Algorithms for the Reconciliation of Ontologies in Open Environments

Yaqing Liu, Rong Chen, Hong Yang

Algorithms for the Reconciliation of Ontologies in Open Environments

Yaqing Liu[1], Rong Chen, Hong Yang

School of Information Science & Technology, Dalian Maritime University,
116026 Dalian, China
{liuyaqing234@yeah.net, tsmc.dmu@gmail.com}

Abstract. The dynamic changing feature of Semantic Web determines that the ontology which is a part of Semantic Web needs constantly to be modified in order to adapt outer environment. In this paper we make a careful analysis of the ontology changes' complexity under open environment. The main contents discussed are as follow. At first we point out all possible relation types between any two ontology change sequences including directly conflict relation, indirectly conflict relation, dependent relation and compatible relation according to ontology change's definition. And then we propose a new algorithm named Algorithm of Searching Maximum and Sequential Ontology Change Sequence Set(ASMSOCSS) to find all maximum and sequential ontology change sequence subset in the prime ontology change sequence set and prove the independence of the result which may be got after running ASMSOCSS. At last we put forward the algorithm by using these maximum and sequential ontology change sequence sets to create new ontology versions according to the dependence relation between ontology change sequences.

Keywords: Ontology Changes Sequence, Maximum and Sequential Ontology Changes Sequence, Ontology Change.

1 Introduction

Ontology Evolution is the timely adaptation of an ontology to the arisen changes and the consistent propagation of these changes to dependent artifacts. Ontology change disposal, as a part of ontology evolution, focuses on exploring some ontology evolution's methods and technologies to modify ontology on the assumption with not breaking ontology consistency. By far, a lot of research work has been done on ontology change disposal and they may be classified into some based on logical reasoning[6] and others based on belief revision[8].But all of them mainly focus on ontology change disposal under centralized environment. The research on ontology change disposal under open environment is infrequent. [1] discussed ontology change disposal under open environment for the first time. Its main contribution is to define

[1] The corresponding Author. This work was supported by National Natural Science Foundation of China (60775028), Dalian Science &Technology Program (2007A14GX042),Young Key Teachers Foundation Projects of Dalian Maritime University (2009QN034)

the mapping relation between any two ontology versions by analyzing these logs of editing ontology. But [1] is not all-inclusive because it can't provide some guide for ontology evolution's trend in semantic level.

Multiple ontology versions will be achieved under open environment. But these ontology versions are not all worthy. Moreover, too many ontology versions will make it more difficult to manage the base of ontology versions. Unfortunately, how to get worthy ontology versions under open environment is seldom concerned. In addition, the problem on ontology change of analysis and disposal is independent of the problem on merging multiple ontology versions. And it is well known that the work of merging ontology versions is very heavy. If the work of merging ontology versions can be integrated into the course of analysis and disposal of ontology changes it will greatly save the work of ontology evolution. In this paper, we propose a new method which may not only create a worthy ontology version but also avoid the work of merging ontologies through analyzing and disposing ontology change sequences.

This paper is organized as follow. The whole scheme of ontology change disposal under open environment is given in section 2. And then we make certain all possible relation types between ontology change sequences in section 3. We propose Algorithm of Searching Maximum and Sequential Ontology Change Sequence Set in section 4 and put forward the algorithm used to create new ontology versions in section 5. Related works are mentioned in section 6 and conclusion and the next work are arranged in the last section.

2 Description of the Approach

Our approach on ontology change in an open environment is composed of four steps as illustrated in Figure 1.

Step 1: Having accesses to a version O_i of an ontology, several users have separate copies O_i' of this ontology in their working spaces.

Step 2: Each user makes changes chs_j to his/her copy respectively.

Step 3: Given sequences of users' ontology changes, we algorithmically analyze the relevance between ontology change operations and search for the maximal consistent subset of the whole ontology change operations, which is called the maximal ontology changes set. We put such subsets together and denote them as a collection of ontology change sequences, i.e., $\{CHS_1, CHS_2, \cdots, CHS_k\}$, where each CHS_i ($1 \le i \le k$) represents a maximum sequential ontology changes.

Step 4: Several versions of ontologies may be derived from the original O_i by applying one of maximal ontology changes. Obviously, $\{CHS_1, CHS_2, \cdots, CHS_k\}$ will produce k distinguished ontologies.

The above-mentioned step 3 and step 4 are the crucial parts of our approach, next we will give their details in the remainder of this paper.

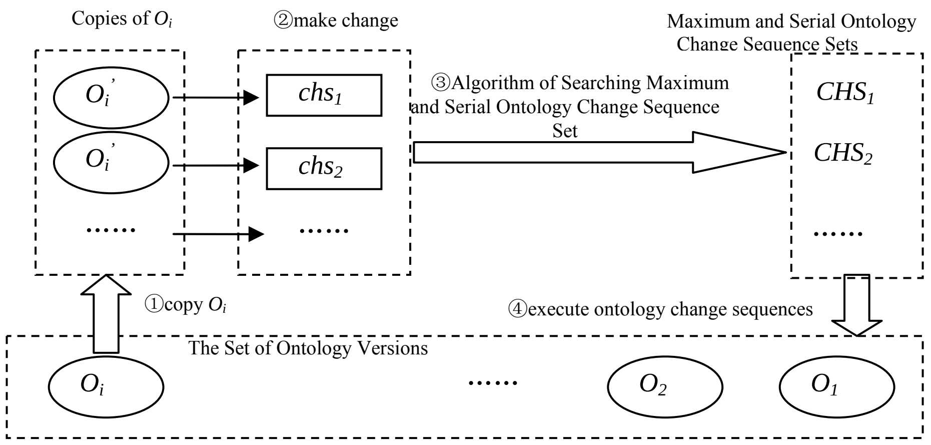
Copies of O_i
②make change
Maximum and Serial Ontology Change Sequence Sets
$O_i^{'}$
$O_i^{'}$
......
chs_1
chs_2
......
③Algorithm of Searching Maximum and Serial Ontology Change Sequence Set
CHS_1
CHS_2
......
①copy O_i
④execute ontology change sequences
The Set of Ontology Versions
O_i
......
O_2
O_1

Fig. 1. Description of our approach

3 Formal Description of Ontology Change

Definition 1: an **ontology** *O* is defined as a 5-tuple:

$$O=\{C,R,H^c,Rel,A^o\} \ . \qquad (\mathbf{1})$$

where:

- C is the set of ontology concepts.
- $H^C \subseteq C\times C$ is a set of taxonomic relationships between concepts.
- R is the set of non-taxonomic relationships. The function Rel: $R\rightarrow C\times C$ maps the relation identifiers to the actual relationships.
- A^o is a set of axioms, usually formalized into some logic language.

For brevity, *O* is short for ontology throughout this paper. Its instances are denoted by *O.I*, concept set by *O.C*, non-taxonomic relationships by *O.R*. We think of an ontology as a knowledge base, which contains not only the elements of an ontology, but also instances of the concepts *C* and relationships *R*.

Stojanovic categorizes all ontology changes into "Add" ontology changes and "Remove" ontology changes respectively. To highlight what type of changes is made to what object, we redefine ontology change as follows:

Definition 2: an **ontology change** *ch* is defined as:

$$ch=\{name,type,object,args\}. \qquad (\mathbf{3})$$

where:

- *name* is the identifier of this change *ch*.
- $type \in \{$"*Add*", "*Remove*"$\}$, is type of *ch*.
- $object \subset O.I \cup O.C \cup O.R$, are the elements which *ch* act on. *O* is an ontology. *O.C* is the set of concepts of *O* and *O.R* is the set of non-taxonomic relationships of *O*.

- $args \subset O.I \cup O.C \cup O.R$, is a list of one or more change arguments. There are changes with one, two or three arguments.

For example, ch={"*AddSubConcept*", "*Add*", {$subc_2$}, {*supc*}} denotes an ontology change that a concept $subc_2$ is added to an ontology as subconcept of *supc*. This can be pictured as Figure 2.

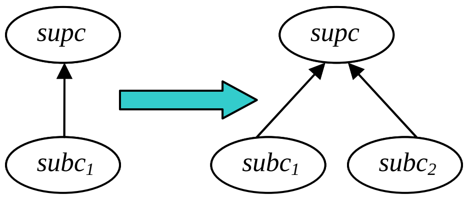

Fig.2. An illustration of an ontology change.

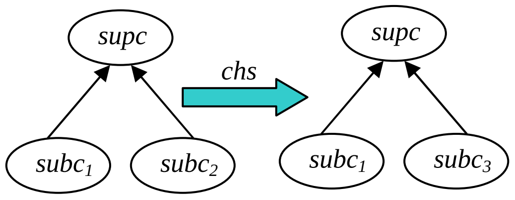

Fig.3. An illustration of an ontology change sequence.

Definition 3: an **ontology change sequence** *chs* is defined as

$$chs{=}{<}ch_1ch_2\ldots ch_n{>},\ ch_i\ (1 \le i \le n) \text{ is an ontology change} \qquad \textbf{(4)}$$

if and only if n=1 or $\forall\, 1 \le i \le n$-1, ch_i always is executed ahead of ch_{i+1}.

3.1 Argument Pool and Remove Pool of an Ontology Change Sequence

For an ontology change sequence $chs{=}{<}ch_1ch_2\ldots ch_n{>}$, the **argument pool** of *chs* is used to enumerate all $ch_i.args$ when $ch_i.type$ is "Add" and the **remove pool** of *chs* is used to enumerate all $ch_j.object$ when $ch_j.type$ is "Remove". Further, the argument pool *chs.APool* and the remove pool *chs.RPool* of an ontology change sequence $chs{=}{<}ch_1ch_2\ldots ch_n{>}$ can be obtained when we apply the *ACAPRP* algorithm to *chs*.

Given a simple example illustrated in Fig.3, we suppose:

- $chs{=}{<}ch_1ch_2{>}$
- ch_1={"*RemoveConcept*", "*Remove*",{$subc_2$},{}}
- ch_2={"*AddSubConcept*", "*Add*",{$subc_3$},{*supc*}}

When *ACAPRP* algorithm is applied to *chs* the output is *chs.RPool*={$subc_2$} and *chs.APool*={*supc*}.

3.2 Relation Types between two Ontology Change Sequences

Definition 4: An ontology change sequence chs_1 **depends on** another ontology change sequence chs_2 if and only if $chs_1.RPool \cap chs_2.APool \neq \varnothing \wedge chs_2.RPool \cap chs_1.APool = \varnothing$. *dependence*(chs_1,chs_2) means that chs_1 depends on chs_2 and ¬*dependence*(chs_1,chs_2) means that chs_1 doesn't depend on chs_2.

Definition 5: An ontology change sequence chs_1 **directly conflicts** with another ontology change sequence chs_2 if and only if $chs_1.RPool \cap chs_2.APool \neq \varnothing \wedge chs_2.RPool \cap chs_1.APool \neq \varnothing$. *directlyConflict*(chs_1,chs_2) means that chs_1 and chs_2 directly conflict with each other and ¬*directlyConflict*(chs_1,chs_2) means that chs_1 and chs_2 don't directly conflict with each other.

```
Algorithm 1:ACAPRP(chs)
Input: an ontology change sequence $<ch_1ch_2\ldots\ldots ch_n>$
Output: $chs.RPool, chs.APool$
1.  $chs.RPool \leftarrow \varnothing$;   $chs.APool \leftarrow \varnothing$;   $i \leftarrow 1$;
2.  WHILE($i<=n$)
3.      $chs.APool \leftarrow chs.APool \cup ch_i.args$;
4.      IF $ch_i.type$="Remove"
5.          $chs.RPool \leftarrow chs.RPool \cup ch_i.object$;
6.      IF $ch_i.type$="Add" and $ch_i.object \subseteq chs.RPool$
7.          $chs.RPool \leftarrow chs.RPool - ch_i.object$
8.      $i$++;
9.  Return $chs.RPool$ and $chs.APool$
```

Definition 6: For a few of ontology change sequences chs_1, chs_2, …, chs_n, chs_1 **indirectly conflicts** with chs_n iff *dependence*(chs_1,chs_2), *dependence*(chs_2,chs_3),…, *dependence*(chs_n,chs_1) all are true, where *indirectlyConflict*(chs_1,chs_n) means that chs_1 and chs_n indirectly conflict with each other and ¬*indirectlyConflict*(chs_1,chs_n) means that chs_1 and chs_n don't indirectly conflict with each other.

Definition 7: An ontology change sequence chs_1 is **compatible** with another ontology change sequence chs_2 iff $chs_1.RPool \cap chs_2.APool=\varnothing \wedge chs_2.RPool \cap chs_1.APool=\varnothing$, where *compatible*(chs_1,chs_2) means that chs_1 and chs_2 are compatible each other.

4 Algorithm of Searching Maximal Ontology Changes Set

According to the relation type between two ontology change sequences, four lemmas may be deduced.

Lemma 1: If an ontology change sequence chs_1 directly conflicts with another ontology change sequence chs_2, not all of chs_1 and chs_2 are executed no matter what the execution order may be.

Lemma 2: If an ontology change sequence chs_1 depends on another ontology change sequence chs_2, chs_1 and chs_2 can be all executed if and only if chs_2 is executed ahead of chs_1.

Lemma 3: If an ontology change sequence chs_1 indirectly conflicts with another ontology change sequence chs_n, not all of chs_1 and chs_n are executed no matter what the execution order may be.

Lemma 4: If an ontology change sequence chs_1 is compatible with another ontology change sequence chs_2, chs_1 and chs_2 may always be executed no matter what the execution order may be.

According lemma 1,2,3 and 4, if any two of a group of ontology change sequences are not *directlyConflict* or *indirectlyConflict* all these ontology change sequences may be executed. In order to find all maximum subset of all ontology change sequences that may be executed from a given group of ontology change sequences, we propose the Algorithm of Searching Conflict Set(ASCS) to be used to find all ontology change sequence pairs which indirectly conflict or directly conflict with each other and the Algorithm of Searching Maximum and Sequential Ontology Change Sequence Set (ASMSOCSS).

Definition 8: Given a group of ontology change sequences $CHS=\{chs_1, chs_2, \cdots\cdots, chs_n\}$, $\mathrm{MSOCSS}_{\mathrm{CHS}}$ is defined as **Maximum and Sequential Ontology Change Sequence Set** if and only if $\neg\exists T\subseteq CHS$ makes all $T\supset MSOCSSM_{CHS}$ and $\forall t_1,t_2\in T$, $\neg directlyConflict(t_1,t_2)\wedge\neg indirectlyConflict(t_1,t_2)$ to be true.

In ASMSOCSS, step 2 means to traverse all elements of *CONF* in unknown order. It is puzzling whether the order of traversing elements can change the final MSOCSSSet or not. The problem can be explained by lemma 5.

Lemma 5: The final MSOCSSSet is identical no matter what the order of traversing elements may be.

Proof

$\forall(e1,e2),(e3,e4)\in CONF$, $e1\in CHS\wedge e_2\in CHS\wedge e_3\in CHS\wedge e_4\in CHS$ is given.

Suppose that $e_1\neq e_3\wedge e_1\neq e_4\wedge e_2\neq e_3\wedge e_2\neq e_4\wedge e_1\neq e_2\wedge e_3\neq e_4$ is true.

If $(e1,e2)$ is traversed ahead of $(e3,e4)$, $\mathrm{MSOCSSSet}_0=\{CHS/\{e_1\},CHS/\{e_2\}\}$ is got. After $(e3,e4)$ is traversed, $\mathrm{MSOCSSSet}_1=\{CHS/\{e_1,e_3\}$, $CHS/\{e_1,e_4\}$, $CHS/\{e_2,e_3\}$, $CHS/\{e_2,e_4\}\}$ is got. In turn, if $(e3,e4)$ is traversed ahead of $(e1,e2)$, $\mathrm{MSOCSSSet}_0=\{CHS/\{e_3\},CHS/\{e_4\}\}$ is got. After $(e1,e2)$ is traversed ,$\mathrm{MSOCSSSet}_2=\{CHS/\{e_3,e_1\}$, $CHS/\{e_3,e_2\}$, $CHS/\{e_4,e_1\}$, $CHS/\{e_4,e_2\}\}$ is got. Obviously $\mathrm{MSOCSSSet}_1=\mathrm{MSOCSSSet}_2$.$\mathrm{MSOCSSSet}_1=\mathrm{MSOCSSSet}_2$ is easy proved when $e_1=e_3\wedge e_1\neq e_4\wedge e_2\neq e_3\wedge e_2\neq e_4\wedge e_1\neq e_2\wedge e_3\neq e_4$ is true.

$\forall(e1,e2),(e3,e4)\in CONF$, the same conclusion may be drew easily whichever of $e1\notin CHS\wedge e_2\in CHS\wedge e_3\in CHS\wedge e_4\in CHS$ or $e1\notin CHS\wedge e_2\notin CHS\wedge e_3\in CHS\wedge e_4\in CHS$ or $e1\notin CHS\wedge e_2\in CHS\wedge e_3\notin CHS\wedge e_4\in CHS$ or $e1\notin CHS\wedge e_2\notin CHS\wedge e_3\notin CHS\wedge e_4\notin CHS$ is given.

So we may know that the order of traversing any two elements of *CONF* is exchangeable. Further, Lemma 5 may be deduced easily according to related mathematical characteristics.

□

5 Algorithm of Generating New Ontology Versions

According to ASMSOCSS, $\forall E\in MSOCSSSet$, $\forall(e_1,e_2)\in E$, e_1 must not directly conflict or indirectly conflict with e_2. So $\forall E\in MSOCSSSet$, all elements in E are executed once. When a $MSOCSS_{CHS}$ is applied to an ontology O, a new ontology version can be got. The algorithm of generating a new ontology version is AGNOV.

```
Algorithm 2：ASCS($CHS$)
Input：a group of ontology change sequences $\{chs_1, chs_2, \cdots\cdots, chs_n\}$
Output：a conflict set $CONF$
1.  $CONF \leftarrow Ø$;        $i1 \leftarrow 1$;   $i2 \leftarrow 1$;
2.  WHILE($i1<=n$)
3.      WHILE($i2>=i1$ and $i2<=n$)
4.          IF($directConflict(chs_{i1},chs_{i2})$ or $indirectConflict(chs_{i1},chs_{i2})$) and $i1 \neq i2$
5.              $CONF \leftarrow CONF \cup \{\{chs_{i1},chs_{i2}\}\}$;
6.          $i2$++;
7.      $i1$++;   $i2 \leftarrow i1$;
8.  RETURN  $CONF$
```

```
Algorithm 3：ASMSOCSS($CHS$,$CONF$)
Input：a group of ontology change sequences $\{chs_1, chs_2, \cdots\cdots, chs_n\}$ and
        ASCS($\{chs_1, chs_2, \cdots\cdots, chs_n\}$)
Output：the set of all possible maximum and serial ontology change sequence
        set $MSOCSSSet$
1.  $MSOCSSSet \leftarrow \{CHS\}$;      $i1 \leftarrow 1$;   $i2 \leftarrow 1$;
2.  FOR EACH $(chs_1,chs_2) \in CONF$
3.      FOR EACH $e \in MSOCSSSet$
4.          IF $chs_1 \in e$ and $chs_2 \in e$
5.              $MSOCSSSet \leftarrow MSOCSSSet - \{e\}$;
6.              $e_1 \leftarrow e - \{chs_1\}$;      $e_2 \leftarrow e - \{chs_2\}$;
7.              IF $\neg \exists e' \in CAND$ , $e_1 \subset e'$
8.                  $MSOCSSSet \leftarrow CAND \cup \{e_1\}$;
9.              IF $\neg \exists e' \in CAND$ , $e_2 \subset e'$
10.                 $MSOCSSSet \leftarrow CAND \cup \{e_2\}$;
11. RETURN $MSOCSSSet$
```

```
Algorithm 4：AGNOV($O$)
Input：an ontology $O$
Output：the set of all new ontology version $OVs$
1. $OVs \leftarrow Ø$;
2. FOR EACH $E \in MSOCSSSet$
3.     $O' \leftarrow O$;
4.     WHILE($E \neq Ø$)
5.         IF $\exists\ chs_i \in E, \forall\ chs_j \in E/\{chs_i\}$,  $\neg dependence(chs_i, chs_j)$
6.             $O' \leftarrow chs_i(O)$;   $E \leftarrow E/\{chs_i\}$;
7.     $OVs \leftarrow OVs \cup \{O'\}$;
8. RETURN OV
```

6 Related Work

Ontology versioning[5] typically involves the storage of several ontology versions and identification issues, the relationship between different versions as well as compatibility information.[9] uses the term versioning to describe their approach of ontology change. They define ontology versioning as the ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology. Adequate methods and tools must be used to distinguish and identify the versions. [4] presents a new ontology evolution approach. The approach keeps track of the different virtual versions of ontology concepts throughout their lifetime by combining the manual request for changes by the ontology engineer with an automatic change detection mechanism. [7] proposes a logic framework used to reason with multversion ontologies. In the framework such problems can be solved as querying log of semantic change, selecting a appropriate ontology version, etc.

7 Conclusion and Future Work

We propose a formal method used to analyze and dispose a group of ontology change sequences under open environment. But our discussion is specific to the same ontology subject to a group of ontology change sequences. In future, we will use graph theory to rephrase such a problem.

References

[1] Michel Klein, Academisch Proefschrift, Michel Christiaan, Alexander Klein, and Prof. Dr. J. M. Akkermans. Change management for distributed ontologies. Technical report, 2004.

[2] Changing Ontologies Peter, Peter Haase, Frank Van Harmelen, Zhisheng Huang. A framework for handling inconsistency in pages 353--367. Springer, 2005.

[3] Aditya Kalyanpur, Bijan Parsia, Evren Sirin. Repairing unsatisfiable concepts in owl ontologies. In *3rd European Semantic Web Conference (ESWC2006)*, June 2006.

[4] Peter Plessers and Olga De Troyer. Ontology change detection using a version log. In *Proceeding of the 4th International Semantic Web Conference*, pages 578--592. Springer, 2005.

[5] Natalya F. Noy, Abhita Chugh, William Liu, and Mark A. Musen. Musen m.: A framework for ontology evolution in collaborative environments. In *5th International Semantic Web Conference*, pages 544--558. Springer-LNCS, 2006.

[6] Peter Haase, Peter Haase, and Ljiljana Stojanovic. Consistent evolution of owl ontologies. pages 182--197. Springer, 2005.

[7] Zhisheng Huang and Heiner Stuckenschmidt. Reasoning with multi-version ontologies: A temporal logic approach. In *Proceeding of the 4th International Semantic Web Conference ISWC*, pages 398--412, 2005.

[8] Giorgos Flouris, Giorgos Flouris, and Dimitris Plexousakis. On belief change and ontology evolution. Technical report, University of Crete, 2006.

[9] Natalya F. Noy and Michel Klein. Ontology evolution: Not the same as schema evolution. *Knowl. Inf. Syst.*, 6(4):428--440, 2004.