Ontology Modification in a Multi-User Concept
Conflict Resolution

Fatma Chamekh, Guilaine Talens and Danielle Boulanger
Magellan - IAE- Jean Moulin University, 6 cours Albert Thomas – BP 8242, Lyon, France

Keywords: Ontology Modification, Multi-Agents System, Multi-User Context, Conflict Situation.

Abstract: The task of ontology evolution has been a topic of several research works leading to a number of tools facilitating the process of ontology evolution. These tools often propose different approaches without taking into account the multi-user context. It is either assumed that an ontology engineer changes the ontology, or that experts modify the ontology in asynchronous way. In this paper, we propose an approach to update ontologies for keeping knowledge up to date. This approach is based on agent's paradigm. The relevance of agents is that they can interact with each other to assist the expert to upgrade dynamically knowledge. In multi-user context, experts are able to modify simultaneously the same ontological entity which can generate conflict situations in the system. This one could generate ontology inconsistency. To control the conflict situation each agent uses predefined rules.

1 INTRODUCTION

In the last decade, the information technology explosion has led to changes in the organization and management of companies. In order to improve their reaction and adaptation period, companies have interested in tools supporting capitalization and management (Gandon, 2002). Knowledge management systems have helped organizations to capture, to capitalize and to share their knowledge. They enrich their in-house knowledge by capturing statements extracted from various sources of information (documents, databases, websites...). These statements could be transformed into ontology compatible elements or axioms, to describe domains in a formal way. However, the dynamic facet of domain has brought about ontology evolution to keep the knowledge up to date.

Handling the ontology evolution process in a distributed, decentralized environment is time-consuming and complex process. Therefore, we propose a framework to modify the ontology from document using the agent's paradigm to reduce the interventions of experts

Using multi-agent system (MAS) is motivated by the following reasons: i) The MAS runs when a new document is added, modified or deleted. Agents use its reactive behaviours to begin this incremental process. ii) To keep ontology consistency, agents use its local knowledge. iii) Interactions between agents support the ontology evolution process.

In a multi-user environment, experts can simultaneously modify the ontology. Collaborators change ontology in asynchronous mode, each one modifies the same version without knowing the modification made by the other one, which leads to conflict situation. Conflict is natural phenomenon in every field of human word, hence it express the divergence of interests or needs. Because the solving of ontology evolution's problem is divided among a number of agents, the conflicts between them happens in the case of incompatible goals. Therefore, it makes sense to examine the conflict situation, with the aim to achieve the ontology modification well.

In traditional way, collaborative conflicts are usually handled by mechanisms of locking or branching/merging provided by version management system (eg, cvs, subversion, Git, etc) (Chen et al., 2009). In this case, a conflict occurs when two developers modify the same source file. However, in the case of ontology evolution, users modify concepts and relationships between them. So, the use of the mechanisms listed is not possible.

In this paper, we try to answer the research questions: How design a multi-agent system to support ontology evolution in a multi-user context? This research question breaks down into the

Chamekh F., Talens G. and Boulanger D..
Ontology Modification in a Multi-User Concept - Conflict Resolution.
DOI: 10.5220/0005081602960303
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2014), pages 296-303
ISBN: 978-989-758-049-9

following sub-questions: (i) how to guide the expert to evolve ontology by keeping its consistency? (ii) How to define and manage the conflict situations?

The reminder of this paper is organized as follows: the section 2 presents some existing ontology evolution approaches. In the section 3 we discuss a use case scenario behind this work. In section 4 we present an overview of our framework. In section 5 we show our methodology for keeping consistency and resolving conflicts in the context of ontology evolution process supported by multi-agent system. Future works and conclusion are presented in section 6.

2 RELATED WORK

According to (Stojanovic, 2004), ontology evolution is defined as the "timely adaptation of ontology to the arisen changes and the consistent propagation of these changes to dependent artefacts".

Many ontology evolution frameworks rely on inconsistencies control lists that define the consequences of each change. To resolve inconsistencies of KAON (KArlsruhe ONtology) ontology, (Klein, 2004) propose a set of preconditions and post-conditions. Consistology (Jasiri et al., 2010) is a tool where providers ontology consistency using change kits (a set of rules and suggestions) which control the inconsistencies generated by each type of change.

Other approaches specify the consistency between components of the semantic web. Among others, (Luong et al., 2006) present CoSWEM (Corporate Semantic Web Evolution Management) as a system to manage effects of the ontological changes to the semantic annotations. They define rule-based approach for solving inconsistency, and (Rogozan et al., 2005) propose an approach for ontology evolution relatively to educational semantic web. They implement the Semantic Annotation Modifier system to keep consistency between ontology and resources.

Other frameworks introduce the patterns for maintaining overall consistency. Onto-Evoal (Ontology Evolution-Evaluation) (Djedid et al., 2010) present inconsistency resolution patterns. EVOLVA (Zablith et al., 2014), is an ontology management framework. It explores background knowledge sources (Wikipedia, WordNet, online ontologies….). The authors use a pattern-based approach to verify the relevance of the change against the base ontology.

Ontology evolution may consider also the multi-user context. Experts could modify ontology simultaneously. (Noy et al., 2006) propose two plugins: the change management plugin and the PROMPT plugin, integrated in PROTEGE to support the ontology evolution in the collaborative environment. The system use CHange and Annotation Ontology (CHAO) to describe the changes between versions. Each user annotates the change made.

There are few approaches investigating the problem of ontology evolution coupled with MAS. DYNAMO is adaptive multi-agent system (AMAS) for building and evolution of Terminological and Ontological Resource (TOR) from texts (Sellami et al., 2012). Each term and each concept try to find its right place in the AMAS organization that is the ontology. A set of behaviours for each type of agent is defined. Local rules are adopted to detect non-cooperative situation and actions to be taken to go back in a cooperative state. The ontology engineer uses inconsistence sheets which contain the inconsistency code and the changed term, to reach the modification. Another study of ontology evolution and MAS is given in (Rahman et al,. 2012). The authors present MAEKM (Multi Agent Enterprise Knowledge Management) based on ontologies modelling functional domains and multi-agent architecture performing the data retrieval and managing the changes that may occur within the data sources.

Our work presented in this paper can be compared with some similar existing studies. (Sellami et al., 2012) has presented DYNAMO as MAS for building and evolution TOR from texts. Nevertheless, this approach considers the agent as term and the MAS as ontology and it not has been mentioned the ontology evolution process. In (Rahman et al., 2012), the MAS manage the ontology consistency. However, they do not use predefined rules to solve the consistency and the collaborative context did not dealt with. Our evolution management system designs the ontology evolution process from documents based on agent's paradigm. Our work differs from the MAS system in DYNAMO for assigning each step of the process as agent's role. Relying on rule-based approach, our framework, encapsulate these rules as agent knowledge base to reach with inconsistency. Regarding to ontology evolution in a multi-user context, .(Noy et al., 2006) has presented CHange and Annotation Ontology. Despite that, this approach only presented a way to annotate the

change of each user but they do not specify techniques to verify and solve it.

3 USE CASE SCENARIO

As an example to illustrate concepts discussed so far, we propose an ontology describing aspects related to a healthcare domain especially general medicine, which acts as information based from Ontologos corporate (www.ontologos-corp.com). This information base contains medical reports and ontology. In order to describe the general medicine domain, the doctors define terms and relations.
The medical report is a document in which the doctor registers a clinical case study. This one is related to research work, for example the report number83 explains the role of the obesity in growing the asthma.

The first state of the system is to implement a semantic research engine. Due to the increase of the medical report, the existent ontology could be changed. Further to this, the expert needs a system to guide him to modify the ontology. Given this, the system requires to capture these changes for it to efficiently serve the expert and reduce inconsistencies.

Suppose that users add a new document. The system may analyse the expert's request in order to identify the change that could be done to the ontology (add concept, add instance, add property).
Once we have the type of modification, the system analyses the change by studying the effect on the consistency of ontology and between ontologies and documents. To identify the adequate operations related to each type of change, it is necessary to determine the type of change and inconsistencies. If different possibilities exist, i.e., different additional operations can be applied with different effects, the users have to choose the appropriate additional changes to implement. Various operations are displayed to experts in order to assist them. Experts validate the operation of change. The system checks the consistency of the dependent artefacts (application, document) after each change.

At the end, changes are merged from different experts. The system checks the coherence of the modifications by using a set of conflict resolution. When the system reaches the consensus, the new version of ontology (Vn+1) is created. The old version is saved in the log of version and all changes are backed up in the log of changes.

4 PLATFORM ARCHITECTURE

Our approach is based on the ontology evolution process identified by (Stojanovic, 2004) and (Klein, 2004). This process is made up of four steps: identification of changes, analysis of changes, and propagation of changes and management of versions. We assign to each agent a process step. Agents interact to run different processing steps.
The architecture of the proposed framework is shown in Figure 1. The system is composed of four agents: the user agent, the ontology agent, the consistency agent and the version agent, and three components: The Learning module, the log of changes and the log of versions (Chamekh et al., 2013).

We focus our ontology model to the OWL DL which is an axiom-oriented language. Classes and properties have structural descriptions specified through some defined constructors (Horrocks et al., 2003). Satisfying ontology within an interpretation is constraint by the satisfaction of all ontology axioms. For convenience we will adhere to more compact, traditional *SHOIN*(D) syntax. For the correspondence between this notation and various OWL DL syntaxes see (Horrocks et al., 2004).

Definition 1: an ontology O is a set of concepts, properties and individuals.
$O=\{C, P, I\}$
$C= \{c_1,c_2,c_3,....c_{n-1},c_n\}$
$P= \{p_1,p_2,p_3,....p_{n-1},p_n\}$

Example 1: As a running example, we will consider an extract of general medicine domain, consisting of the following axioms:
Non inflammatory pathology ⊆ Pathology, inflammatory pathology ⊆ Pathology, (Non inflammatory pathology and inflammatory pathology are pathologies)
Infection inflammatory pathology ⊆ Non inflammatory pathology

4.1 User Agent

It detects the change to be realized by analysing the user request. In order to perform all tasks related to the management expert request, we have provided several behaviours: analyse the user request, communicate with the other agents and user, trigger learning module and verify the existence of documents.

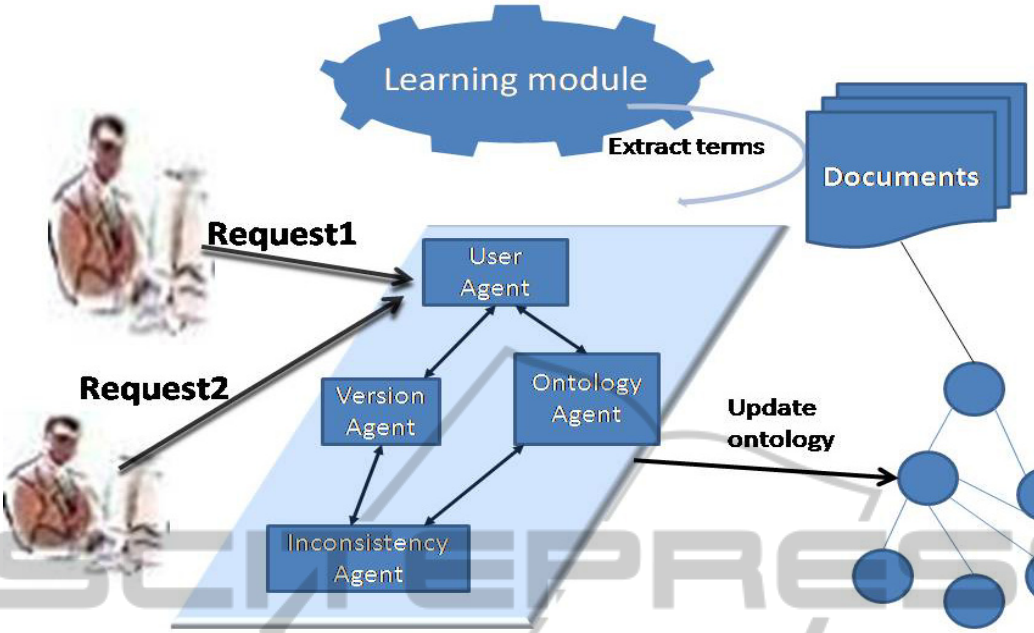

Figure 1: Platform architecture.

When the user agent analyses the request, three alternative operations exist : Add document, Delete document, Modify document:

4.2 Learning Module

This module receives a document as input. Using Text2Onto (Maedche, et al,2000) which is a framework for data-driven change discovery by incremental ontology learning. It uses natural language processing and text mining techniques. The output generated is XML/OWL file. This one contains the extracted terms. These terms are categorised as concepts, instances and taxonomic relations.

As example of concept presentation:

<a:Concept rdf:ID="non infection inflamatory pathology_c">

<a:Ratingrdf:datatype="http://www.w3.org/2001/XMLSchema#double">0.0136986301369863</a:Rating>

<owlx:Labelrdf:datatype="http://www.w3.org/2001/XMLSchema#string">night</owlx:Label> </a:Concept>

Example2. Let's consider a change made by user 1 which adds a new medical report. The user agent verifies if the document exists. If it is true, the user agent informs the user that the document exists and he must drop this operation. If it not, the user agent triggers the learning module, and extracts:

The concept: *Non infection inflammatory pathology.*

4.3 Ontology Agent

It analyses a change. In order to perform all tasks related to the management of the user request, we provide this agent by various behaviours: detect the ontology to modify, search similar entities and define the changes to be proposed.

Definition2: An ontology change operation is a set of change operation assigned to the ontological entity $OC = \{oc_1, oc_2, oc_3, \ldots oc_{n-1}, oc_n\}$.

When the user adds a document, the ontology agent detects the ontologies to modify. The ontology is founded by similarity measure between the name of ontology and the topic of document. If ontology exists the ontology agent seeks similarities between extracted terms and the ontological entity. Depending to the level of similarity, the ontology agent proposes the changes based on its knowledge base (Chamekh, et al, 2013).

Example 3. Let's take again the change made by user 1. The ontology agent1 detects the ontology to modify and seeks similarities between extracted terms and the ontological entities of the ontology O. In this case, it uses its knowledge base and proposes that the concept *Non infection inflammatory pathology* can be a sub concept to the concept *inflammatory pathology.*

The management of the ontology consistency and the conflict resolution are ensured by the

consistency agent and the version agent. A detailed presentation is given in section5.

4.4 Consistency Agent

The identification of types of change to apply on the ontology, expresses formally the needs of evolution required by users. When they are applied, the ontology changes from a current version to another one. However, the application of a type of change could generate inconsistencies on the new ontology version. In our framework, we assign the task of keeping consistency to the consistency agent.

Two types of OWL consistency are distinguished: structural consistency and logical consistency. Structural consistency refers to syntactic conditions of OWL DL constructors and to the constraints specified on its elementary axioms and their combinations (Djedidi el al., 2010). Logical consistency refers to the formal semantic of the ontology and thus, to its satisfiability in the meaning that ontology is semantically correct and does not present any logical contradiction.

Definition 3: we define consistency Q as a set of consistency condition: $Q= \{q_1,q_2,q_3,.....q_{n-1},q_n\}$. An Ontology O is consistent if it satisfies all the consistency conditions Q(O).

Many reasoners are available to verify logical inconsistency. However, some of them like Pellet detects inconsistency but does not precise the axiom that cause inconsistencies neither how to resolve the detected inconsistencies.

In our work, we propose to detect and resolve inconsistencies before they have occurred. So, in this case the reasoner cannot be used.

To guarantee a logical and structural consistency, we identify a set of rules. They are implemented as a knowledge base of consistency agent (Chamekh, et al, 2013).

Definition4: we define an additional change as a set of operation that can keep the consistency of ontology, $AOC=\{ aoc_1, aoc_2,.....aoc_n\}$.

The role of the consistency agent is keeping the consistency of ontology during the process. Each consistency agent is connected to another ontology agent and another version agent in accordance with the negotiation protocol. The negotiation allows the consistency agent to perceive its environment. The consistency agent has to achieve three objectives:

- To denote a type of change by perceiving the message sent by the ontology agent.
- To study the effect of changes and propose additional changes. The agent uses consistency rules encapsulated as a knowledge base.
- To propose the additional changes to the user by communication with the user agent.

The general behaviour of consistency agent is presented through the algorithm1.

The consistency agent perceives the proposed change and the ontological entity from the ontology agent. This one checks the consistency of ontology: if inconsistency exists, the consistency agent proposes additional change (based on its knowledge base) to the user, if he validates the consistency agent launches the task of creation new version; otherwise, the change is cancelled.

Algorithm 1: general behaviour of consistency agent.

Begin
Perceive ()
$OC=\{ oc_1,oc_2,oc_3\}$;
O ontology
$C= \{c_1,c_2,c_3,....c_{n-1},c_n\}$
Process verification of consistency ();
If (not Q(O))
Process of denotation of consistency problem ();
If (aoc_1€ AOC exist)
Process proposes additional change ();
Process to the user ();
If user validate
Process creation new version ();
Else
Undo change ();
End if
Else
Process to the user ();
End if
Stop agent ();
End if
End

Example 4: Let's consider again a required change presented up.The inconsistency agent1 studies the impact of this change (the concept Non infection inflammatory pathology can be a sub concept to inflammatory pathology) to the ontology consistency by using the rules R1 and R2.

R1: if we add a sub-concept A to a non-leaf concept B, we must compare the property of concept A with the property of concept C (which is the sub-concept of concept B). If there are common properties between concept A and concept C, we must verify R2, else (if not) we must verifyR3

R2: If there are common properties between the concept A and the concept C

R3: If there are not common properties between the concept C and the concept A

4.5 Version Agent

In a multi-user context, users add simultaneously the new document. Agents run the process of ontology evolution until the creation of new version. There are two types of conflict situations: The simple conflict situation and the complex conflict situation.

Definition5: we define a conflict situation as a set of simple and complex conflict.

$C=\{c_{simple}, c_{complex}\}$

$C_{simple}=\{c_{s1},c_{s2},.....c_{sn}\}$

$C_{complex}=\{c_{c1},c_{c2},.....c_{cn}\}$

The agent behaviour is illustrated by algorithm 2. Firstly, the agent receives the change, the concerned ontological entity and the version of ontology. Secondly, it verifies the current version; it applies the change and creates the new version, if it is the same version. Else the version agent verifies in the log of changes the modifications that bound to the ontological entity. The version agent searches in the local rules the concerning conflict situation. If it is a simple conflict situation, it resolves the problem, and creates the new version and if it is a complex situation conflict, the version agent sends a message to the inconsistency agent with a proposed change. When the conflict situation doesn't exist in the rules base, the version agent requests the validation of the user and updates its knowledge base.

The simple conflict situation that can be directly solve by the version agent.

Simple conflict situation 1:
The user 1 links or adds the instance I to the concept A in the ontology O version X and in the log of changes the property Y of the concept A has been renamed in the property Y'. The version agent adds or links the instance I to the concept A and creates the new version.

Simple Conflict situation 2:
The user 1 rename the property Y (property Y') to the concept A in the ontology O version X and in the log of changes the concept A has been renamed to the concept A'. The version agent renames the property Y (property Y') to the concept A' and creates the new version.

Simple Conflict situation 3:
The user 1 links or adds the instance I to the concept A in the ontology O version X and in the log of changes the concept A has been renamed to the concept A'. The version agent adds or links the instance I to the concept A' and creates the new version.

Simple Conflict situation 4:
The user 1 adds the property Y or renames the property Y (property Y') to the concept A in the ontology O version X and the concept A has been renamed to the concept A'. The version agent renames the property Y (property Y') or adds the property Y to the concept A' and creates the new version.

Simple Conflict situation 5:
The user 1 renames the property Y (property Y') to the concept A in the ontology O version X and in the log of changes the property Y of the concept A has been renamed to the property Z. The version agent renames the property Z in to the property Y' and creates the new version.

Simple Conflict situation 6:
The user 1 adds upper-concept/sub-concept B to concept A in the ontology O version X and in the log of changes the concept A has been renamed to the concept A'. The version agent adds upper-concept/sub-concept B to the concept A' and creates the new version.

Simple Conflict situation 7:
The user 1 adds a synonym concept B to concept A in the ontology O version X and in the log of changes the property Y of the concept A has been renamed to the property Y'. The version agent adds a synonym concept B to the concept A and creates the new version.

The complex conflict situation cannot be resolved by the version agent. It sends proposed changes to the inconsistency agent. This one uses the consistency rules to solve the problem.

Complex Conflict situation 1:
The user 1 links or adds the instance I to the concept A in the ontology O version X and in the log of changes new instances have been added to the concept A. The version agent sends a change <add/link the instance I to the concept A> to the inconsistency agent.

Complex Conflict situation 2:
The user 1 links or adds the instance I to the concept A in the ontology O version X and in the log of changes the instance I has been linked to the concept B. The version agent sends change <add/link the instance I to the concept A> to the inconsistency agent.

Complex Conflict situation 3:
The user 1 adds upper-concept/sub-concept B to concept A in the ontology O version X and in the log of changes the sub-concept C or upper-concept D has been added to the concept A. The version agent

sends a change <adds upper-concept/sub-concept B to concept A > to the inconsistency agent.

Complex Conflict situation 4:

The user 1 adds upper-concept/sub-concept B to concept A in the ontology O version X and in log of changes the property Y has been added to the concept A. The version agent sends a change<adds upper-concept/sub-concept B to concept A >to the inconsistency agent.

Complex Conflict situation 5:

The user 1 adds upper-concept/sub-concept B to concept A in the ontology O version X and in the log of changes, the property Y has been renamed to the property Y'. The version agent sends a change<add upper-concept/sub-concept B to concept A >to the inconsistency agent.

Algorithm 2: general behaviour of version agent

Begin
Perceive ()
OC= { oc_1,oc_2,oc_3};
Vn ontology version;
Process verification version ();
If (Vn=current version)
Process create new version ();
Else
Process check change ();
If (OC not exist in log of change)
Process create new version ();
Else
Process search conflict situation ();
If (C_{simple})
Process create new version ();
Else
Process sends change ();
End if
End if
End if
Stop change ();
End.

Example 5. To illustrate a possible instantiation of this conflict situation and let's consider again a required change Ch1 made by user 1 defining non infection inflammatory pathology as a sub concept of inflammatory pathology. After verifying the consistency of ontology behind the change made by user 1, the consistency agent sends the change, the corresponding entity and the version of ontology. The version agent verifies if it is the current version. The current version is not the same version sent by the consistency agent. The version agent verifies in the log of changes if the entity *inflammatory pathology* has been changed. It detects that this entity has been renamed by another user to inflammatory pathologies. The version agent detects a conflict situation. It found that it can use the simple conflict situation 6. The version agent creates the new version.

5 CONCLUSIONS

In this work, we treat ontology evolution from concepts extracted from documents. The different steps of evolution phases are decomposed inside different agents. We try to deal with the problem of conflict situations in a multi-user context. So the modifications on concepts (and ontologies) may be performed in a parallel way.

Some conflict situations can be directly solved by agents. Some of them must be sent to the concerned agent who is in charge of modifications. The first perspective is to formalize the different conflict rules. Secondly, to define the process of decision of each agent and finally, to confirm our assumptions by the implementation of the system.

REFERENCES

Chen,Y, Peng, X and Zhao, W., 2009 An Approach to Detect Collaborative Conflicts forOntology Development, *In the procedeegins of Advances in Data and Web Management, Joint International Conferences, APWeb/WAIM,* China, pp.242-254.

Chamekh, F, Talens, G, Boulanger, D, 2013. Corporate semantic web evolution: an approach based on multi-agent system. *In the procedeegins of International conference on Management of computaionnal and collective intelligence in digital ecosystems*, Luxembourg, pp. 235-240.

Chamekh, F, Talens, G, Boulanger, D, 2013. Ontology evolution in the corporate semantic web. *In the procedeegins of International conference on Entreprise Information Systems*, Angers, France, pp. 182-189.

Djedidi,R, Aufaure, M.A., 2010.Onto-Evo an Ontology Evolution Approach Guided by Pattern Modelling and Qualit Evaluation*, In the procedeegins 6th International Symposium on Foundations of Information and Knowledge Systems* , LNCS: Vol. 5956, G.Berlin, Germany: Springer. Sofia, Bulgaria, pp. 268-305.

Gandon, F., 2002. Distributed Artificial Intelligence and Knowledge Management: Ontologies and multi-agent systems for a corporate semantic web, PhD Thesis INRIA, November .

Horrocks, I, Patel-Schneider,P.F and van Harmelen, F. From SHIQ and RDF to OWL: The Making of a Web Ontology Language .Journal of Web Semantics, 1(1), 2003.

Horrocks, I, Patel-Schneider,P.F and van Harmelen, F. Reducing OWL Entailment to Description LogicSatisfiability. Journal of Web Semantics, 1(4), 2004

Klein, M, 2004. Change Management for Distributed Ontologies,PhD thesis,University of Amsterdam.

Luong,P-H, Dieng-Kuntz, Boucher, A., 2006. Managing semantic annotations evolution in the CoSWEM system, *In Proceedings of the Third National Symposium on Research, Development and Application of Information and Communication Technology*, Hanoi ,pp. 215 –223.

Maedche,A . Staab,S., 2000. The text-to-onto ontology learning environment, In Software Demonstration at ICCS-2000-Eight *International Conference on Conceptual Structures.*

Noy,N.F, Chugh,A, Liu,W, Musen ,M A., 2006. A framework for ontology evolution in collaborative environments, *In Proceedings of the 5th International Semantic Web Conference* ISWC'06.

Rogozan,D ,Paquette, D., 2005.Managing ontology changes in the semantic web, *In the IEEE/WIC/ACM International Conference on Web Intelligence*, Compiegne, France.

Rahman,S.D, Yadav, D, Agarwal, P, Bisht, P.S., 2012. Multiagent Knowledge Management Architecture. Journal of Software Engineering and Application , Volume 5,pp. 33–40.

Stojanovic,L.,2004., Methods and Tools for Ontology Evolution, PhD thesis, University Karlsruhe.

Sellami, Z, Camps, V., 2012. Evaluation of a multi-agent system for the evolving of domain ontologies from text, *In proceedings of International Conference on Practical Applications of Agents and Multiagent Systems (PAAMS), Salamanca , p. 169-179.*

Jaziri,W, Sassi, N, Gargouri, F., 2010. Approach and tool to evolve ontology and maintain its coherence, Metadata, Semantic and Ontologies, vol. 5, n°2, pp. 151-166.

Zablith, F, Antoniou,G, d'Aquin, M, Flouris,G, Kondylakis,H, Motta,E , Plexousakis,D and Sabou, M.,2014 Ontology Evolution: A Process Centric Survey, In: The Knowledge Engineering Review, Vol0:00, pp.1-31.