Like evolution phase, conception phase needs: knowledge acquisition, shared conceptualization building, formalization (Semantic Web formalisms [25]…) and integration of the existing resources (other ontology, applications…). The permanent phase of management and planning underlines the importance of having a constant monitoring and a global policy to detect or initiate, prepare or evaluate the lifecycle iterations. This work intends to guarantee that an iteration of the lifecycle is activated when an evolution is ready to be completed. The management step requires tools not only to prepare the ontology to adapt the domain changes but also to keep trace of the previous versions of the ontology. These goals can be reached with a versioning system [26]. Diffusion phase deals with the deployment of the ontology. The use phase encloses all the activities related to the access of the ontology. Finally, the evaluation phase aims at evaluating the ontology state. Moreover, like the needs detection phase, it collects beforehand the knowledge of the domain and can also rely on previous studies or feedbacks. Except for the evolution and management phases, all the steps described can be considered as mature domains. Furthermore, this description of the lifecycle shows that evolution and management remains the most complex phases. Evolution is the backbone of the lifecycle iterations. Therefore, the change management process is totally based on it.

The rest of our state of art is articulated in three parts. According to the literature, we will first define the evolution role, operations and process. Then we'll have a look at the existing solutions for change representation and ontology versioning. We will see how to link the evolution process and a versioning system in order to integrate both of them in existing ontologies.

A. Ontology Evolution

As stated by [26], ontology evolution aims at responding to one or several changes in the domain or the conceptualization by applying them on the source ontology. This brief definition looks abstract and leads us to ask: what kind of changes does the evolution apply? How evolution applies them? What are the criteria to respect? How can we manage a good evolution? Evolution changes are defined in the literature and especially in [9] as a succession of simple or complex operations the user wants to apply on the intension (schema) or the extension (data) of the ontology. This evolution aims at adapting the ontology to the changed domain. Applying and propagating the change are often manual tasks but can be done automatically by synchronization with the domain. According to [27] these tasks usually occur during the use phase of the ontology. Ontology Dynamics clearly define the evolution criteria. [28] and [29] qualify the maintenance of the ontology as the most important criterion. Evolution has to maintain whatever relies on the ontology. Maintaining the ontology consistent and pertinent, in a consensus is an inescapable issue of evolution [30]. Applying changes on ontology can turn the conceptualization inconsistent and irrelevant. That's why an evolution should never be validated before the user has a preview of the impact of the changes on the ontology. This impact can only be estimated if the evolution operations are semantically clearly defined.

In order to assure that this process is fully respected, some works propose an approach in six phases.

1. the **change detection** phase consists in detecting what changes occurred in the domain or in the point of view must be propagated to the conceptualization. Lots of papers in the Ontology Dynamics deal with this phase and propose methods and tools like integrated event handlers [27], ontology learning [31] etc…

2. the **representation phase** aims at representing the selected changes with ontological operations. [10] classifies the evolution operations in two types: elementary (atomic) operations and composed (complex) operations. According to [10], elementary operations are simple operations that modify only one entity like addition/suppression of classes/relations, of hierarchy, domain, range links, of class/relation properties like disjoint, transitivity, etc…whereas composed operations are a composition of several elementary operations. The choice of composed operations depends on the granularity of the evolution needs. Therefore, we aim at displaying our proposition to the major part of formal ontologies. So we need to integrate usual operations. Usual operations correspond to operations the ontology developers are the most expected to use when creating and evolving an ontology. In addition to elementary operations, the literature gives some lists of usual operations (e.g. [32,33]). In complement, we have extracted other usual operations like "change the place of an entity"… from the application Protégé. Moreover we make a distinction between operations on the intension and operations on the

extension. The cited works on change operations don't specify specific operations for the instances because they argue that an instance can become a class [10]. However, we maintain that schema operations can't be confounded with instance operations. Actually, it is impossible to create an instance (instance operation) related to a class if this class is not created. Inversely a class can be created (schema operation) without instances.

3. the **semantic phase** helps the user from inconsistency risks by determining the sense of the represented changes. For example, if composed operations have been selected, this phase will allow seeing their decomposition in elementary operations.

4. the **implementation** of the changes alerts the user of the impact on data in terms of data gain or loss. [10] gives these impacts from a list of 22 usual operations (the elementary ones and some composed).

5. the **propagation phase** aims at informing all the dependent parts of the ontology (other ontologies, application) of these changes.

Finally, in sixth step comes the **validation** of the changes.

In the following part we will see how our proposition can integrate these operations in the versioning system and follow these evolution phases.

B. Versioning

This section is articulated in three parts. First we define the role of versioning, bringing our new vision on this definition; Then we describe the versioning process of our versioning system based on the 6 phases of evolution process. A state of art on the existing solutions of change representations will help us to build the tools needed in this process. Finally we present our suggestion to permit the identification and the retrieval of a version of an ontology.

[26] gives in 2007 a very strict definition of the role of versioning : give a transparent access to different existing versions of an ontology by creating a versioning system. This system identifies the versions by their "Id" and delimits their mutual compatibility. In the past three years, Ontology Dynamics proposals extend its role: manage several chronologic and multi temporal versions [34], at local or web level [35], when collected, distributed, accessed by search engines [35]. All these definitions correspond to a retroactive versioning because versions of the ontology have to preexist. However in our objective, we want to integrate a versioning system since the creation of the first version of the ontology. Therefore, we need, as the ontology development, a dynamic and incremental process, which could take into account a new version at each evolution phase. That's why we propose to merge the evolution process (following the 6 phases) with the versioning one.

First, the user chooses the list of operations to apply (cf. change detection phase). The versioning system formalizes them (cf. representation phase), turn them semantically understandable (cf. semantic phase), records and implements them (cf. implementation phase). Then after the propagation of the changes, (cf. propagation phase), the user validates them (cf. validation phase) and the versioning system applies them and generates the new version of the ontology corresponding to an evolution iteration. Finally the versioning system can give a transparent access to both of the versions with criteria defined by the user [36]. It can delimit compatibility by retracing evolution operations [32, 33].

To follow this process, we need to specify the tools displayed by our versioning system. According to [37], a change specification should enclose an operational change specification (our list of operations), then the conceptual relationship between the first version and the new one (the selected operations on the selected entities).The first phase of the evolution process is then completed. The next step is to represent these changes. Several approaches are proposed in the literature to represent changes. Major part of them uses logs. Versioning logs [38] record the different versions of an ontology by representing each entity at a given time. For each class, relation and instance, a new instance of "EvolutionConcept" class is created. [37] argues that metadata should be added to identify this change. In versioning logs, each instance is annotated with metadata (Id, cause, transaction time, state validated or not…). This solution is interesting if the versioning log can be integrated in the ontology. However for our purposes there is no need to represent each entity if it's not modified by the evolution. Evolution logs [39] don't save the versions but act like a change history. Not each entity but each substitution in the ontology is recorded in order to be reused when the user wants to access a version. Tracing the substitution rather corresponds to our objectives as a substitution contains the selected operations and the entities affected. In order to cope with our evolution process we propose to create a Version concept like in the versioning logs integrated in the ontology that will be created at each evolution iteration. This Version concept encloses: 1/ the substitutions operated in the intension or 2/ those operated on the extension and 3/ the metadata. Then, the implementation phase can be helped by introducing event detectors on data. In the application Jena supporting the ontology, the idea is to insert methods using "ActionListener" objects. The propagation phase can be performed by generating events activating the "ActionListener" objects. Finally, the validation is similar to the "Commit" operator of a DBMS, can be done by a simple click by the user. Our incremental versioning process following the 6 evolution phases constitutes the first part of our versioning system.

The second part corresponds to the transparent access definition. The first issue is the identification of the versions. Most of the versioning systems use "Id" of the ontologies to identify them [35]. Though, it's not enough to identify in which version a change on a certain entity occurred. As we have introduced metadata and the list of substitutions occurred when a Version is created, those data

can serve as search criteria to identify and retrieve the right version. We have chosen to extend Jena operators (access on ontology etc…) in order to take into account the search criteria. This extension can be performed by an override of the access methods. For example, by adding metadata and operation attributes.

This state of art permitted us to build the evolution and versioning process of our proposition. We also managed to design the versioning tools in order to represent changes and access the ontology.

III. VERSIONGRAPH ARCHITECTURE

In this section, we present the VersionGraph architecture which implements the choices of our state of art. First, we focuses on the operations corresponding to the evolution operations. Then we describe our versioning system. Finally, we give an example of evolution on the Wine ontology.

A. Evolution Operations

The schema and instance operations are differentiated respectively by `SchemaOperation` and `InstanceOperation`. `SchemaOperation` type operations correspond to the creation and deletion of classes (`AddClass`) and properties (`AddProperty`) but also to additions and deletions of restrictions on them. We distinguish restrictions on the classes and properties or properties of the data link hierarchy (`HierarchyLink`) such as class / subclass, property / sub-property. Also in the class restrictions, limitations like classes / properties such as the relationship between properties and classes (`ClassPropertyLink`, `ClassDataPropertyLink`), cardinality (`ClassPropertyCardinality`) are classified. Also in the restrictions we find domain and range restrictions of attributes (`PropertyAttributeLink`). Finally `TypeProperty` operations are used to define a specific constraint of a property (transitive, symmetric etc ...).

`InstanceOperation` type operations, correspond to operations of addition and deletion of individuals and statements about these individuals. We distinguish between the assertions relying individuals to the values (`DataPropertyAssertion`) and those specifying the types for these individuals (`ObjectPropertyAssertion`).

B. From evolution to versioning

From these evolution operations and the study of the different versioning solutions of our state of art, we derived a versioning system. At each evolution of the ontology, the system stores in the ontology, the changes impacted by the operations used and the context. This versioning system is an independent ontology which intends to be integrated into the existing ontology by a simple addition operation. Then, the user can start a first evolution of ontology in choosing whether to change the schema (intension) or data (extension) using the above operations. Each list of changes chosen by the user during the evolution is kept using a concept `SchemaVersionGraph` for `SchemaOperation` operations and `InstanceVersionGraph` for `InstanceOperation` operations on instances by specifying which elements of the ontology are concerned (concepts, relationships...). Contextual information can be added (as version, date, author, description...). These data are traced during the evolution using a concept of context `VersionContext`.

The set containing `SchemaVersionGraph` or `InstanceversionGraph` and `VersionContext` is called `VersionGraph`. Figure 2 depicts an overview of the ontology schema. For more clarity, it only shows concepts and their relationships under 6th hierarchical degrees. In a transparent way, each application of changes made by the user generates a new `VersionGraph`. The `VersionGraph` definition in Protégé is presented in Figure 3. As depicted in this figure a `VersionGraph` contains a link with the previous version of the ontology (`hasPreviousVersionGraph`). It's actually a link to the core ontology (for the first `VersionGraph`) or to the previous `VersionGraph`. Because of its nature, our system of evolution and versioning can be integrated into applications using ontologies Jena. The access operations of the library Jena can be overridden by the criteria of change and context.

Until now, proposals for versioning are often accompanied by a specific application that the user must install to access the version it wants if the use of URI is not enough (Evolva). However, many ontologies are accessed using a Java API Jena. Indeed, this library supports ontology-based formalisms like RDF, RDFS, OWL and the various DAML + OIL. Jena contains all the methods to access and edit ontologies. In addition, it also implements all the basic operations of evolution and the commonly used composed ones. Overridden access methods are able to take into account the criteria of versions thanks to new attributes. These criteria are integrated into the ontology itself as we saw in the previous paragraph.

C. The Wine Ontology Versionning

The Wine ontology is an ontology example in which international wines are described. For the first step, we import the VersionGraph ontology into the Wine ontology by an addition operation. Then the system creates the first version of the wine ontology with a first instance of `VersionGraph`. This Versiongraph only has a link with the source ontology.

```
VersionGraph0
        hasPreviousVersiongraph
        →ontology:Wine
```

Then we want to add the "Straw Wine" wine which doesn't exists in the Wine ontology. Straw Wine's fruit is selected then dried in the sun so that the juice is very concentrated in flavor and sugar. So it is a dessert style wine sometimes heavy or balanced or straw gold color. It can be made from red grapes Cabernet Franc and Cabernet Sauvignon or Chardonnay white grapes and Sauvignon Blanc. To add this new concept and describe it, the system creates another `VersionGraph`. This new one is linked with the previous one. The system specifies a SchemaVersionGraph which contains the operations needed to describe and add the concept in the ontology.

```
Versiongraph1
 haspreviousVersionGraph
VersionGraph:VersionGraph0
 hasDate → date:11/05/2010
 has Author → name:Perrine_PITTET
 hasSchemaVersiongraph
 →SchemaVersionGraph:SchemaVersionGraph1

SchemaVersionGraph1
 hasAddClass → class:StrawWine
 hasAddClassHierarchyLink →class:DessertWine
 →subclass:StrawWine
 hasAddClassDataPropertyLink:
 →class:StrawWine
 →dataproperty:hasColor value:Golden
 hasAddClassDataPropertycardinality:
 →class:StrawWine
 →dataproperty:hasBody
 →value:Full,Moderate
 →cardinality:only
 hasAddClassDataPropertycardinality:
 →class:StrawWine
 →dataproperty:madeFromGrape
 →value:(CabernetSauvignon and
 Carbernetfranc) or (Chardonnay and
 SauvignonBlanc) →cardinality:only
```

Then, we want to add an individual of Straw Wine type: "Vin Paillé de Corrèze". First, we need to validate the previous changes by a "Commit". Then changes in the schema are recorded and the new schema version is propagated to the ontology. A third `VersionGraph` is generated for the addition of the individual. This time it contains an `InstanceVersionGraph`.

```
VersionGraph2
 haspreviousVersionGraph
 →versiongraph:VersionGraph1
 hasDate →date:11/05/2010
 hasAuthor →name:Perrine PITTET
 hasInstanceVersiongraph
 →InstanceVersiongraph:InstanceVersionGraph1

InstanceVersionGraph1
 hasAddIndividual →individual:VinPaillé
 hasAddMemberClass →individual:VinPaillé
 →class:StrawWine
 hasAddObjectPropertyAssertion
 →individual:VinPaillé →property:locatedIn
 →value:FrenchRegion
```

IV. Conclusion

Ontology evolution and versioning are recent domains of search. Most of current ontology versioning approaches are not based on the evolution process. Rare are the solutions which integrate these mechanisms since the creation of the ontology. Our proposed architecture Versiongraph is a semantic solution towards the characterization of a dynamic ontology which reaches these objectives. Our ongoing research shows preliminary results on evolution of several ontologies like Wine, FOAF or Pizza. Our short coming plan is to enhance our evolution and versioning process on several projects applied to online press comments, tourism and town heritage ontologies.

References

1. Gruber, T.,R. - A translation approach to portable ontologies- Knowledge Acquisition ,1993.
2. Moguillansky, M., O., Rotstein, N., D., Falappa, M., A. - A Theoretical Model to Handle Ontology Debugging & Change Through Argumentation - Proceedings of the International Workshop on Ontology Dynamics (IWOD 2008) at ESWC 2008, Karlsruhe, Germany. 2008.
3. Klein, M. and Fensel, D. - Ontology Versioning on the Semantic We- . s.l. : SWWS Standford, 2001, SWWS Stanford.
5. Ventrone, V. and Heiler, S. - Semantic Heterogeneity as a Result of Domain Evaluation. 1991, SIGMOD Record Special Issue: Semantic issues in Multidatabase Systems.
6. Klein, M. and Noy, N. - A Component-Based Framework for Ontology Evolution., F. 2003. IJCAI-03 Workshop on Ontologies and Distributed Systems, CEUR-WS, vol. 71.
7. Calvanese, D., De Giacomo, G. & Lenzerini, M. - A Framework for Ontology Integration - in Cruz, I., Decker, 2002.
8. Pinto, H.S., Gomez-Perez, A. & Martins, J. P. - Some Issues on Ontology Integration - Proceedings of the Workshop on Ontologies and Problem-Solving Methods (KRR5) at 16th International Joint Conference on Artificial Intelligence (IJCAI-99).
9. Avesani, P., Giunchiglia, F. and Yatskevich, M. - A Large Scale Taxonomy Mapping Evaluation - s.l. : Lecture Notes in Computer Science, Springer, 2005, Vol. Volume 3729. 67-81.
10. Noy, N. F., Klein, M. - Ontology Evolution: Not the Same as Schema Evolution - Stanford Medical Informatics, Stanford University, Stanford, CA, USA Vrije University Amsterdam, Amsterdam, The Netherlands.
11. Haase, P. & Qi, G. 2007. - An Analysis of Approaches to Resolving Inconsistencies in DL-based Ontologies - Proceedings of the International Workshop on Ontology Dynamics (IWOD-07), pp. 97-109.
12. Kalfoglou, Y. & Schorlemmer, M. - Ontology Mapping: the State of the Art- Knowledge Engineering Review,18 (1), pp. 1-31 2003.
13. Hu, W. & Qu, Y. - Block Matching for Ontologies- Proceedings of the 5th International Semantic Web Conference (ISWC-06), pp. 300-313.
14. Flouris, G. & Plexousakis, D. - Handling Ontology Change: Survey & Proposal for a Future Research Direction - Technical Report FORTH-ICS/TR-362. 2005.
15. Baader, F., & al.-The Desc. Logic Handbook : Theory, Implementation & Applications - Cambridge University Press, pp. 495-505, 2003.
16. Ontology Dynamics : http://www.ontologydynamics.org/od/
17. Djedidi, R., Aufaure, M., A. - Change Management Patterns for Ontology Evolution Process - Proceedings of the Int. Workshop on Ontology Dynamics at ESWC 2008, Karlsruhe, Germany. 2008.
18. Ribeiro, M., M., Wassermann, R., Antoniou, G., Flouris, G., Pan, J. E.- Belief Contraction in Web-Ontology Languages - Proceedings of the

International Workshop on Ontology Dynamics (IWOD 2008) at ESWC 2008, Karlsruhe, Germany. 2008.
19. Pan, J., Z. - A Stratification-based Approach for Inconsistency Handling in Description Logics, Proceedings of the International Workshop on Ontology Dynamics at ESWC 2007, Innsbruck, Austria. 2007.
20. Allocca, C., D'Aquin, M., Motta, E.- Detecting Different Versions of Ontologies in Large Ontology Repositories – Proc. of the International Workshop on Ontology Dynamics, Karlsruhe, Germany. 2008.
21. OWL : http://www.w3.org/TR/owl-features/
22. Jena: http://jena.sourceforge.net/
23. Wine Ontology : http://www.w3.org/TR/owl-guide/wine.rdf
24. Hodgson, R.- The Potential of Semantic Technologies for e-government- presentation of eGov Open Source Conference- Washington, DC, March 18th, 2003
25. Semantic Web: http://semanticweb.org/wiki/Main_Page
26. Flouris, F., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G. - Ontology Change: Classification & Survey - The Knowledge Engineering Review, 1–29, 2007, Cambridge University Press
27. Tovar, E., Vidal, M., E. - REACTIVE: A Rule-based Framework to Process Reactivity - Proceedings of the International Workshop on Ontology Dynamics at ESWC 2008, Karlsruhe, Germany. 2008.
28. Atle Gulla, J. and Sugumaran, V. - An Ontology Creation Methodology: A Phased Approach.. Karlsruhe, Germany : s.n., 2008. Proc. of the International Workshop on Ontology Dynamics at ISWC 2008.
29. Dividino, R. and Sonntag, D. - Controlled Ontology Evolution through Semiotic-based Ontology Evaluation. Karlsruhe, Germany : s.n., 2008. International Workshop on Ontology Dynamics at ISWC 2008.
30. Zablith, F., et al. - Using Background Knowledge for Ontology Evolution, Int. Work. on Ontology Dynamics, Karlsruhe, Germany 2008.
31. Novacek, V., Laera, L. and Handschuh, S. - Semi-automatic Integration of Learned Ontologies into a Collaborative Framework.
32. Stojanovic, L., et al. User-driven Ontology Evolution Management. 13th Int. Conf. on Knowledge Engineering and Knowledge Management. 2002.
33. Stuckenschmidt, H. and Klein, M. - Integrity and Change in Modular Ontologies. 18th International Conference on Artificial Intelligence, 2003.
34. Grandi, F. - Multi-temporal RDF Ontology Versioning. Karlsruhe, Germany, International Workshop on Ontology Dynamics at ISWC 2008.
35. Allocca, C., D'Aquin, M. and Motta, E. - Detecting Different Versions of Ontologies in Large Ontology Repositories.
36. Stuckenschmidt, H. and Klein, M. - Integrity and Change in Modular Ontologies, 18th Int. Joint Conference on Artificial Intelligence, 2003.
37. Klein, M. and Fensel, D. - Ontology Versioning on the Semantic Web. SWWS Standford, 2001
38. Yildiz, B. - Ontology Versioning and Evolution, Asgaard, 2006
39. Liang, Y. - Ontology Versioning and Evolution For Semantic Web-Based Applications. 2005.

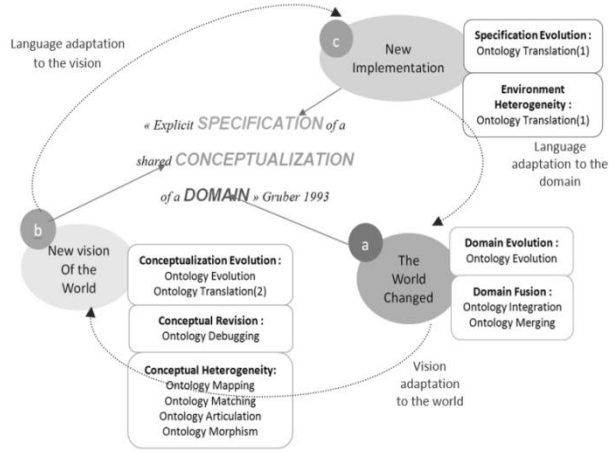

Figure 1. Causes of changes in the lifecycle of an ontology.

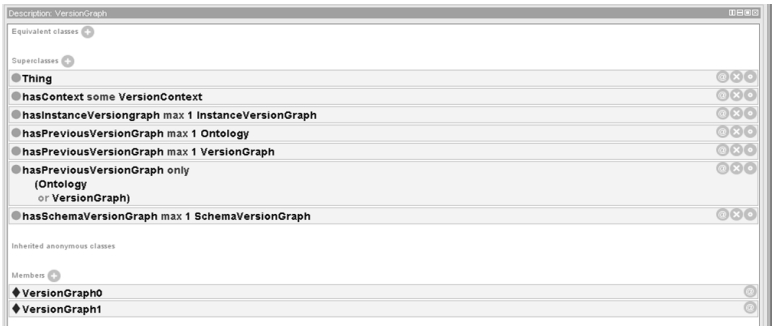

Figure 3. VersionGraph definition in Protege.

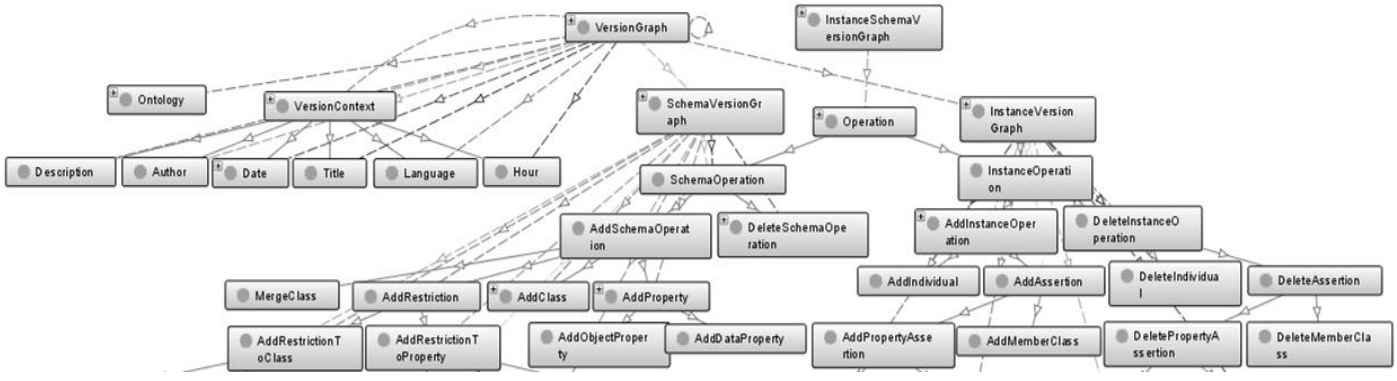

Figure 2. Overview of the VersionGraph Ontology