International Journal of Database Theory and Application
Vol.7, No.2 (2014), pp.121-130
http://dx.doi.org/10.14257/ijdta.2014.7.2.12

An Improved Approach to Reconciling Multiple Ontology Change Sequences

Ke Zhao[1], Changxian Li[2] and Yannan Sun[2]

[1]*School of Electronics and Information Engineering, Dalian Jiaotong University, Dalian 116028, China*
[2] *School of Motor car Operation and Maintenance Engineering, Dalian Jiaotong University, Dalian 116028, China*
Kezhao1978@163.com

Abstract

Ontology evolution in collateral environments mainly features that multiple users modify the same ontology. All of the ontology change sequences submitted by the users may not be done to the ontology because there are conflicts between them. Unlike previous approach that some ontology change sequences must be removed, this paper focuses on the relationship between ontology changes rather than ontology change sequences. We defined a dependence relationship between two ontology changes and searched all the dependences from different ontology change sequences. We constructed a direct graph for ontology changes with the dependences set. On this basis, we proposed an algorithm based on Traveling Salesmen Problem to find a suitable evolution path. A prototype is implemented and the experiment showed that our approach could keep more ontology changing.

Keywords: *Ontology Change; Ontology Evolution; Traveling Salesmen Problem*

1. Introduction

Ontology Evolution is a timely adaptation of an ontology to the arisen changes and the consistent propagation of these changes to dependent artifacts. It focuses on exploring some methods and technologies to modify ontology on the assumption that the ontology consistency is not broken. But the process of ontology evolution in the collateral environments is quite different from that in the centralized environments. In the collateral environments, many users could modify the same ontology simultaneously, so the ontology change sequences submitted by users may conflict with another, causing ontology evolution dangling. The key of ontology evolution is to resolve those unseen conflicts between ontology change sequences and to find a suitable evolution solution for keeping consistency and aggregating the preferences of multiple users as much as possible. In literature [15], we proposed an approach to reconciling multiple ontology change sequences. But in course of resolving the conflict, a distinct shortcoming is that some ontology change sequences will be discarded, which leads to loss of some useful ontology change made by the discarded ones. So In this paper, we tried to reconcile multiple ontology change sequences rather than to discard them. In addition, we proposed a novel evolution strategy for ontology change sequences to keep ontology changes as more as possible.

This paper is organized as follows. An introductory example for clarifying our problem is shown in Section 2. And the related definitions are shown in Section 3. The whole scheme of ontology evolution and evolution algorithms in collateral environments is given in Section 4.

ISSN: 2005-4270 IJDTA

A prototype system is introduced in Section 5. The related works are mentioned in Section 6 and conclusion and the next work are arranged in the last section.

2. An Example

An abstract ontology is shown in Figure 1, where ellipse nodes are concepts, and every arrow links two concepts – from subconcept to superconcept. Assume thatan engineer *E1* made changes to the ontology in steps: (1) add a concept *new* c_1 to the subconcepts of c_{11}; (2) delete c_{12} and set every subconcept of c_{12} as the subconcepts of c_0. The ontology after *E1*'s changes is shown in Figure 2. Also another engineer *E2* changed the ontology in this way: (3) add a concept *new*c_2 to the subconcepts of c_{12}; (4) delete c_{11} and set every subconcept of c_{11} as the subconcepts of c_0. The ontology made by *E2*'s changes is shown in Figure 3. Unfortunately, care should be taken since, the ontology changes requested by *E1* and those by *E2* can't be performed, regardless of order, to the initial ontology. If *E1* changed the ontology before *E2*, the operation (3) could not be done because c_{12} had been removed by *E1*. If *E2* changed the ontology before *E1*, the operation (1) could not be done because c_{22} had been removed by *E2*. If we changed the ontology in the order of (1)→(3)→(2)→(4) rather than (1)→(2)→(3)→(4) or (3)→(4)→(1)→(2), all operations (1)-(4) could be done consistently and the resulting ontology is illustrated in Fig.4.

This example illustrates sequences of changes made to an ontology may lead to conflicts. In the paper [15], we proposed an algorithm seperating the initial set of the ontology change sequences into multiple different subsets not conflicting mutually. In this approach, one or more ontology change sequences are removed in order to solve the conflicts between all the ontology change sequences. Unlike the previous work, we expect to reconcile multiple ontology change sequences rather than discard them to keep consistency. In addition, we also favor a novel evolution strategy for keeping most of the consistent ontology changes while aggregating the preferences of multiple users, the more, the better.

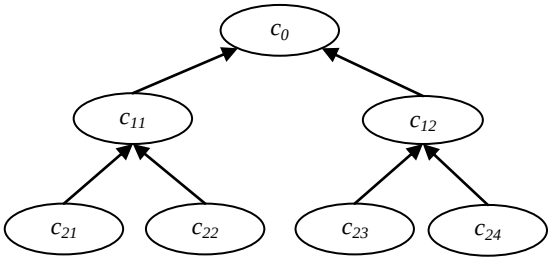

Figure 1. Initial Abstract Ontology

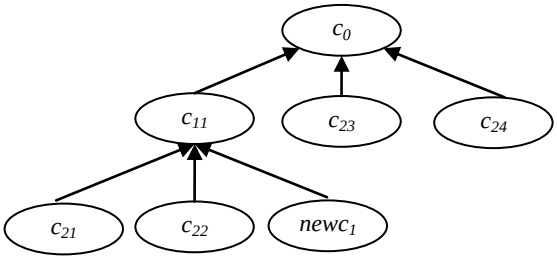

Figure 2. Ontology after *E1*'s Changes

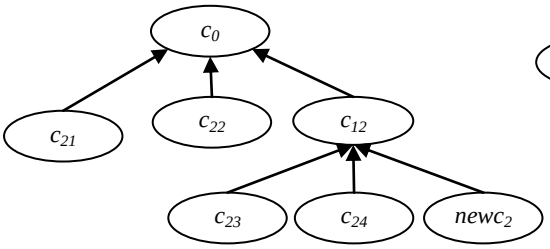

Figure 3. Ontology after *E2*'s Changes

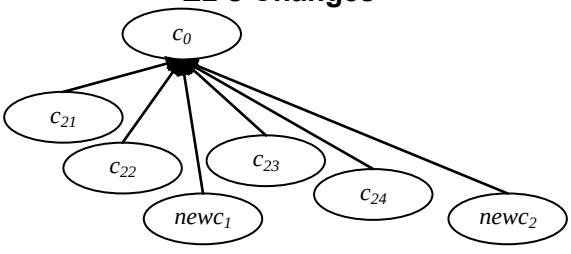

Figure 4. Ontology after Changed in order of (1)→(3)→(2)→(4)

3. Formal Description of Ontology Change

According to Stojanovic[13], an ontology change *oh* is defined as *oh*={*name*, *args*, *preconditions*, *postconditions*}, where *name* is the identifier of this change *c*, *args* is a list of one or more change arguments, *preconditions* denotes a set of assertions that the assertions must be true before the change applied, *postconditions* comprise a set of assertions that it must be true after the change applied. Stojanovic categorizes all the ontology changes into the "Add" ontology changes and the "Remove" ontology changes respectively. Moreover, he pointed out that all the complex ontology changes may be decomposed into a group of "Add" ontology changes and "Remove" ontology changes. To highlight the different type of changes made to a different object, we redefine the ontology change.

Convention 1: the parameter *of children*(*c*) is a set composed of all the subconcepts of the concept *c*.

Convention 2: the parameter *of father*(*c*) is a set composed of all the superconcepts of the concept *c*.

Definition 2: **expression** *exp*(Φ), $\Phi \subseteq O.C$, is defined as

- c is an expression, $c \in \Phi$;
- the parameter , *children*(*c*), is an expression, $c \in \Phi$;
- the parameter , *father*(*c*), is an expression, $c \in \Phi$;
- if E is an expression, $O.C\text{-}E$ is an expression;
- if E_1 and E_2 are expressions, $E_1 \cap E_2$ is an expression;
- if E_1 and E_2 are expressions, $E_1 \cup E_2$ is an expression.

Definition 3: an **addition ontology change** *oh* is defined as

$$oh=\{objects, supcons, subcons\}.$$

where:

- $objects, supcons, subcons \subseteq O.C$.
- $\forall c \in objects$, $\forall pc \in supcons$, $\forall bc \in subcons$, *c* will become the subconcept of *pc* and the superconcept of *bc* after *oh* is done.

Definition 4: a **remove ontology change** *oh* is defined as

$$oh=\{objects, exp(\Phi)\}.$$

where:

- $objects \subseteq O.C$ are concepts to be removed;
- *objects* is the value of *exp*(Φ).

Definition 5: For two ontology change oh_1 and oh_2, oh_1 **precedes** oh_2 and is represented as $oh_1 \rightarrow oh_2$ iff oh_1 must be executed ahead of oh_2。 .

Definition 6: The expression of $ohs=|oh_1,oh_2,\ldots,oh_n|$ is an **ontology change sequence** iff the expression of $oh_1 \rightarrow oh_2 \wedge oh_2 \rightarrow oh_3 \wedge \ldots \wedge oh_{n-1} \rightarrow oh_n$ holds.

Definition 7: For an addition ontology change of *aoh* and a remove ontology change of *roh*, *aoh* **depends on** *roh* , which is represented as $aoh<roh$ iff the formula of $(aoh.supcons \cup aoh.subcons) \cap roh.objects \neq \varnothing$) holds.

4. Description of Our Approach

Our approach includes two steps as illustrated in Figure 5.

Step 1: Given a group of ontology change sequences, we represent them as a directed graph, recognize all the dependences by Definition 7, and refine the directed graph.

Step 2: Solution method for a TSP(Traveling Salesmen Problem) [14] is adopted to compute an evolution path, which maximally and consistently covers all change sequences. By representing an evolution problem as a directed graph, we will observe that an evolution path is analogous to the solution of a TSP. So evolution problem is in nature a variance of the TSP, some previous approaches for solving TSP can be applied to solve the problem of our interest.

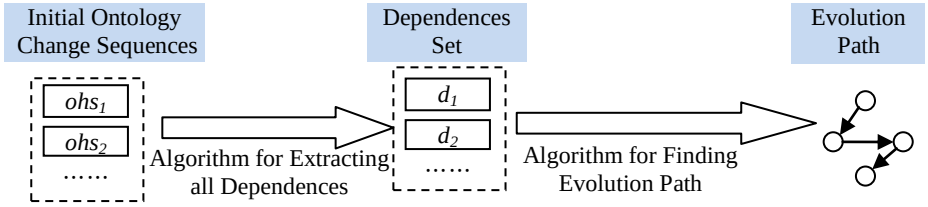

Figure 5. The Whole Scheme of our Approach

4.1. Algorithm for Extracting all Dependences

Algorithm 1 is used to find all the dependences from a group of ontology change sequences *OHS*. By algorithm 1, all the dependences about that example in section 2 are $DEP=\{oh_{11}<oh_{12},\ oh_{12}<oh_{13},\ oh_{13}<oh_{14},\ oh_{14}<oh_{15},\ oh_{21}<oh_{22},\ oh_{22}<oh_{23},\ oh_{23}<oh_{24},\ oh_{24}<oh_{25},\ oh_{11}<oh_{22},\ oh_{21}<oh_{12}\}$.

Algorithm 1:findDependencs (*OHS*)
Input: *OHS*, a group of ontology change sequences $OHS=\{ohs_1,ohs_2,\ldots,ohs_n\}$
Output: $\mathcal{DEP}$, a group of constraints

```
1.  DEP←∅;
2.  i←1;
3.  WHILE(i<=n)
4.    Let ohs_i be |oh_1,oh_2,...,oh_m|;
5.    j←1;
6.     WHILE(j<=m-1)
7.       DEP←DEP∪{oh_j<oh_{j+1}};
8.       k←1;
9.       WHILE(k<=n)
10.         IF k≠i and ∃oh∈ohs_k, oh_j<oh
11.              DEP←DEP∪{oh_j<oh};
12.           k←k+1;
13.      j←j+1;
14.   i←i+1;
15. RETURN DEP
```

4.2. Algorithm for Finding Evolution Path

DEP contains all the dependences between two ontology changes. All the ontology changes in a group of ontology change sequences are not performed unless any one of DEP is met. So, we should find an evolution path, which covers all the ontology

changes while every dependence of DEP is met. In nature, to find an evolution path is to find the solution of a TSP [14]. Next we briefly introduce the TSP.

4.2.1. TSP(Traveling Salesmen Problem)

Let $G=(V,E)$ be a weighted directed graph, where $V=\{v_1,v_2,\ldots,v_n\}$ is a set of all the vertices and $E=\{e_{i,j}|v_i,v_j\in V,\ i\neq j\}$ is a set of all the edges, represented by a $n\times n$ matrix. Let $d_{i,j}$ be the distance between v_i and v_j, where $d_{i,j}>0$ and $d_{i,j}\neq\infty$ and $d_{i,j}=d_{j,i}$ holds. A TSP is to find a path in G such that

$$z = \min \sum_{i} \sum_{j} d_{i,j} x_{i,j} \quad (1)$$

Where

- $$x_{i,j} = \begin{cases} 1 & e_{i,j} \text{ is on the optimalizing path} \\ 0 & e_{i,j} \text{ is not on the optimalizing path} \end{cases} \quad (2)$$
- $$\sum_{i=1}^{n} x_{i,j} = 1,\ j = 1,2,\cdots,n \quad (3)$$
- $$\sum_{j=1}^{n} x_{i,j} = 1, i = 1,2,\cdots,n \quad (4)$$
- $$\sum_{i,j\in S}^{n} x_{i,j} \leq |S| - 1, 2 \leq |S| \leq n - 2 \quad (5)$$

Formula (1) is an object function of the TSP. Formulas (2), (3) and (4) demand that every vertex has only one incoming edge and only one outgoing edge, too. Formula (5) demands that every vertex is visited only once and no cycle is included in the final path. TSP is a classic problem and there are many approaches to solve it [14].

4.2.2. TSP for Finding Evolution Path

In order to find the evolution path for our problem, we should rephrase the problem of ontology evolution as a TSP by build a directed graph upon ontology change sequences and their dependences, and the TSP also needs to be revised.

Firstly, we introduce the construction of the directed graph $G=(V,E)$, which is done by Algorithm 2 below. The graph of our running example created by the algorithm is shown in Figure 6.

Algorithm 2: buildGraph(*OHS*, *DEP*)
Input: *OHS*, a group of ontology change sequences; *DEP*, a group of dependences.
Output: $\mathcal{G}(\mathcal{V},\mathcal{E})$, a directed graph of ontology changes.

```
1.  V←∅;
2.  E←∅;
3.  FOR EACH ohs in OHS
4.    FOR EACH oh in ohs
5.      V←V∪{oh};
6.  FOR EACH oh1,oh2 in V
7.    IF oh1≠oh2
8.      E←E∪{e1,2}∪{e2,1};
9.  FOR EACH oh1<oh2 in DEP
10.   E←E-{e2,1};
11. RETURN G(V,E)
```

Secondly, TSP is revised as TSP*

$$path = (V, E_p) \quad (6)$$

Where

- $E_p \subseteq E$ (7)
- $\sum_{i=1}^{n} e_{i,j} = 1,\ j = 1,2,\cdots,n \quad e_{i,j} \in E_p$ (8)
- $\sum_{i=1}^{n} e_{i,j} = 1,\ j = 1,2,\cdots,n \quad e_{i,j} \in E_p$ (9)
- $\sum_{i,j \in S}^{n} x_{i,j} \le |S| - 1, 2 \le |S| \le n - 2$ (10)

Formula (6) is an object function of TSP* and it returns an evolution path. Formulas (7), (8) and (9) demand that every vertex has only one incoming edge and only one outgoing edge. Formula (10) demands that every vertex is visited only once and no cycle is included in the evolution path.

In reality, the number of evolution paths can be 0, 1 or more than one. For the example shown in Section 2, an evolution path from oh_{11} to oh_{15} is shown in Figure 7. In order to find an evolution path as quickly as possible, we applied the hybrid GA–PSO–ACO algorithm to TSP*[14], which was proposed in literature [14].

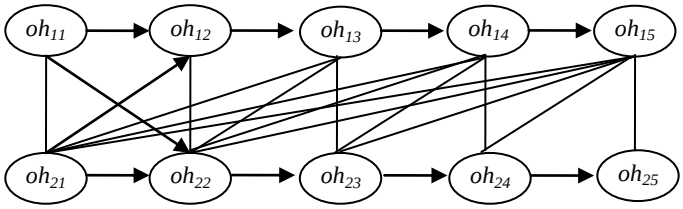

Figure 6. Graph of Example in Section 2

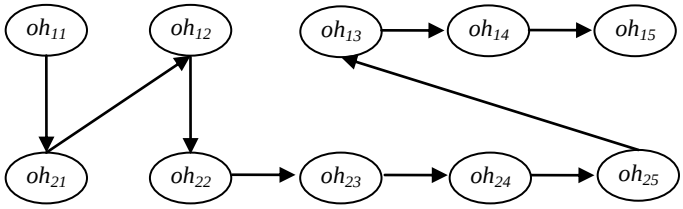

Figure 7. An Evolution Path of the Example in Section 2

5. Implementation of Prototype for our Approach

We implemented a prototype based on the approach described above. The prototype is for OWL Ontology in JAVA. The architecture of prototype is shown in Figure 8. The whole system is composed of six components and an OWL ontology. Application GUI is an interface for users. By Application GUI, multiple users may submit ontology change sequences to the system. In Checker, the validity of expression is checked. If expression is not correct, an error report is outputted to Application GUI. Parser is responsible to parse ontology change sequence into another format, which is fitter for Dependence Search

Component. Dependence Search Component is implementation of Algorithm 1. It will output all the dependences to Evolution Path Finding Component. Evolution Path Finding Component is implementation of algorithm 2 and GA–PSO–ACO. It will usually output an evolution path to Evolution Component. If there is no evolution path is found, an error report is outputted to Application GUI. Evolution Component can apply ontology change sequence from evolution path to OWL Ontology. The changed Ontology will be shown in Application GUI.

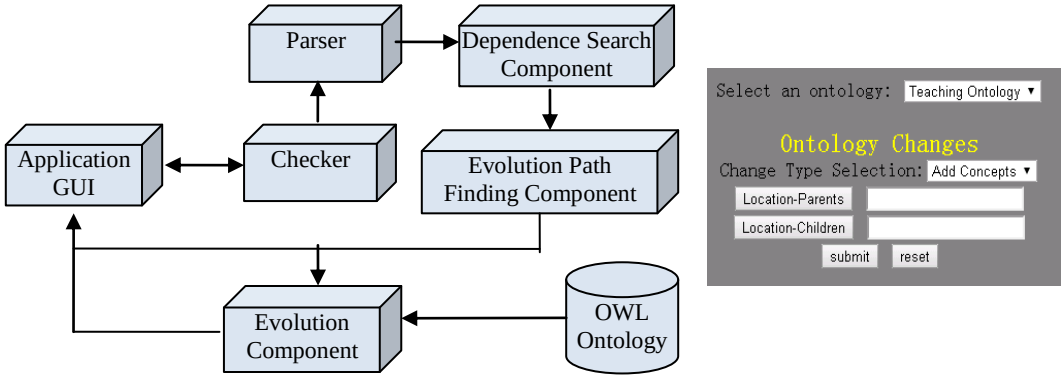

Figure 8. Architecture of Prototype and Screenshot of Input Page

5.1. Experiments with Prototype

We have tested the current implementation of the prototype system on real ontologies from different domains. The whole experiment is carried out on a PC with 2Ghz CPU 512 MB memory under Windows xp.

In the following, we briefly report experiments performed for detecting changes in the concept hierarchy of the following two ontologies. The first ontology contains about 80 concepts and 60 relations. The second ontology has about 180 classes and around 70 properties. We showed the results of experiment in Table 1. *OHS* is an abbreviation of ontology change sequence. *OHN* is an abbreviation of ontology changes. *Ain17* is an abbreviation of approach in literature [15]. From Table 1, we submit 10 ontology change sequences to OJKL. There are 45 ontology changes in total. By our approach, we search 43 dependences and find 11 evolution paths finally. There are at least 2 ontology changes sequences are removed from *OHS* when we apply the approach in literature [15] to OJKL. In a similar way, we submit 10 ontology change sequences to SAIL. There are 52 ontology changes in total. By our approach, we search 50 dependences and find 13 evolution paths finally. There are at least 3 ontology changes sequences are removed from *OHS* when we apply the approach in literature [15] to SAIL.

Table 1. Result of Experiments on Ontologies OJKL and SAIL

Ontologies	Concepts	OHS	OHN	Dependences	Paths	Ain17
OJKL	80	10	45	43	11	2
SAIL	180	10	52	50	13	3

6. Related Work

By far, lots of works have been done on ontology change disposal [1-6]. And they may be classified into some based on machine learning, some based on logical reasoning and others based on belief revision. Based on lifecycle of ontology evolution, researchers mainly focus on finding and managing ontology changes.

Finding ontology changes is responsible to diagnose inconsistency of ontology and reason or compute matching ontology changes which are used to repair ontology. Technologies from software code refactoring are used to find structure inconsistency [7, 8]. Properties of concepts are considered as branch sentence fragments. And concepts are considered as the whole branch sentences. So code refactoring technology is used to find inconsistency of ontology such as Single Subconcept, Too Many Subconcepts and Concepts having not any property. In addition, approach based on graph is used to Cycle Concepts [13]. Also, reasoning technology is used to find inconsistency of certain logical constraint between concepts[12]. Machine learning technology is used to mine some potential inconsistency such as Unsatisfied Concepts, Concepts Having Too Instances, *etc*. For example, clustering algorithm and Formal Concept Analysis [10] are used to find new concepts and new hierarchies according to distribution of instances.

In order to make ontology reach to consistency, managing ontology changes is responsible to decide evolution strategy. After found ontology changes are submitted to ontology, ontology will properly reach to a new inconsistency. Parsia [9] adds axioms to ontology one by one until a maximum consistent sub-ontology is got. Analogously, he deletes axioms from ontology until a minimum inconsistent sub-ontology is got. By search maximum consistent sub-ontology and minimum inconsistent sub-ontology, the extra, but necessary, ontology changes are found. Based on the similarity between ontology evolution and knowledge-base update, belief revision are used to compute ontology changes. Because belief revision [4, 11] is studied for a long time, some related technologies are easily applied to ontology. But the gap between closed hypothesis, the keystone of belief revision, and open one, the keystone of ontology evolution, need be filled up.

7. Conclusion and Future Work

In order to change an ontology concurrently and accurately with less or even without intervene of humans, we proposed an approach to reconciling multiple ontology change sequences in collateral environments. In contrast to previous approach, we introduced expressions to represent ontology change, which declaratively depicts ontology changes. In addition, our approach can keep more consistent ontology changes while aggregating the preferences of multiple users than previous approaches. Also a prototype is implemented to validate the proposed approach.

References

[1] M. Klein, Academisch Proefschrift, Michel Christiaan, Alexander Klein, and J. M. Akkermans. Change management for distributed ontologies. Technical report. **(2004)**.
[2] P. Haase, F. Van Harmelen, Z. Huang and H. Stuckenschmidt, “A framework for handling inconsistency in changing ontologies”, Springer, Proceedings of the 4th international conference on The Semantic Web, **(2005)**, pp. 353-367.
[3] A. Kalyanpur, B. Parsia, E. Sirin and B. Cuenca Grau, “Repairing unsatisfiable concepts in owl ontologies”, In 3rd European Semantic Web Conference, **(2006)**, pp. 170-184.
[4] N. Foo, “The Ontology Revision”, Proceedings of the 3rd International Conference on Conceptual Structures, **(2011)**, pp. 16-31.

[5] L. Stojanovic, "Methods and Tools for Ontology Evolution", PhD thesis, University of Karlsruhe, Germany, **(2004)**.
[6] P. Haase and L. Stojanovic, "Consistent evolution of owl ontologies", 2rd European Semantic Web Conference, **(2005)**, pp. 182-197.
[7] M. Fowler, K. Beck, J. Brant, W. Opdyke and D. Roberts, "Refactoring: improving the design of existing code", **(2002)**.
[8] L. Tokuda and D. Batory, "Automating three modes of evolution for object-oriented software architecture", Proceedings of the 5th conference on object oriented technologies and Systems, **(2009)**, pp. 189-202.
[9] B. Parsia, E. Sirin and A. Kalyanpur, "Debugging OWL ontologies", Proceedings of 14th International Conference on World Wide Web, **(2005)**, pp. 268-293.
[10] A. Maedche and V. Zacharias, "Clustering ontology-based metadata in the Semantic Web", Proceeding of the 6th European conference on principles and practice of knowledge discovery in databases, **(2012)**, pp. 348-360.
[11] G. Flouris, D. Plexousakis and G. Antoniou, "Evolving Ontology Evolution", Proceedings of SOFSEM10, **(2010)**, pp. 14-39.
[12] L. Stojanovic, "User-driven Ontology Evolution Management", Proceedings of European Conference Knowledge and Management, **(2012)**, pp. 285-300.
[13] L. Stojanovic, A. Maedche, N. Stojanovic and R. Studer, "Ontology Evolution as Reconfiguration-Design Problem Solving", Proceedings of the 2nd international conference on knowledge capture, **(2003)**, pp. 162-171.
[14] W. Deng, R. Chen, B. He, Y. Q. Liu, L. F. Yin and J. H. Guo, "A novel two-stage hybrid swarm intelligence optimization algorithm and application", Soft Computing, vol. 16, no. 10, **(2012)**, pp. 1707-1722.
[15] Y. Q. Liu, R. Chen, J. Gao and H. Yang, "A Conflict-Resolving Approach to Ontology Evolution in Open Environments", Engineering Intelligent Systems, vol. 18, no. 3-4, **(2010)**, pp. 223-231.

Authors

Ke Zhao, lecturer, received the master degree in Control Theory and Control Engineering from Inner Mongolia University of Technology, Hohehot, China, in 2006. The main research directions: Semantic Web, Ontology, Intelligent Control.

Changxian Li, associate professor, received the doctor degree in Control Theory and Control Engineering from Zhejiang University, Hangzhou, China, in 2003. The main research directions: Semantic Web, Ontology, Network of high-speed EMU control technology.

Yannan Sun, lecturer, received the doctor degree in Control Theory and Control Engineering from Dalian University of Technology, Dalian, China, in 2007. The main research directions: Semantic Web, Intelligent Control.

