Consistent Ontologies Evolution Using Graph Grammars

Mariem Mahfoudh, Germain Forestier, Laurent Thiry, and Michel Hassenforder

12 rue des Frères Lumière F-68093 Mulhouse Cedex
{mariem.mahfoudh,germain.forestier,laurent.thiry,michel.hassenforder}@uha.fr

Abstract. Ontologies are often used for the meta-modelling of dynamic domains, therefore it is essential to represent and manage their changes and to adapt them to new requirements. Due to changes, an ontology may become invalid and non-interpretable. This paper proposes the use of the graph grammars to formalize and manage ontologies evolution. The objective is to present an a priori approach of inconsistencies resolutions to adapt the ontologies and preserve their consistency. A framework composed of different graph rewriting rules is proposed and presented using the AGG (Algebraic Graph Grammar) tool. As an application, the article considers the EventCCAlps ontology developed within the CCAlps European project.

Keywords: ontologies, graph grammars, evolution, rewriting, ontology changes, category theory, AGG.

1 Introduction

Designed as a response for vocabulary heterogeneity problems and semantic ambiguities of data sources, ontologies play the role of a semantic structure that represents and formalizes human knowledge of a specific domain. As they are often used for meta-modelling of dynamic domains, they constantly require to adapt to knowledge evolution. However, this evolution presents several challenges, especially in the changes definition and consistency preservation of the modified ontology. In fact, a misapplication of a change can alter the consistency of an ontology by affecting its structure and/or its semantic. This promotes the need to formalize the process of evolution.

This work proposes the use of graph grammars, based on algebraic approaches to express and manage ontologies evolution. Graph grammars are a rigorous formal method, allowing the verification of the feasibility of ontology changes. Thanks to their application conditions, they avoid the execution of changes that do not satisfy a set of constraints. They also offer many tools such as AGG (Algebraic Graph Grammar) which provides a simple environment for defining rewriting rules, helping the user to easily express his needs. Thus, the main

This is the author's version of an article published at KSEM 2013. The final authenticated version is available online at: https://doi.org/10.1007/978-3-642-39787-5_6

objective of this work is to present a formal method for managing ontology changes and ensuring the consistency of the modified ontology.

This paper is organized as follows: section 2 presents the ontologies changes and the graph grammars. Section 3 proposes the formalization of ontology changes with graph grammars. Section 4 presents an application using the ontology EC-CAlps which has been developed in the context of the European project CCAlps. Section 5 shows some related works. Finally, a conclusion summarizes the presented work and gives some perspectives.

2 Ontology changes and Graph Grammars

2.1 Ontology changes

Ontologies are commonly defined as a "specification of a conceptualization" [1]. They are composed by a set of classes, properties, individuals and axioms and they often need to evolve to integrate and reuse knowledge. Different classifications of ontology changes have been proposed [2,3]. One of the most widely used [4] distinguishes two types:

1. *Elementary/basic changes*: represent primitive operations which affect a single ontology entity, e.g. addition, deletion and modification;
2. *Composite/complex changes*: are composed of multiple basic operations that together constitute a logical entity, e.g. merge or split of classes.

Whatever its nature (basic or complex), an ontology change should be formalized in order to properly identify its inputs, its outputs and the inconsistencies that it is likely to generate. In this work, the ontology is considered as a graph G = (V, E) where V is a set of vertices which represent classes, individuals, etc. E is a set of edges which represent axioms. Thus, an ontology change can be expressed and formalized as a graph rewriting rule $r : G \rightarrow G'$.

2.2 Graph grammars

Definition 1 (Graph Grammars). A graph grammar (GG) is a pair composed of an initial graph (G) called host graph and a set of production rules (P) also called graph rewriting rules or graph transformation.
A production rule P = (LHS, RHS) is defined by a pair of graphs:

- LHS (Left Hand Side) presents the precondition of the rule and describes the structure that has to be found in G.
- RHS (Right Hand Side) presents the postcondition of the rule and should replace LHS in G.

Graph grammars can be typed (TGG) and is defined as: $TGG = (G_T, GG)$ where $G_T = (V_T, E_T)$ is a type graph which represents the type information (type of nodes and edges). The typing of a graph G over G_T is given by a total graph morphism $t : G \rightarrow G_T$ where $t : E \rightarrow E_T$ and $t : V \rightarrow V_T$.

The graph transformation defines how a graph G can be transformed to a new graph G'. More precisely, there must exist a morphism (m) that replaces LHS by RHS to obtain G'.
There are different graph transformation approaches to apply this replacement, as described in [5]. The algebraic approach [6] is based on category theory with the *pushout* concept.
Definition 2 (Category Theory). A category is a structure consisting of:

1. a collection of objects O;
2. a set of morphisms M and a function $s : M \to O \times O$, $s(f) = (A, B)$ is noted $f : A \to B$;
3. a binary operation, called composition of morphisms $(\circ) : M \times M \to M$;
4. an identity morphism for each object $id : O \to O$.

The composition of $f : A \to B$ and $g : B \to C$ is associative and is written $g \circ f : A \to C$.
Definition 3 (Pushout). Given three objects A, B and C and two morphisms $f : A \to B$ and $g : A \to C$. The pushout of B and C consists of an object D and two morphisms $m_1 : B \to D$ and $m_2 : C \to D$ where $m_1 \circ f = m_2 \circ g$.

The algebraic approach is divided into two sub-approaches: the *Single pushout SPO* [7] and the *Double poushout DPO* [8]. In this work, only the SPO approach was considered as it is more general (e.g. without the gluing condition) and sufficient to represent the different ontology changes. Therefore, applying a rewriting rule (r) to an initial graph with the SPO method, consists in (Figure 1):

1. Find LHS in G using a morphism m : $LHS \to G$.
2. Delete $LHS - (LHS \cap RHS)$ from G.
3. Add $RHS - (LHS \cap RHS)$ to G. This operation is done by the construction of a pushout and gives a new version G' of G.

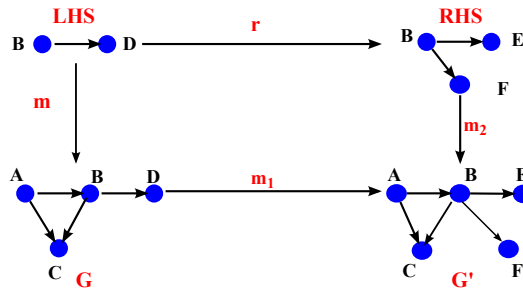

Fig. 1. Application of a rewriting rule graphs with the SPO approach.

AGG tool Several tools have been proposed to support graph rewriting as AGG [9], Fujaba [10] or Viatra [11]. The AGG tool is considered as one of the most important tools. It supports the algebraic approach and typed attributed graphs. The AGG environment provides graphical editors for graphs and rules. It also allows to add the NACs (Negative Application Condition) which specifies a subgraph that may not occur when matching a rule. It is important to note that the internal graph transformation engine can also be used by a Java API and thus, be integrated into other custom applications.

3 Formalisation of ontology changes

This section introduces the definition and formalisation of ontology changes using typed graph grammars. The first step consists in creating the type graph which presents the meta-model of the ontology. The next step defines the ontology changes under the form of graph rewriting rules $(r_1, r_2, ...r_n)$.

3.1 Type graph

In this article, OWL was chosen to describe ontologies since it is the standard proposed by the W3C, and the language usually adopted to represent ontologies. However, other languages can be considered by using converters[1].

Figure 2 shows the representation of OWL meta-model with AGG (G_T). The OWL meta-model [12] defines the basic conceptual primitives of OWL which are classes, properties (`ObjectProperty` and `DataProperty`), individuals, axioms (`disjointWith`, `equivalentTo`, etc.). The classes model the set of individuals and it can be primitive or complex (`UnionClass`, `ComplementClass`, `IntersectionClass`). The `ObjectProperty` models the relationship between classes (`Domain` and `Range`) whereas the `DataProperty` link a class (`Domain`) to a `Datatype`. All these primitives are represented as nodes and each of them have two attributes inherited from the class `Entity`. The attribute `name` specifies the name of the local entity, while the attribute `iri` (Internationalized Resource Identifier) allows to identify and to reference them. The G_T also defines the restrictions which are a particular type of class description. There are two types: restriction values (`AllValuesFrom`, `SomeValuesFrom`, `HasValue`) and cardinality (`CardinalityRestriction`). Axioms are represented as edges expressing the relationships between classes, properties and individuals. For example, the edge `disjointWith` represents the disjunction between two classes or two properties.

3.2 Ontology changes with graph grammars

Adapting an ontology to new requirements consists in modifying its structure. However, these changes can cause inconsistencies which require the application of derived changes to correct them. This section describes how consistently express

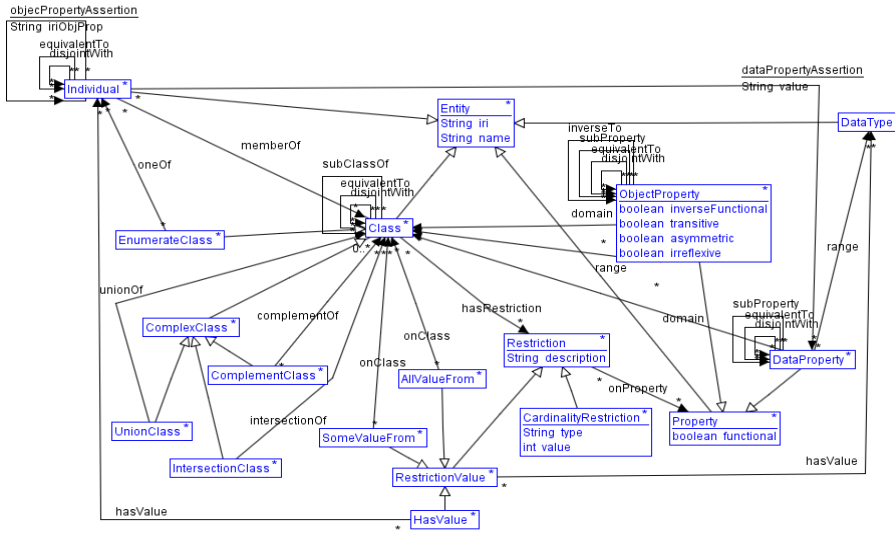

Fig. 2. Type Graph used for graph rewriting.

some ontology changes using graph grammars. In this paper, only elementary changes were considered (Figure 3).

Definition 4 (Ontology changes). An ontology change is formalized by 5-tuplet CH = (Name, NAC, LHS, RHS, CHD) where:

1. Name specifies the type of change;
2. NAC defines the condition which must not be true to apply the rewriting rule;
3. LHS presents the precondition of the rewriting rule;
4. RHS defines the postcondition of the rewriting rule;
5. CHD presents the derived changes. They are additional operations that could be attached to CH to correct its inconsistencies.

Inconsistencies addressed in this work are:

- Data redundancy can be generated following an add or rename operation. This type of inconsistency is corrected by the NACs.
- Isolated node, a node N_x called isolated if $\forall N_i \in N,\ \nexists V_i \in V | V_i = (N_x, N_i)$. This incoherence requires to link the isolated node to the rest of the graph. Depending of the type of node, derived changes are proposed.
- Orphan individual is an inconsistency which is generated as a result of removal of classes containing individuals.
- Axioms contradiction, the addition of a new axiom should not be accept if it contradicts an axiom already defined in the ontology. Such verification is necessary to maintain the semantics of the evolved ontology.

[1] owl.cs.manchester.ac.uk/converter

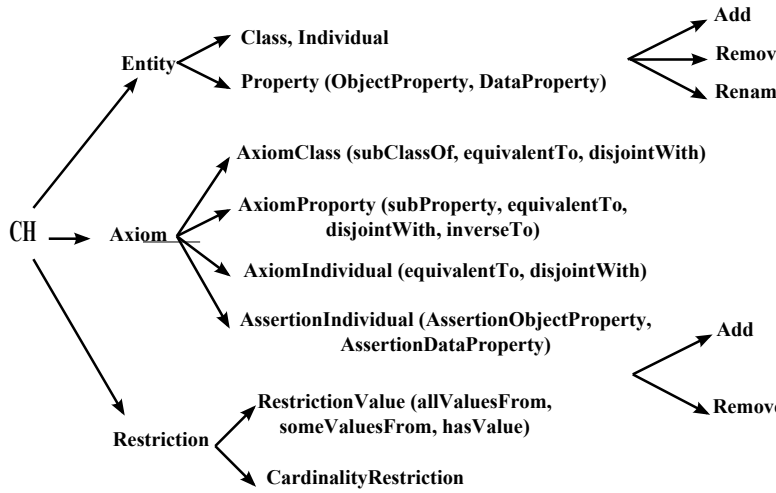

Fig. 3. Elementary changes.

Thus, the *RenameObjectProperty(OBIRI, OBIRINew)* change consists in the renaming of a node `ObjectProperty` (OB). Then, three graphs should be created: 1) the LHS consisting of a node `OB` where its attribute `iri` is equal to `OBIRI`; 2) the RHS consisting of a node `OB` where its attribute `iri` is equal to `OBIRINew`; 3) the NAC is equal to RHS to prevent the redundancy (Figure 4).

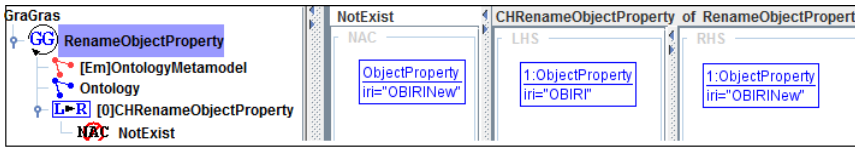

Fig. 4. Rewriting rules of the RenameObjectProperty change.

The *AddClass(Cnew)* change allows the add of a new node of type `Class` in the host graph G ($Cnew \in G \wedge t : Cnew \rightarrow V_{Class}$). The rewriting rule consists of three graphs: 1) $LHS = \emptyset$; 2) $RHS = Cnew$; 3) $NAC = RHS = Cnew$; the NAC should be equal to RHS to prevent data redundancy. Besides, a node should not be isolated. To attach a node of type `Class` to the graph, two types of correction can be applied: `AddObjectProperty` or `AddAxiom`. The first one consists in adding a new property where the node *Cnew* is one of its member. The second inserts a new axiom to link *Cnew* to an existing property (`addDomain`, `addRange`) or connect it to another node of type `Class` applying the changes `AddEquivalentClass`, `AddDisjointClass`, `AddSubClass`, etc.

Figure 5 shows the rewriting rules of the `AddClass` change followed by some derived changes. They are classified by layers to define the sequence of their application: the user can select by a simple activation the derived changes which he wishes to apply.

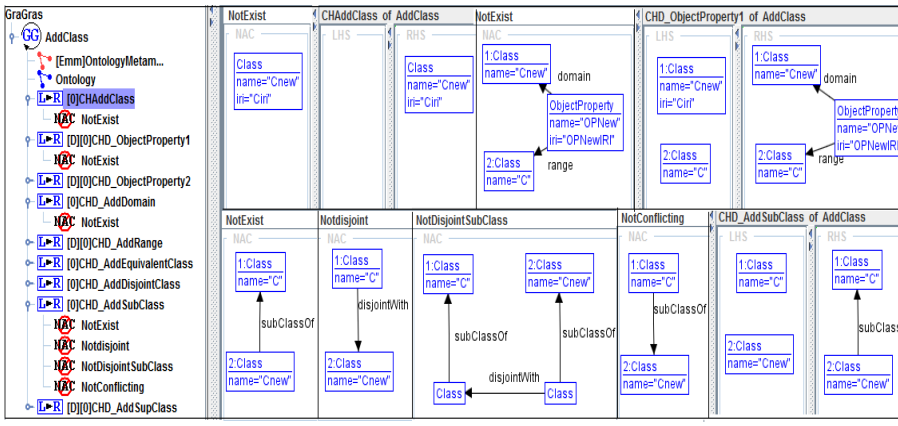

Fig. 5. Rewitting rules of the AddClass change.

The *AddDisjointClass (C1, C2)* change adds a disjunction axiom between two nodes of type `Class` (see Figure 6). Thus, three NACs are defined to verify the absence of edges of the type: 1) `disjointWith` to avoid redundancy; 2) `equivalentTo`, two classes can not be disjoint and equivalent at the same time; 3) `subClassOf`, two classes what share a subsumption relation can not be disjoints.

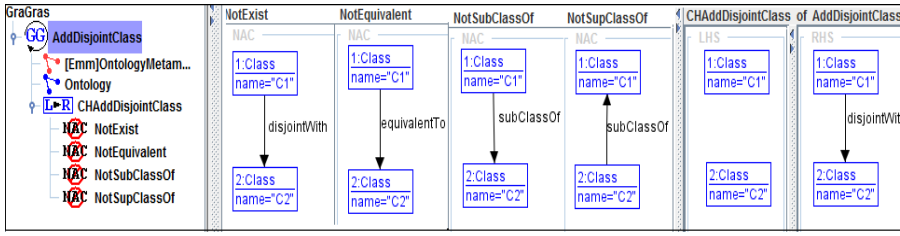

Fig. 6. Rewriting rule of AddDisjointClass change.

The *RemoveClass(C)* change. The application of this type of change may cause some inconsistencies such as the existence of orphans individuals or the lack of restriction members. Thus, before deleting a node, all its dependencies (its axioms) should be checked to propose correction alternatives. Indeed, the

restriction should be deleted whereas the processing of individuals goes through different steps illustrated in Figure 7. Then, before deleting a class C defining individuals ($I\ memberOf\ C$), it should check: 1) If $C\ subClassOf\ C_p \wedge \forall C_i\ subClassOf\ C_p \wedge !disjointWith\ C$. Then, $I\ memberOf\ C_p$; 2) Else If $\exists C_i \in G$ where $C_i\ equivalentTo\ C$. Then, $I\ memberOf\ C_i$; 3) Else if $\exists I_i \in G$ where $I_i\ memberOf\ C_i \wedge I_i\ equivalentTo\ I$. Then, $I\ memberOf\ C_i$; 4) If none of these cases is satisfied, the orphans individuals will be deleted from G.

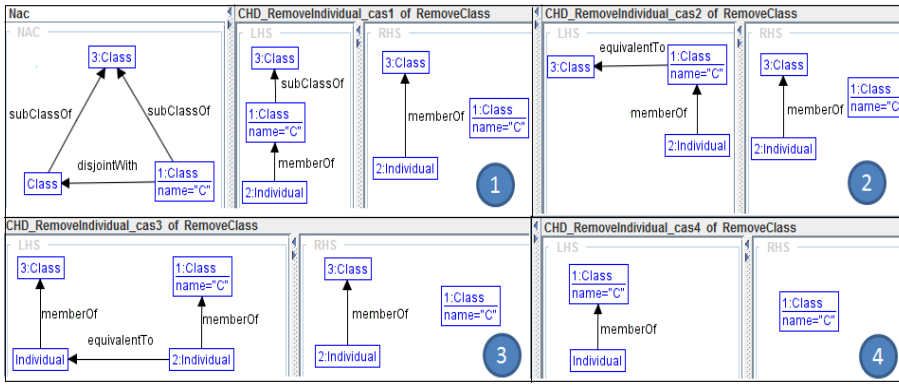

Fig. 7. Processing of RemoveIndividual of the RemoveClass change.

4 Application

This work was developed in the frame of the CCAlps European project[2] which aims at providing an infrastructure to facilitate the collaboration between the creative industries and regions.

In this context, four OWL ontologies have been proposed: EventCCAlps, HubCCAlps, CompanyCCAlps and RegionCCAlps. The EventCCAlps ontology defines the concepts of the events. It presents the characteristics of an event (description, location, time, etc.) and its different relationships with other concepts (Company, Region, Hub, etc.). EventCCalps is based on the Event Ontology [13] and the Linking Open Descriptions of Events [14].

As an example of transformation, this section presents the deletion of the `Employee` class. In order to start the process of transformation and apply the rewriting rules, the ontology should be converted into an AGG graph. Indeed, two programs (*OWLToGraph* and *GraphToOWL*) have been developed to automate the transformation of OWL to AGG and vice versa. They are based on the AGG API and Jena library[3], an open source API to read and manipulate ontologies

[2] www.ccalps.eu, the project reference number is 15-3-1-IT

[3] jena.sourceforge.net

described in OWL, RDF and RDFS.
Figure 8 shows a result of the transformation of EventCCalps. Note that for reasons of readability the IRI have been removed from the figure.

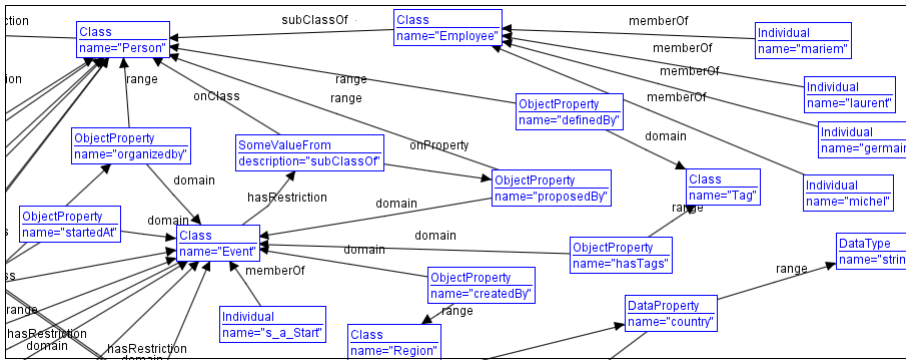

Fig. 8. An extract from EventCCAlps ontology after transformation to AGG graph.

The `Employee` class has different individuals: `mariem`, `laurent`, `germain` and `michel`. It is a subClassOf the `Person` class and there is no class in the ontology which inherits from `Person` class and it is, in the same time, disjoint with the `Employee` class. Then, the RemoveClass change invokes the derived change RemoveIndividual and attaches the individuals to the `Person` class. In this way, the individuals and the knowledge can be saved without affecting the consistency of the ontology. Figure 9 shows the definition of this change with AGG and the Figure 10 presents the result of the transformation. This simple example illustrates how the presented work could be used to manage ontology evolution.

5 Related work

Ontologies evolution is often confused with the filed of database schemas evolution. In fact, many issues in ontology evolution are exactly the same as the issues in schema evolution. However, there are several differences between them. Instead of comparing directly the process of evolution, ontologies and database schemas evolutions are generally compared through an analyses of the differences between the ontology and the database schemas.

Noy and al. [15] have summarized this difference by the following points: 1) Ontologies themselves are data to the extent to which database schemas have never been. So, ontologies evolution must consider both the structure and instances of the ontologies; 2) Ontologies themselves incorporate semantics while database schemas do not provide explicit semantics for their data. Then, the restrictions must be considered in the ontology evolution process; 3) Ontologies

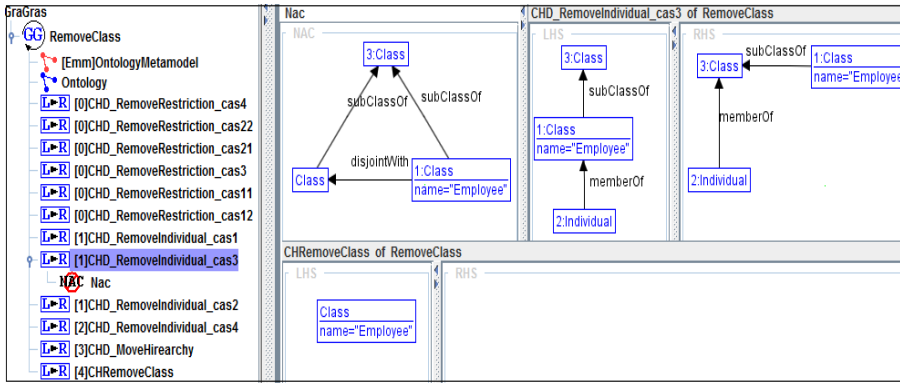

Fig. 9. Rewriting rules of deleting "Employee" class from EventCCAlps ontology.

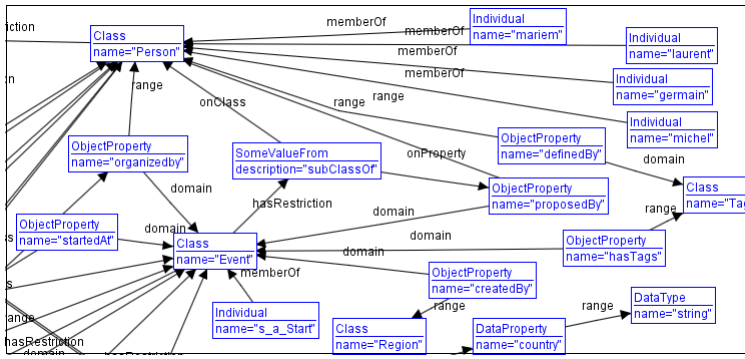

Fig. 10. EventCCAlps ontology after delete the "Employee" class.

are decentralized by nature so their content and usage are often more complex than a database schemas.
Ontologies evolution has been influenced by the research on schema evolution database [16] but it is a different area of research having its own characteristics.

The first proposed methods in the literature [17,4,18] have presented ontologies evolution process in general but they are considered as the basis of most current works. Thus, Hartung and al. [19] have studied the evolution and the difference between two versions of the same ontology. This work provided a *COnto-Diff* tool which can detect different basic changes, however, it has not presented any processing for inconsistencies. Khattak and al. [20] and Luong and al. [21] have proposed posteriori approaches to solve inconsistencies. This type of approach, unlike the a priori process that we propose, requires the implementation of changes to verify the alteration of the ontology and then cancel them if something went wrong. This causes a waste of time and resources. Dragoni

and al. [22] have also addressed the impact of the ontologies evolution. They consider the ontology as a hierarchy of concepts and they ignore the conceptual and semantic relation which it models. Then, the proposed correction for monitoring changes have addressed only the subsumption relation. An interesting work has been presented in [23] which is based on pi-calculus. It manages the ontology changes with a formal method and it proposed some rules for preserving ontologies consistency.

The graph grammars allow the definition, formalization and application of ontology changes. Their ability to avoid the inconsistencies is the most important characteristics. It allows, due to application conditions, to verify the validity of each type of change and its effects on the graph.

6 Conclusion and future work

In this paper, we presented the use of the graph grammars to formalize and implement the ontology changes. We proposed an a priori approach of inconsistencies resolutions to adapt ontologies and preserve their consistency. The use of AGG allowed a simple definition of rewriting rules and it presented many advantages. Two programs were developed *OWLToGraph* and *GraphToOWL* to automate the back and forth process of transformation of the ontologies to graphs. They allow the user to work and avail the benefits of graph grammars even if his ontologies are defined by another representation language.

Many perspectives can be identified. Firstly, it is important to extend the study for the complex ontology changes. It would also be interesting to exploit ontology changes to define a formal approach of ontologies composition knowing that the composition is a combination of some basic changes (`AddClass`, `removeClass`, `AddAxiom`, `RemoveAxiom`, etc.). Integration of a query language (e.g. SPARQL) is envisaged in order to optimize the selection of ontologies entities.

References

1. Gruber, T.R., et al.: A translation approach to portable ontology specifications. Knowledge acquisition **5**(2) (1993) 199–220
2. Stojanovic, L.: Methods and Tools for Ontology Evolution. PhD thesis, University of Karlsruhe, Germany (2004)
3. Qin, L., Atluri, V.: Semdiff: An approach to detecting semantic changes to ontologies. International Journal on Semantic Web and Information Systems (IJSWIS) **2**(4) (2006) 1–32
4. Klein, M.: Change Management for Distributed Ontologies. PhD thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands (2004)
5. Rozenberg, G.: Handbook of graph grammars and computing by graph transformation. Volume 1. World Scientific (1999)
6. Ehrig, H., Pfender, M., Schneider, H.J.: Graph-grammars: An algebraic approach. In: Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on, IEEE (1973) 167–180

7. Löwe, M.: Algebraic approach to single-pushout graph transformation. Theoretical Computer Science **109**(1) (1993) 181–224
8. Ehrig, H.: Introduction to the algebraic theory of graph grammars (a survey). In: Graph-Grammars and Their Application to Computer Science and Biology, Springer (1979) 1–69
9. Ermel., C., Rudolf., M., Taentzer, G.: The agg approach: Language and environment. In: Handbook of graph grammars and computing by graph transformation, World Scientific Publishing Co., Inc. (1999) 551–603
10. Nickel, U., Niere, J., Zündorf, A.: The fujaba environment. In: Proceedings of the 22nd international conference on Software engineering, ACM (2000) 742–745
11. Varró, D., Pataricza, A.: Generic and meta-transformations for model transformation engineering. In: UML 2004-The Unified Modeling Language. Modelling Languages and Applications. Springer (2004) 290–304
12. Object Management Group: Ontology definition metamodel (omg) version 1.0. Technical report, Object Management Group (2009)
13. Raimond, Y., Abdallah, S.: The event ontology. Technical report, Technical report, 2007. http://motools.sourceforge.net/event (2007)
14. Shaw, R., Troncy, R., Hardman, L.: Lode: Linking open descriptions of events. The Semantic Web (2009) 153–167
15. Noy, N.F., Klein, M.: Ontology evolution: Not the same as schema evolution. Knowledge and information systems **6**(4) (2004) 428–440
16. Rahm, E., Bernstein, P.A.: An online bibliography on schema evolution. ACM SIGMOD Record **35**(4) (2006) 30–31
17. Stojanovic, N., Stojanovic, L., Handschuh, S.: Evolution in the ontology-based knowledge management system. In: Proceedings of the European Conference on Information Systems-ECIS. (2002)
18. Rogozan, D., Paquette, G.: Managing ontology changes on the semantic web. In: Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on, IEEE (2005) 430–433
19. Hartung, M., Groß, A., Rahm, E.: Conto-diff: Generation of complex evolution mappings for life science ontologies. J Biomed Inform (1) (2013) 15–32
20. Khattak, A.M., Latif, K., Lee, S.: Change management in evolving web ontologies. Knowledge-Based Systems **37**(0) (2013) 1–18
21. Luong, P.H., Dieng-Kuntz, R.: A rule-based approach for semantic annotation evolution. Computational Intelligence **23**(3) (2007) 320–338
22. Dragoni, M., Ghidini, C.: Evaluating the impact of ontology evolution patterns on the effectiveness of resources retrieval. In: 2nd Joint Workshop on Knowledge Evolution and Ontology Dynamics EvoDyn 2012. (2012)
23. Wang, M., Jin, L., Liu, L.: A description method of ontology change management using pi-calculus. Knowledge Science, Engineering and Management (2006) 477–489