International Journal of Control and Automation
Vol.8, No.3 (2015), pp.387-404
http://dx.doi.org/10.14257/ijca.2015.8.3.38

An Approach to Detect Conflicts for Collaborative Evolution of Medicine Ontology

Song Yingjie[1,2,3] , Zhang Bin [1,2] and Mao Yanyan[1,2]

[1] Key Laboratory of Intelligent Information Processing in Universities of Shandong (Shandong Institute of Business and Technology), 264005 Yantai, China
[2] School of Computer Science and Technology, Shandong Institute of Business and Technology , 264005 Yantai, China
[3] Artificial Intelligence Key Laboratory of Sichuan Province, Sichuan University of Science and Engineering, 643000 Zigong, China
tiantianyingjie@gmail.com (Corresponding author)

Abstract

In the collaborative evolution of biomedical ontology, participants are a sizeable, balanced mix of scholars and physicians, all of whom are experienced in the biomedical domain and who represent diverse viewpoints, experiences, and backgrounds. In the field of collaborative evolution of biomedical ontology on large-scale ontology, there exists inevitable conflicts, which may cause the inconsistent ontology. In this paper, a new method to detect conflicts in ontology evolution is presented, which classifies conflicts as three groups: internal inconsistencies conflicts in change sequence, direct conflicts between the sequences and Inconsistent conflict between the sequences. For different conflict, high effective detecting algorithms are presented with evaluation. Before the conflicts detecting, semantic extended rules are employed to depict the evolution requirements of the participants. In particular, we discuss the situation where maximum consistent changing subsequence is needed if there are inconsistent conflicts between changing subsequences. We also show how detecting algorithms could be taken in the collaborative evolution of medicine ontology. As a result, internal and mutual inconsistencies can be detected from change sequences. And if there are conflicts between the sequences, the algorithm will provide the maximum consistent changing subsequence as the evolution basis. The designed experiments verify our approach and achieve the expected results.

Keywords: *Collaborative Ontology Evolution; Conflicts Detecting; Semantic Conflicts*

1. Introduction

In the network, there are many independently developed and different structure biomedical databases, which are used for experts in biomedical to retrieve information and find out the potential knowledge. Different databases have a great difference with terminology, semantic and structure, all of these limit the knowledge sharing and hinder the effective search of person and computer. The Biomedical Ontology Development is working to develop a sound biomedical ontology to enable the various knowledge processing applications to communicate with one another [1]. It is mainly focus on the representation and (re-)organization of biomedical terminologies. Physicians developed their own specialized languages and lexicons to help them store and communicate general medical knowledge and patient-related information efficiently. Biomedical information systems, on the other hand, need to be able to communicate complex and detailed medical concepts (possibly expressed in different languages) unambiguously. For example, the Biological database Bio2RDF [2, 3] uses the Semantic web technologies to provide interlinked life science data. It integrated more than 40 information resources of biomedicine, such as GenOntology, OMIM, PubMed, GenID, UniProt and so on. The EU FP 7 Large-Scale Integrating Project LarKC [4] developed by OntoText company, is to build an integrated platform for semantic computing on a scale well beyond what is currently possible. It integrated more than 25 information resources of biomedicine, and contains more than 4 billion. This is obviously a difficult task and requires a profound analysis of the structure and the concepts of medical terminologies. The size of medicine ontologies on the Web is growing exponentially in recent years (For example, the Gene Ontology contains 333960 terms, where there are 20632 terms from Biological Process Ontology, 2819 terms from Cellular component and 9013 terms from Molecular function, and 1496 Obsolete terms by the end of April 8, 2012. Moreover, GO have annotated near 90 million Genes and Gene Products). And knowledge about the biomedical is constantly accumulating and changing. Take the GO as an example, since 1998 until now, it developed by an international consortium include full-time editors and many other part-time editors at databases, and developers made modifications continuously. The suggested changes initially submitted by email and Moved to an online tracking system when this became unmanageable. Larger questions about the higher ontology structure changing remain unresolved and makes some items impossible to close. So one of the most import problems is biomedical ontology evolution, which is the problem of modifying an medicine ontology in response to a certain change in the domain or its conceptualization. Several reasons are likely to lead to medicine ontology evolution. The medicine ontology may need to change simply because the medical circle has changed [5], we may need to change the perspective under which the medical circle is viewed [6], or we may discover a problem in the original conceptualization of the medicine, and so on.

Further, the process of the collaborative evolution of biomedical ontology takes place in a distributed environment. The ontology evolution is a big challenge in collaborative environment, in which conflicts are often a problem. In the collaborative medicine ontology evolution, automated conflicts detection is essential. Collaborative ontology evolution is a social process [7]. Since its participants have slightly different views on the same domain, a harmonization effort requires discussing the resulting ontology. In the collaborative ontology evolution, participants are a sizeable, balanced mix of scholars and physicians, all of whom are experienced in the medicine domain and who represent diverse viewpoints, experiences, and backgrounds. In the field of medicine ontology evolution on large-scale ontology, there exists inevitable conflicts, which may cause the inconsistent ontology. Traditional methods, such as SVN and so on, are ineffective if collaborative ontology evolution with distributed participants who are lacking communications. The traditional method for conflicts is using mechanisms of locking to prevent the occurrence of modification of the same document. For the existence of complicated relations and semantic, a basic change operation may exerts an influence on the other ontologies and their applications, so the traditional one is inappropriate for solution of ontology evolution. However, the multi-changing sequences of ontology evolution indicate different change requests, and the target of evolution is to meet more requests.

Ontology Evolution [8] is the timely adaptation of an ontology to the arisen changes and the consistent propagation of these changes to dependent artifacts. Medicine ontology evolution is one of the key problems facing ontology users today. Assessing and comprehending changes to, and between ontologies has been a large problem within the ontology community for some time. Adapting ontologies to meet new the advancements of medical science requirements involves understanding various sections of ontologies and the changes made thereafter. However, currently available methodologies to support ontology evolution for Medicine focus their attention mostly on the development of static ontologies, which is a complex, expensive and time-comsuming process [9] by knowledge engineers and a small number of physicians. In recent years, some methods and tools [10] for collaborative ontology construction are also proposed which meet the requirements of public ontologies having relevance and value to a broad audience better. For these systems, it is essential to keep knowledge in consistency.

Protégé [11] is a free, open-source java-based platform that provides a growing users community with ontologies. But it not contains reasoning function. OntoEdit [12] is a development environment for ontology design and maintenance. It support multilingual development, and the knowledge model is related to frame-based languages. Every plug-in provides other feature to deal with the requirements an ontology engineer has.

In the previous works [13], we have arrived at a solution via scientific investigation for ontologes consistency reasoning. We propose a new approach to interpreting ontologies document in a lightweight modeling language for software design, Alloy, which is used to provide a non-standard reasoning service for the verification of ontologies. Motivated by the challenges of medicine ontology evolution and based on the previous works, in this paper, we present a methodology for ontology evolution, by focusing on the conflicts of multi-participant. In particular, we discuss the situation where maximum consistent changing subsequence is needed if there are inconsistent conflicts between changing subsequences. The experiments show the conflicts can be detected.

The paper is organized as follows. First, the overview section introductions the background and significance of the issue, and some tools of ontology evolution. Section 2 addresses the concepts and terms in ontology evolution. Section 3 proposes the algorithms to detect conflicts. Section 4, we give out the Similarity formulas. Section 5 analyzes the time complexity of the formula. And then we show experiments with evaluation. The last section is our conclusion.

2. Formal Description of Ontology Change

Definition 1: **Medicine Ontology $\mathcal{O}$** is defined as a 5-tuple:

$$\mathcal{O}:=\{C, A^C, R, H, A\},$$

where:

- C is the set of medicine concepts.
- A^C is the set of attributes, there are many-to-one relationships between attributes and concepts.
- R is the set of non-hierarchical relationships between concepts. The function *Rel*: $R \subseteq C \times C$ maps the relation identifiers to the actual relationships.
- H is the set of hierarchical relationships between concepts. The function is $H \subseteq ((C \times C) \cup (R \times R))$. If there are two concepts c_1, c_2, and $(c_1, c_2) \in H$, then we say the class c_1 is defined as a subclass of class description c_2. R and H are both the subset of the cartesian product of C.
- A is the set of axioms.

For simplicity, let $\mathcal{O}.\ C$ be the shortened form of concepts C of ontology $\mathcal{O}$. Its attributes are $\mathcal{O}.\ A^C$, relationships are $\mathcal{O}.\ R$, hierarchical relationships are $\mathcal{O}.\ H$, and axioms are $\mathcal{O}.\ A$. The other definitions are similar.

Definition 2. The **Property** P is defined as a 3-tuple:

$$P{:=}\{name,\ Domain,\ Range\},$$

where:

- *name* is the naming of the property.
- *Domain* asserts that the subjects of such property statements must belong to the class extension of the indicated class description, $d(P){:=}\{d|\exists r,(d,r)\in P\}$.
- *Range* asserts that the values of this property must belong to the class extension of the class description or to data values in the specified data range, $r(P){:=}\{r|\exists d,(d,r)\in P\}$.

Definition 3: The **Ontology Changes** *Och* is defined as a 4-tuple:

$$Och_{user}{:=}\{id,\ operation,\ subject,\ constraint\},$$

where:

- *id* is used to as a unique identification of the sequencing in changing sequence.
- *operation* is the basic type of *Och*, which are {*Add*, *Delete*, *Modify*}.
- $subject \subseteq \mathcal{O}.\ C \cup \mathcal{O}.\ P$, means that the subjects of operation are {*Concept*, *Property*}.
- $constraint \subseteq \mathcal{O}.\ C \cup \mathcal{O}.\ P$, is used to indicate depended concepts and properties.

The following table shows ontology change operations which this paper involved. These operations refer to the atomic ones.

Table 1. The Atomic Change of Ontology

Operation	description	extension
AddConcept(C_1, C_2)	Add concept C_1 as the subconcetp of C_2	Not delete C_2
DeleteConcept(C_1, C_2)	Delete concept C_1 whose supconcetp is C_2	C_1 is subconcept of C_2
ModifyConcept(C_1)	Modify concept C_1, contain renaming and the modification of its attributes	
AddProperty(P_1, $\{C_1, C_2\}$)	Add property P_1, whose *domain* is C_1 and *range* is C_2	Not delete C_2 and C_2
DeleteProperty(P_1)	Delete property P_1	
ModifyProperty(P_1, $\{C_1, C_2\}$)	Modify property P_1, whose new *domain* is C_1 and new *range* is C_2	Not delete C_2 and C_2

In the table1, there are three types atomic ontology changes, which are *Add*, *Delete* and *Modify*. Hürsch [14] divide atomic changes into *Add* and *Delete*. We add *Modify* as the third one. For that, the operation *Modify* can be broken down as delete the modified concept first and then add the objective concept outwardly. However, in the actual evolution the process is inadvisable. For example, under the assumption of *Modify* is nonexistent, if we want to rename the concept C_1 as C_1'. First, the concept C_1 should be deleted, if there is a subconcept C_2 of C_1, it will also be handled, which may be deleted too or as subconcepts of C_1 's supconcept C_0 . Then C_1' will be added to replace C_1. Obviously, the hiberarchy of the ontology has been destroyed and the process can not satisfy the requirement.

The last column in the table is extensions for changing operations, it can also be understood as the dependent condition when the changes take place. It reveals the implicit semantics. For example, the add operation *AddConcept*(C_1, C_2) in the first row has extension that the concept C_2 is not allowed to be deleted. We can see from the description of the operation that the semantic of the operation is to add the concept C_1 as a subconcept of C_2. If no extension, there may be a operation of deleting the concept C_2, and if the operation of delete concept C_2 take place earlier than the operation of adding operation *AddConcept(*C_1*,* C_2*)*, then the added concept C_1 will become an isolated concept in the ontology, which is unreasonable. The other extensions are the similar. However, not all operations need extension, e.g. the operation of modifying the concept *ModifyConcept(*C_1*)* in third row and the operation of deleting property *DeleteProperty(*P_1*)* in the fifth row.]

```
Add-extension rules:
IF  $Och_i = Add\,(subject, constraint)$ and $constraint.subject \neq \emptyset$
   IF  $subject.type=Class$ and $subject.\mathrm{parent} \neq \emptyset$
      THEN  extended $Och_i$ as $\{(Och_i, Och_i^1) \mid Och_i = Add\,(subject, constraint), Och_i^1 = \neg Delete\,(subject.\mathrm{parent})\}$;
   IF  $subject.type=Property$
      THEN  extended $Och_i$ as $\{(Och_i, Och_i^1, Och_i^2) \mid Och_i = Add\,(subject, constraint), Och_i^1 = \neg Delete\,(subject.domain), Och_i^2 = \neg Delete\,(subject.range)\}$;
   IF  $subject.type=Axiom$
      THEN  extended $Och_i$ as $\{(Och_i, Och_i^1) \mid Och_i = Add\,(subject, constraint), Och_i^1 = \neg Delete\,(subject.domain)\}$;
```

For the different operation in the table the extension rules are given as follows.

The understand from the *Add*-extension rules is that if add a concept C_1 as the subconcept of C_2, then the supconcept C_2 is not allowed to be deleted. Similarly, if we want to add a property *P*, then the domain and the range of the property is not allowed to be deleted too. When adding a axiom *A*, the relative object (concept, property or instance) can't be deleted. In summary, the dependency of *Add* operation is should not be deleted.

The *Delete*-extension rules contain three parts. The first one is that the deleted subject type is class. In such a case, the properties of the deleted class should be deleted too. Further, if the deleted class has subclasses, we replace the deleted class's supclass as its subclasses's supclass. If the deleted subject type is property, then its subproperties are also be deleted.

```
Delete-extension rules:
IF  $Och_i = Delete\,(subject, constraint)$
IF $subject.\mathrm{type=Class}$
   THEN  extended $Och_i$ as $\{(Och_i, Och_i^i) \mid Och_i = Delete\,(subject), Och_i^1 = Delete(subject.Property)\}$;
IF $subject.\mathrm{type=Class}$ and ($subject.children \neq \emptyset$ and $subject.parent \neq \emptyset$)
THEN  extended $Och_i$ as $\{(Och_i, Och_i^1, Och_i^2) \mid Och_i = Delete\,(subject), Och_i^1 = Delete(subject.Property), Och_i^2 = Modify\,((subject.children).parent = subject.parent)\}$;
        IF $subject.\mathrm{type=Property}$ and $subject.subProperty \neq \emptyset$
           THEN  extended $Och_i$ as $\{(Och_i, Och_i^i) \mid Och_i = Delete\,(subject), Och_i^1 = Delete(subject.subProperty)\}$;
```

```
Modify-extension rules:
IF  $Och_i$ = Modify (subject, constraint)
    IF subject.type=Property.domain||Property.range
        THEN  extended $Och_i$ as $\{(Och_i, Och_i^1)|Och_i$ = Modify (subject, constraint), $Och_i^1 = \neg$Delete (constraint.subject)$\}$;
    IF subject.type= Axiom
        THEN  extended $Och_i$ as $\{(Och_i, Och_i^1)|Och_i$ = Modify (subject, constraint), $Och_i^1 = \neg$Delete (constraint.subject)$\}$
```

For the *Modify*-extension rules, if the domain (or range) of a property is modified, the new domain (or range) is not allowed to be deleted. It is the similar with the axioms's modification.

Definition 4: The **Ontology change sequence** is defined as

$$Chs = < Och_1, Och_2, \ldots Och_n >,$$

Which is composed of a succession of Ontology Changes Och_1, Och_2, …Och_n. The adjacent changes are separated by ",". If there are two changes Och_i and Och_{i+j}(j>=1) , Och_i is the predecessor of Och_{i+j}, and the order of the same change sequence can't be changed.

Definition 5: The **Ontology diagram** is defined as follows:

$$G{:=}\{V,E\},$$

where

- V is the set of vertexes. Each $v \in V$, it is used to represent a class of the OWL ontology. $V = \mathcal{O}.C \cup (\forall Chs).C$. The vertexes are real line ellipses in the diagram. Meanwhile, the changed concepts are drawn as dotted line ellipses. Using dotted line ellipses add "+" to represent the added concepts and dotted line ellipses add "-" to represent the deleted concepts.
- E is the set of directed edges. $E=\{(v,u)|v,u \in V$, and (v,u) is the edge from v to u, the property of the ontology defines the semantic of the edge (v,u), which is used to represent the property whose domain is the class represented by v and range is the class represented by u.$\}$. There are two categories directed edges, one is Property-edge and another is Inheritance-edge. When $P \in C.R$, the edge is Property-edge, and sign $P.name$ to the edge. When $P \in C.H$, the edge is Inheritance-edge, and the edge with no sign. In fact, when $P \in C.H$, $P.name$ is *subclass* for all. The changed properties should be represented with dotted arrows, and then the dotted arrows with "+" are used to represent added properties and the dotted arrows with "-" are used to represent the deleted properties.

The Figure 1(a), (b) shows the source ontology *biological_process* and the changed ontology respectively. The vertexs of the former corresponding to the concepts set of ontology $\mathcal{O}.C$={*biological_process*, *response_to_external_stimulus*, *response_to_endogenous_stimulus*, *response_to_biotic_stimulus*, *tropism*}, the concept *biological_process* is the supconcept of {*response_to_external_stimulus*, *response_to_endogenous_stimulus*, *response_to_biotic_stimulus*}, the property $Property_1$ is from *response_to_external_stimulus* to *tropism*. (b) on the right is the result of (a)'s changing after the sequences {Chs_1, Chs_2, Chs_3} have acted on it. Chs_1={*DeleteProperty*($Property_1$), *DeleteConcept*(*tropism*, *response_to_external_stimulus*)},
Chs_2={*AddConcept*(*response_to_extracellular_stimulus*,

response_to_external_stimulus), *AddProperty* (*Property*$_2$,{ *response_to_external_stimulus*, *response_to_extracellular_stimulus* })}, *Chs*$_3$={*ModifyConcept*(*response_to_biotic_stimulus*)}.

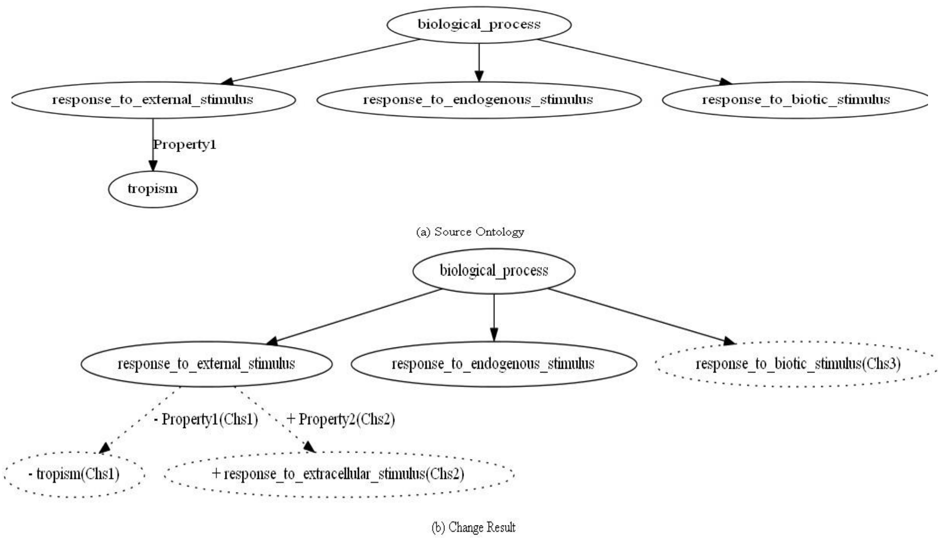
biological_process
response_to_external_stimulus
response_to_endogenous_stimulus
response_to_biotic_stimulus
Property1
tropism
(a) Source Ontology
biological_process
response_to_external_stimulus
response_to_endogenous_stimulus
response_to_biotic_stimulus(Chs3)
- Property1(Chs1)
+ Property2(Chs2)
- tropism(Chs1)
+ response_to_extracellular_stimulus(Chs2)
(b) Change Result

Figure 1. Ontology Diagram

Definition 6: Two operations Och_i and Och_{i+j} from the same sequence (Och_i, $Och_{i+j} \in Chs$) are said to have **Dependency** relation, if and only if Och_i, Och_{i+j} satisfy Och_i.*operation*="*Add*" And Och_{i+j}.*constraint*$\cap Och_i$.*subject*$\neq\emptyset$, we denote this by Dependency(Och_i,Och_{i+j}).

The changes of the same changing sequence are likely to have Dependency relation. As is shown in Figure 2. Changing sequence such as *Chs*={ *AddConcept*(*response_to_external_stimulus*, *biological_process*), *AddConcept*(*response_to_extracellular_stimulus*, *response_to_external_stimulus*)}. Och_1,Och_2 $\in$ *Chs*, Och_1.*operation*="*Add*" And Och_2.*constraint*$\cap Och_i$.*subject*={ *response_to_external_stimulus* }. According to the definition 7, we believe that Och_2 depends on Och_1.

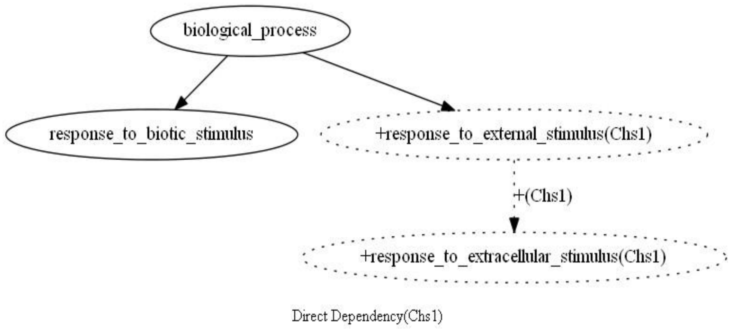
biological_process
response_to_biotic_stimulus
+response_to_external_stimulus(Chs1)
+(Chs1)
+response_to_extracellular_stimulus(Chs1)
Direct Dependency(Chs1)

Figure 2. The Dependency Relation

Lemma 1. Two operations Och_i and Och_{i+j} from the same sequence (Och_i, $Och_{i+j}\in Chs$) satisfy Dependency(Och_i,Och_{i+j}), the order of execution can't be reversed.

Prove: On the premise that Och_{i+j} depends on Och_i. Assume that Och_{i+j} is executed earlier than Och_i, due to $Och_{i+j}.constraint \cap Och_i.subject \neq \emptyset$, in other words, the precondition of Och_{i+j} 's execution is not satisfied, which restrict the happen of Och_{i+j}. So the assumption is false.

Lemma 2. There is no circular dependency in the same changing sequence. That is to say, if $Och_1, Och_2, Och_3 \in Chs$, then the three dependency relations, which are Dependency(Och_1, Och_2), Dependency(Och_2, Och_3) and Dependency(Och_3, Och_1), cannot be simultaneously true.

Prove: Assume that $\exists Och_1, Och_2, Och_3 \in Chs$, and the three dependency relations, which are Dependency(Och_1, Och_2), Dependency(Och_2, Och_3) and Dependency(Och_3, Och_1), is simultaneously true. According to the definition 7, we can come to the following conclusion: $Och_1.opertation=Och_2.opertation=Och_3.opertation=$ "*Add*", $Och_2.constraint \cap Och_1.subject \neq \emptyset$, $Och_3.constraint \cap Och_2.subject \neq \emptyset$ and $Och_1.constraint \cap Och_3.subject \neq \emptyset$, by the Ontology Changes of definition 3 and the operations in Table 1, the type *Och.constraint* is class, further, the operations which is depended on, can only be *AddClass*. In the assumption, the three operations are pairwise dependent, the three operations are all *AddClass*. In addition, $Och_2.constraint \cap Och_1.subject \neq \emptyset$, $Och_3.constraint \cap Och_2.subject \neq \emptyset$ and $Och_1.constraint \cap Och_3.subject \neq \emptyset$, it can be inferred that $Och_2.subject.supClass=Och_1.Subject$, $Och_3.subject.supClass=Och_2.Subject$, $Och_1.subject.supClass=Och_3.Subject$. Thus, we arrive at the conclusion that there is loop circle in the hierarchy of the ontology, which violate the rules of ontology construction. So we can deduce that the assumption is false.

3. Collaborative Evolution Conflicts Detection

Three types of collaborative evolution conflicts, which are Internal Inconsistency within the changing sequence, Direct Conflict among different changing sequences and Indirect Conflict among different changing sequences, and how to detect these conflicts is introduced in this section. First, The definitions of collaborative evolution conflicts are given, and then the propose algorithms are used to detect the conflicts. Finally, in order to enhance efficiency of algorithms, we propose methods to account the similarity of changing concepts and changing properties.

Internal Inconsistency occurs within the changing sequence, which means there are semantic conflicts in changing sequence. This conflict detection is relatively simple. We make use of the previous works [13, 15] for the Internal Inconsistency checking, and if there are some inconsistent operations, they will be deleted. The output of the checking algorithm is a changing sequence which meets the consistency checking.

The purpose of the following algorithm is to check **Internal Inconsistency**

Algorithm 1: Check Internal Inconsistency

```
Input: The initial ontology $O$ and a changing sequence $Chs=\{Och_1, Och_2, \ldots Och_n\}$
Output: A consistent changing sequence $Chs'(Chs' \subseteq Chs)$
1. Begin
2. $Chs' \leftarrow \emptyset$;
3. For (i=1; i<=n; i++)
   a) $ChsTemp \leftarrow Chs' \cup \{Och_i\}$;
   b) If IsConsistent($O_{ChsTemp}$) is TRUE
      i. Then $Chs' \leftarrow ChsTemp$;
```

```
4.  Return Chs';
5.  End
```

We provide the correctness proof of the algorithm.

Theorem 1. [*Soundness*] The changing sequence of the algorithm's output is consistent.

Proof:

1) If *Chs'* is Null, it acts on the original ontology, the result ontology is consist (Assume that the original ontology is consist).
2) Assume that i=k (1<=k<=n), the result changing sequence is consist;
3) Then when i=k+1, there are two cases which are: ⑴ The condition statement in the 3.b) line is true, in other words, the result ontology is $\mathcal{O}_{ChsTemp}$ consist, and *Chs'*= *ChsTemp*, so the result changing sequence is consist. ⑵ The condition statement in the 3.b) line is false, it is borne out in step 2.

Theorem 2. [*Completeness*] The changing sequence of the algorithm's output contains all changing operations which are not conflict with each other.

Proof:Assume that $\exists Och_k \in Chs$ and $Och_k \notin Chs'$, make $\mathcal{O}_{Chs' \cup Ochk}$ is consist. Then when i=k, $ChsTemp \leftarrow Chs' \cup \{Och_k\}$, **IsConsistent**($\mathcal{O}_{ChsTemp}$) is true, *Chs'=ChsTemp*, and then $Och_k \in Chs'$, which contradicts the assumption.

Definition 7. Two changing sequences Chs_i and Chs_j are said to have **Direct Conflict**, if and only if Chs_i and Chs_j satisfy $(Chs_i.AS \cap Chs_j.DS \neq \emptyset) \cup (Chs_i.DS \cap Chs_j.MS \neq \emptyset)$, we denote this by Conf(Chs_1,Chs_2).

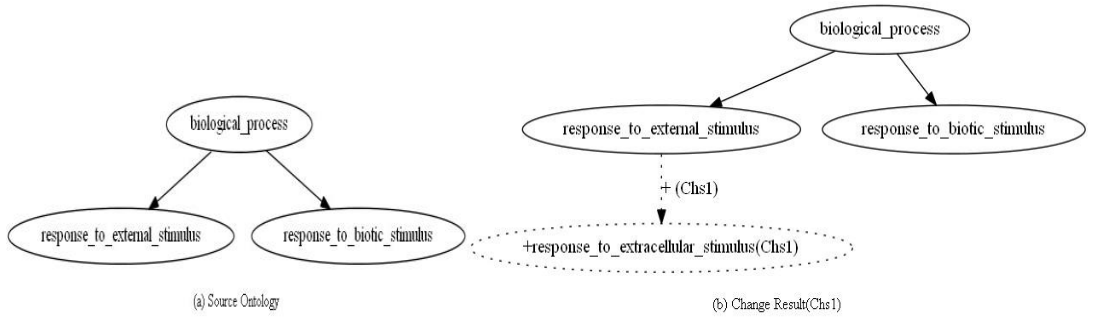
biological_process
response_to_external_stimulus
response_to_biotic_stimulus
(a) Source Ontology
biological_process
response_to_external_stimulus
response_to_biotic_stimulus
+ (Chs1)
+response_to_extracellular_stimulus(Chs1)
(b) Change Result(Chs1)

Figure 3. Direct Conflict

Figure 3 shows an example of a direct conflict. The source ontology is like Figure 3(a). There are two changing sequences, which are Chs_1={*AddConcept*(*response_to_extracellular_stimulus*, *response_to_external_stimulus*)}, Chs_2={ *DeleteConcept*(*response_to_external_stimulus*, $Concept_1$)}. According to the *Add*-extension rules, Chs_1 is extended as Chs_1={*AddConcept*(*response_to_extracellular_stimulus*, *response_to_external_stimulus*), ¬*Delete*(*response_to_external_stimulus*)}. Then $Chs_1.AS$={…, ¬*Delete*(*response_to_external_stimulus*)}, $Chs_2.DS$={ *Delete*(*response_to_external_stimulus*)}. It is apparent that $Chs_1.AS \cap Chs_2.DS \neq \emptyset$, so we can come to the conclusion that Conf(Chs_1,Chs_2).

The following is the algorithm for direct conflict checking.

Algorithm 2: Check Direct Conflict

```
   Input: The extended changing sequences set {$Chs_1$, $Chs_2$, …$Chs_n$}
   Output: The set of conflict sequence pairs Conf
1. Begin
2. Conf←∅;
3. DS[n], MS[n]←∅;
4. For (i=1; i<n; i++)
   a) For (j=1; j<=$Chs_i$.len; j++)
      i.   If $Chs_i^j$.operation $\in$ { $AddConcept \cup AddProperty$ } Then
           1.AS[i]←AS[i]∪$Chs_i^j$;
      ii.  Else If $Chs_i^j$.operation $\in$ { $DeleteConcept \cup DeleteProperty$ } Then
           1.DS[i]←DS[i]∪$Chs_i^j$;
      iii. Else MS[i]←MS[i]∪$Chs_i^j$;
5. For (i=1; i<=n; i++)
   a) For (j=1; j<=n && j≠i; j++)
      i.   For (k=1; k<= AS[i].len; k++)
           1.For (l=1; l<= DS[j].len; k++)
              a) If AS[i].k.suject = DS[j].l.subject
              b) Conf←Conf∪{($Chs_i$,$Chs_j$)};
6. For (i=1; i<=n; i++)
   a) For (j=1; j<=n && j≠i; j++)
      i.   For (k=1; k<= DS[i].len; k++)
           1.For (l=1; l<= MS[j].len; k++)
              a) If DS[i].k.suject = MS[j].l.subject
              b) Conf←Conf∪{($Chs_i$,$Chs_j$)};
7. Return Conf;
8. End
```

Definition 8. The changing sequences Chs_i and Chs_j are said to have **Inconsistent Conflict** if, and only if, Conf(Chs_1,Chs_2) = False, and $\mathcal{O}_{Chs1 \cup Chs2}$ not satisfies consistency checking.

As is shown in Figure 4, on the basis of Figure 4(a), which contains concepts $\mathcal{O}.C$={*medication*, *plant*, *animal*}, where {*plant*, *animal*} is the sub-concept set of *medication*, the first changing sequence is Chs_1={*AddConcept*(*aweto, plant*)}, and the second one is Chs_2={*AddConcept*(*aweto*, *animal*), *AddProperty* (*disjoint*,{*plant*, *animal*})}. In this example, it is obvious that the two changing sequences $Chs_1$and Chs_2 are not conflict in operation, and can be used for evolution of the source ontology. However, the result as is shown in Figure 4(c) is inconsistent with logic. Because there is a *disjoint* property from *plant* to $Concept_3$. The antagonistic fact in the result ontology is that *aweto*is both the subconcept of $Concept_2$ and *animal*. In other words, *plant* and *animal* have the same children which is not allowed with logic. So we say $Chs_1$and Chs_2 have **Inconsistent Conflict**.

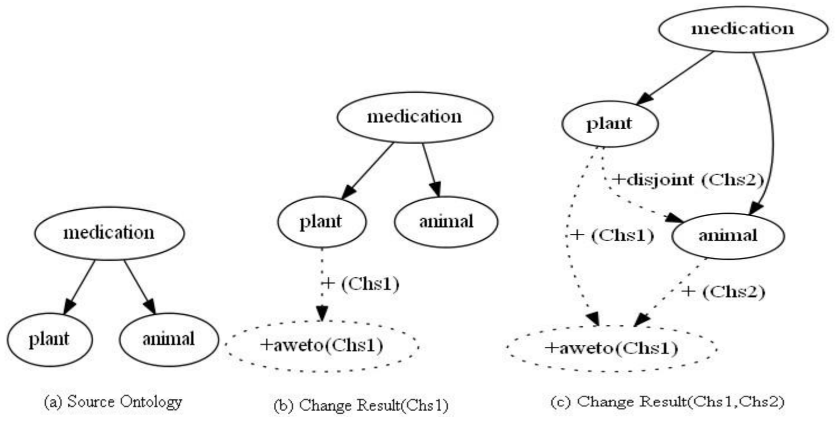

Figure 4. Inconsistent Conflict

Algorithm 3 is used to check **Inconsistent Conflict** between changing sequences.

Algorithm 3: Check Indirect Conflict

Input：The original ontology $\mathcal{O}$, the extended changing sequences set with out direct conflict among them $\{Chs_1, Chs_2, \ldots Chs_n\}$

Output：The set of inconsistent conflict sequence pairs **ConsisConf**

```
1. Begin
2. ConsisConf←∅;
3. For (i=1; i<n; i++)
   a) For (j=i+1; j<=n; j++)
      i.  For (k=1; k<=$Chs_j$.len; k++)
            1.$ChsTemp \leftarrow Chs_1 \cup \{ Chs_j^k \}$;
            2.If IsConsistent($\mathcal{O}_{ChsTemp}$) is FALSE
               a) ConsisConf←ConsisConf∪{($Chs_i$, $Chs_j$)};
               b) break;
4. Return ConsisConf;
5. End
```

The algorithm 3 is used to check inconsistent changing sequence pair. For the result, it was not wise to delete them directly, which is not meet users' changing demand. To satisfy the users with high limit and meet the requirement of consistent, we propose a concept which is named as **maximal consistent changing sequence**. The purpose is to find out the maximal changing sequence, which meet the consistent requirement and contains the largest number of operations, from the inconsistent changing sequence pairs.

The definition 8 gives the definition of **Inconsistent Conflict**. And the Figure 4 shows the inconsistent conflict example. In the example, the first changing sequence is Chs_1={*AddConcept*($Concept_4$,$Concept_2$)}, the second one is Chs_2={*AddConcept*($Concept_4$, $Concept_3$), *AddProperty* (*disjoint*,{$Concept_2$, $Concept_3$})}. It is apparent that the two changing sequences are not conflict in operation level, so can used to change the original ontology simultaneously. However, the result, as is shown in Figure 4. (c), is logical inconsistent obviously. Because there is a *disjoint* property from the concepts $Concept_2$ to $Concept_3$, and the concept $Concept_4$ is both a subconcept of $Concept_2$, and a subconcept of $Concept_3$. We use Och_1, Och_2, Och_3 to represent the changing operations separately. The target of the algorithm 4 is to find out a **maximal consistent changing sequence**. In the example, {Och_1,Och_2}, {Och_2,Och_3} and {Och_1,Och_3} are the maximal consistent changing sequences.

Before the algorithm, we definite a data structure (Figure 5), which is used to store the list structure of changing sequences.

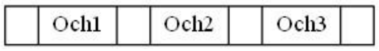

Figure 5. Data Structure

```
typedef  struct{
      ElemType  *elem
      int       length   //length
      int       listsize
} Seq_List;
```

The algorithm is to find out the maximal consistent changing sequence.

Algorithm 4: Getting MaxComseq

```
Input: the ontology $\mathcal{O}$, inconsistent changing sequence pair ($Chs_1$, $Chs_2$,)
Output: The set of maximal consistent changing sequence MaxComseq
1. Begin
2. MaxComseq←∅;
3. List←$Chs_1$;
4. Seq_List  *p = List;
5. For first $Och_i$ in $Chs_2$ do
   a) For each interval in *p        //for the purpose of not changing the original
        sequence, the insert position is behind $Och_{i+j}$ of *p
      i.    Insert(Och, *p);
      ii.   $Chs_2$←$Chs_2$–{Och};
      iii.  If ∃ Dependency(Och,Och')
      iv.   $Chs_2$←$Chs_2$–{Och'};
      v.    If IsConsistent($\mathcal{O}_{*p}$) is TRUE
            1.List' = Insert(Och, *p);
            2.InsertChild(List', *p);
   b) If *p has Sibling
      i.    p = *p.rightSibling;
   c) Else  p = *p.leftChild;
6. MaxComseq ← LeafList;
7. Return MaxComseq;
```

In the example of Figure 4, if Chs_1 is selected as the first changing sequence, according to the algorithm 4, we can get the computational process as is shown in Figure 6. Firstly, there is only one changing operation Och_1 in the first changing sequence, so there is one element in the first list. Next, the changing operations of the second changing sequence are inserted into the list. The operation Chs_2 take the first, and we can get the two lists in the second level. And then is the operation Och_3. In the understratum, there are three sequences, which are all not met the consistent requirement. So when the operation Chs_1 is selected as the first operation, the maximal consistent changing sequences are {Och_2, Och_1} and {Och_1, Och_2}, which are the same one.

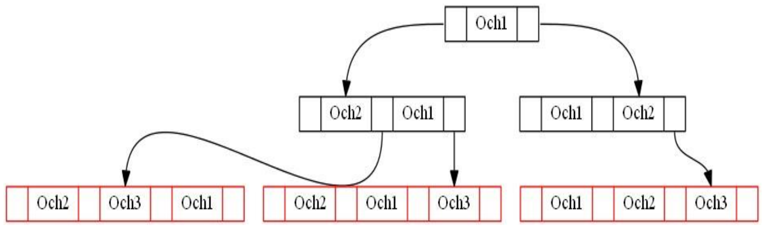

Figure 6. Computational Process

Similarly, if the operation Chs_2 is selected as the first changing sequence, we can get two maximal consistent changing sequences which are {Och_2, Och_3} and {Och_3, Och_1}. So for the inconsistent changing sequence pair (Chs_1, Chs_2,), according to algorithm 4, we can get three maximal consistent changing sequences {Och_1, Och_2}, {Och_2, Och_3} and {Och_3, Och_1}.

4. Similarity Calculation

In practice, in the different changing sequences there are always some same or similar operations. To improve the efficiency of the algorithm, we propose the formulas to calculate the similarity of the changing operations.

The changing operations we defined involve classes and properties. So we need to give the changing similarity both to the changing classes and changing properties.

The similarity formula of changing class is as follow.

$$\text{SemSim}_{\text{class}}(Och_1,Och_2) = \text{Equal}(Och_1.\text{operation}, Och_2.\text{operation}) *\text{SemSim}(Och_1.\text{subject}, Och_2.\text{ subject})* \text{SemSim}(Och_1.\text{constraint}, Och_2.\text{constraint}) \quad (1)$$

where

$$\text{Equal}(Och_1.operation,\ Och_2.operation) = \begin{cases} 1(Och_1.\text{operation} = Och_2.\text{operation}) \\ 0(Och_1.\text{operation} \neq Och_2.\text{operation}) \end{cases} \quad (2)$$

SemSim(Och_1.*subject*, Och_2. *subject*) and SemSim(Och_1.*constraint*, Och_2.*constraint*) are both used to calculate the similarity of the changing classes. It has close relationship with the semantic (mainly consider the class *name*), the location in the ontology hierarchy (*Depth*), sup-class (*supClass*) and sub-class (*subClass*).

$$\text{SemSim}(C_1,C_2)= \text{nameSim}(C_1,C_2)*\text{DepthSim}(C_1,C_2)* \text{supClassSim}(C_1,C_2)* \text{subClassSim}(C_1,C_2) \quad (3)$$

where

$$DepthSim\left(C_1,C_2\right)=\begin{cases} 1\ ,(C_1.dirsupClass = C_2.dirsupClass, \\ \qquad C_1 and C_2 have\ the\ same\ direct\ sup-class) \\ \dfrac{2*Depth\left(supClass\left(C_1,C_2\right)\right))}{Depth\left(C_1\right)+\ Depth\left(C_2\right)} \end{cases} \quad (4)$$

where Depth(C) is used to indicate the location of class C in the ontology hierarchy. supClass(C_1,C_2) gets the nearest sup-class of C_1 and C_2.

$$supClassSim\left(C_1,C_2\right) = \begin{cases} 1,(C_1.\ supClass = C_2.\ supClass) \\ 0,\text{else} \end{cases} \quad (5)$$

where C.supClass gets all ancesrtors sof class C. The function subClassSim (C_1,C_2) is similar to supClassSim(C_1,C_2).

$$nameSim\left(C_1, C_2\right) = \begin{cases} 1, (same,\ synonym\ or\ abbreviation) \\ StrSim\left(C_1, C_2\right), else \end{cases} \tag{6}$$

StrSim(C_1,C_2) is the similarity function of Strings C_1 and C_2, it compare the name's strings of C_1 and C_2.

Unlike the changing similarity of classes, the changing similarity of properties dosen't involves the location of the ontology hierarchy. It is as follow

$$\begin{aligned} SemSim_{property}\left(Och_1, Och_2\right) &= Equal\left(Och_1.operation,\ Och_2.operation\right) * \\ SemSim&\left(Och_1.subject.domain,\ Och_2.subject.domain\right) * \\ SemSim&\left(Och_1.subject.domain,\ Och_2.subject.domain\right) \end{aligned} \tag{7}$$

The formula 7 involves the *name*, *domain* and *range* of the properties. Where the similarity of *name* is similar to which of the changing classes (formula 6). The *domain* and *range* of properties are both classes. So the calculation of similarity can borrow ideas from formula 3.

According to formulas 1-7, we can merge and simplify the changing sequence pair which have inconsistent conflict, then reducing computation load and improving the efficiency of algorithm.

5. Time Complexity Analyzing

The purpose of algorithm 4 is to find out the maximal consistent changing sequence from two sequences are indirect conflict sequences. The time complexity is directly related to the siza of the two sequences. Before the time complexity analyzing, we assume that there are two sequences are indirect conflict changing sequences Chs_1 and Chs_2, and contains n_1 and n_2 operations respectively. The process is initializing *List* with a changing sequence (line 3), and then inserting the operations of another changing sequence into the *List* in turn. If Chs_1 is used to initialize *List*, then there are n_1+1 positions to insert the first operation of Chs_2. And at worst, if there are no dependency relationship among the operations, there are n_1+2 positions to insert the second operation, and so on, the last operation of Chs_2 will have n_1+n_2 positions to select. So, we can learn that, the algorithm runs in $O(n_1*n_2)$time.

The similarity formulas are used before the algorithm. The purpose is to improve the efficiency. For if there are two operations from different changing sequences and the different changing sequences are indirect conflict, then the search procedures will be reduced n_1 or n_2 time.

6. Experimental Design

We design our experiment using Java 1.6. The server is conducted on Intel(R) Core(TM) i3 CPU M 350 @ 2.27GHZ with RAM 2.00GB. The evolved ontology is shown in Figure 7. It is a fraction of biological process ontology, which is one part of gene ontology (http://www.geneontology.org/GO.ontology.structure.shtml).

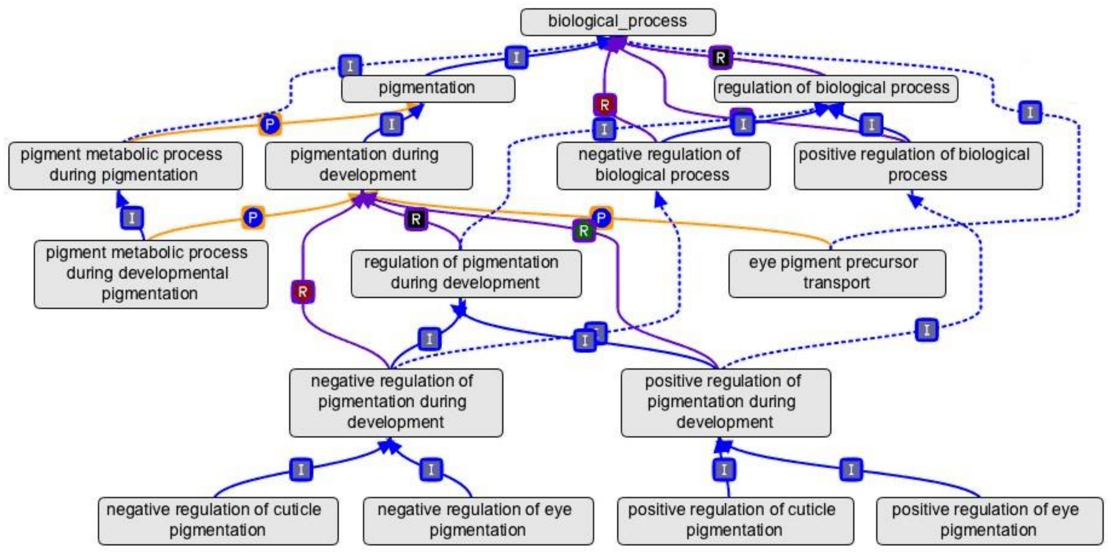
biological_process
pigmentation
regulation of biological process
pigment metabolic process during pigmentation
pigmentation during development
negative regulation of biological process
positive regulation of biological process
pigment metabolic process during developmental pigmentation
regulation of pigmentation during development
eye pigment precursor transport
negative regulation of pigmentation during development
positive regulation of pigmentation during development
negative regulation of cuticle pigmentation
negative regulation of eye pigmentation
positive regulation of cuticle pigmentation
positive regulation of eye pigmentation
I
P
R

Figure 7. A Fraction of Biolobical Process Ontology

The experimental result is summarized in Table 1. In order to evaluate, we studied out some changing sequences. The first row and the second one list the number of the changing operations of the changing sequences. And the third row provides the number of same or similar changing operations between the previous two. As seen from the table, if similarity computing is used to merge same or similar operations before algorithm 4, the efficiency can be greatly improved.

Table 1. The Experiment Data for Algorithm 3

	Chs1	Chs2	same operations	time(sm)	the result
1	3	2	1	0	1
2	3	2	0	0	8
3	4	3	1	0	1
4	4	3	0	0	21
5	5	4	2	0	1
6	5	4	0	0	48
7	6	5	2	0	1
8	6	5	0	0	124
9	7	6	3	0	1
10	7	6	0	63	425
11	8	7	3	0	1
12	8	7	0	297	1078
13	9	8	3	0	1
14	9	8	0	1716	2696
15	10	9	3	0	1
16	10	9	0	13291	8118
17	11	10	3	0	1
18	11	10	0	67954	21757
19	12	11	3	0	1
20	12	11	0	382104	54657

7. Summary and Future Works

With the rapid growth in popularity and size, the complexity of ontology increases tremendously. In the field of collaborative evolution of biomedical ontology on large-scale ontology, there exists inevitable conflicts, which may cause the inconsistent ontology.

In this paper, we present a new method to detect conflicts for collaborative ontology of Medicine Ontology. We classify conflicts as three groups: internal inconsistencies conflicts in change sequence, direct conflicts and inconsistent conflict between change the sequences. And for different conflicts, high effective detecting algorithms are presented with evaluation. In particular, if there are inconsistent conflicts between changing subsequences we present an algorithm to get the maximum consistent changing subsequence to ensure the process of ontology evolution. Finally, we design our experiment to show that the approach can find out the same operations of different changing sequences and improve efficiency of the algorithm.

In the future, we will continue work on the conflicts of collaborative ontology evolution. In this research, we focus on the conflicts checking, and more work should be done to find out the root of conflicts. So, our future research may focus on the conflicts diagnosis method of collaborative ontology evolution.

Acknowledgements

This work is supported by the The PhD Start-up Fund of ShanDong Institute of Business and Technology(DS201405), the Open Project Program of Artificial Intelligence Key Laboratory of Sichuan Provence(Sichuan University of Science and Engineering), China (2013RYJ05), the National Natural Science Foundation of China (Grant No.61272244,61373079,61175023,61175053,61272430,61272369) and Shandong Provincial Natural Science Foundation (ZR2013FL022)

References

[1] Y. Yongyan, "The study on marketing strategies using auto-reasoning model based on ontology and rules," International Journal of Digital Content Technology and its Applications, vol. 6, no. 14, **(2012)**, pp. 182-188.

[2] F. Belleau, M. A. Nolin, N. Tourigny, P. Rigault, and J. Morissette, "Bio2RDF: towards a mashup to build bioinformatics knowledge systems," Journal of biomedical informatics, vol. 41, no. 5, **(2008)**, pp. 706-716.

[3] M.-A. Nolin, P. Ansell, F. Belleau, K. Idehen, P. Rigault, N. Tourigny, P. Roe, J. M. Hogan and M. Dumontier, "Bio2RDF network of linked data," in Semantic Web Challenge; International Semantic Web Conference (ISWC 2008), **(2008)**.

[4] D. Fensel, F. van Harmelen, B. Andersson, P. Brennan, H. Cunningham, E. Della Valle, F. Fischer, Z. Huang, A. Kiryakov and T.-I. Lee , "Towards LarKC: a platform for web-scale reasoning," in Semantic Computing, 2008 IEEE International Conference on, **(2008)**, pp. 524-529.

[5] L. Stojanovic, A. Maedche, N. Stojanovic, and R. Studer, "Ontology evolution as reconfiguration-design problem solving," presented at the Proceedings of the 2nd international conference on Knowledge capture, Sanibel Island, FL, USA, **(2003)**.

[6] N. Noy, S. Kunnatur, M. Klein, and M. Musen, "Tracking changes during ontology evolution," The Semantic Web–ISWC 2004, **(2004)**, pp. 259-273.

[7] C. Tempich, H. Pinto, Y. Sure, and S. Staab, "argumentation ontology for distributed, loosely-controlled and evolving engineering processes of ontologies (DILIGENT)," The Semantic Web: Research and Applications, **(2005)**, pp. 21-43.

[8] L. Stojanovic, "Methods and tools for ontology evolution," **(2004)**.

[9] S. H. Hwang, H. G. Kim, and H. S. Yang, "A FCA-based ontology construction for the design of class hierarchy," Computational Science and Its Applications–ICCSA 2005, **(2005)**, pp. 307-320.

[10] A. G. Perez, Ó . Corcho, M. Fernandez-Lopez, and J. Angele, "Survey on Ontology Tools", no.1.3, **(2002)**.

[11] M. Klein and N. F. Noy, "A component-based framework for ontology evolution," in Proceedings of the IJCAI, **(2003)**.

[12] Y. Sure, J. Angele, and S. Staab, "OntoEdit: Multifaceted Inferencing for Ontology Engineering," Journal on Data Semantics, vol. 1, no. 1, **(2003)**, pp. 128-152.
[13] Y. Song and R. Chen, "Non-standard Reasoning Services for the Verification of DAML+OIL Ontologies," in Artificial Intelligence Applications and Innovations. vol. 339, **(2010)**, pp. 203-210.
[14] W. L. Hürsch, "Maintaining Consistency and Behavior of Object-Oriented Systems during Evolution," in Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'97), ACM SIGPLAN Notices, **(1995)**.
[15] Y. Song, R. Chen, and Y. Liu, "A Non-Standard Approach for the OWL Ontologies Checking and Reasoning," Journal of Computers, vol. 7, no. 10, **(2012)**.

Authors

Song Yingjie
2013 Doctor's degree in Dalian Maritime University
2009 Master's degree in Dalian Maritime University
Research interests Semantic Web, Ontology Evolution, and Behavior Identification

Zhang Bin
2010 Master's degree in Dalian Maritime University
2008 Bachelor's degree in Qilu University Of Technology
Research interests Software Quality Assurance、Embed-Software testing、Software reliability testing

Yanyan Mao
2001 Bachelor's degree in Shandong University of Technology
2004 Master's degree in Dalian University of Technology
Research interests wireless sensor networks, internet of things, mobile computing, and distributed simulation.