Fourth International Conference on Semantics, Knowledge and Grid

A Collaborative Ontology Construction Tool with Conflicts Detection

Yewang Chen, Shaolei Zhang, Xin Peng, Wenyun Zhao
School of Computer Science and Technology, Fudan University, Shanghai 200433，China
{061021061, 072021121,pengxin,wyzhao }@fudan.edu.cn

Abstract—*Ontology construction itself is a big challenge, especially when the ontology is expected to have relevance and value to a broad audience. In recent years, many collaborative methods for ontology construction are proposed. However, whenever there is collaborative developments there are conflicts. Current collaborative methods are suitable for teamwork, but they are ineffective if the team is big and lacks communication. In this paper, we introduce the tool we establish for collaborative ontology construction, as well as collaborative conflict detection method.*

I. Introduction

Ontology is used to unify the formal description of knowledge throughout the whole process of knowledge management(KM), such as [17]． In knowledge-based systems, it is important to get support for ontology creation and maintenance. However, ontology construction itself is a big challenge, especially when the ontology is expected to have relevance and value to a broad audience[9]. For example, in our project on knowledge and resource repository in the agriculture domain, an ontology-based knowledge and resource management system is being developed to provide both knowledge and knowledge resources (e.g. audio, video and documentations on agricultural techniques and markets) for users. In the system, ontology is employed as both the representation of direct agriculture knowledge and semantic annotations for resources.

Recently, there are many tools for ontology construction, some are centralized methods and the others are distributed and collaborative. Construct ontology manually by centralized teams is a complex, expensive and time-consuming process [8,24]. Therefore, collaborative methods are proposed which make the ontology development a joint effort reflecting experiences and viewpoints of persons who intentionally cooperate to produce it [9]. Collaborative construction methods meet the requirements of public ontologies having relevance and value to a broad audience better.

However, whenever there is collaborative developments there are conflicts. The case is even worse in ontology, for the basic units are concepts and rich semantic relationships among these concepts. Therefore, conflicts in collaborative ontology development much more likely exist. On the other hand, in large-scale collaboration-based ontology development, there are many participators working on the same ontology model. Most of them are not aware of the existences of their co-workers at all, let alone have sufficient communications. Lack of effective coordination will also exacerbate the problem.

Therefore, in this paper, we establish a tool for collaborative ontology development with collaborative conflict detection. In our system, we differentiate three categories of conflicts in the collaborative construction process, i.e. collaborative conflict、consistency conflict and anomalistic conflict. We deal with collaborative conflict different from handling inconsistency in tradition methods do. In this paper, we only address how to detect collaborative conflict in detail because of pages limited.

The remainder of this paper is organized as follows. Section II discusses related work. Section III gives an overview of our framework and identifies three categories of conflicts. Section IV addresses an approach to detect collaborative conflicts. Section V shows a case study and the result of experiments. The last section gives conclusion.

II. Related Works

In this section, we will try to provide an overview of some available tools for the building of ontologies. There are some good surveys on ontology construction tools[1,2]. [3]focus on collaborative constructions editors and make comparisons. [13] also provides a comprehensive ontology editors (including 56 editors) survey results. We will provide a brief description of some tools, and their conflicts handling mechanism as follows.

Protégé[7] is an established line of tools developed at Stanford University for knowledge acquisition. It provides a graphical and interactive ontology-design and knowledge-base–development environment. Ontology developers can access relevant information quickly whenever they need it, and can use direct manipulation to navigate and manage ontology. One of the major advantages of the Protégé architecture is that the system is constructed in an open, modular fashion. WebOnto[10] is a tool developed by the Knowledge Media Institute (KMi) of the Open University (England). One of its main features is that by means of broadcast/receive and making annotations, it supports the collaborative browsing, creation and editing of ontologies, which are represented in the knowledge modeling language OCML. KAON[5] focuses on that changes in ontology can cause inconsistencies, propose deriving evolution strategies in order to maintain consistencies. OntoWiki[11] and MediaWiki[12] foster social collaboration aspects by keeping track of changes, allowing to comment and discuss every single part of a knowledge base, enabling to rate and measure the popularity of content and honoring the activity of users. It enhances the browsing and retrieval by offering semantic enhanced search strategies. [6] introduces resolution strategies to ensure the consistent of as the ontology evolves, and present a model for the semantics of change for OWL ontologies, considering structural, logical, and user-defined consistency.

TABLE 1. A SIMPLE ONTOLOGY TOOLS COMPARISON

Feature	Protégé	KAON	WebOnto	OntoWiki	MediaWiki
Consistencychecking	Yes	Yes (for evolution of ontology).	Yes	No	Yes
Graphical	Yes	Yes	Yes	Yes	Yes
Collaborative working	Yes	Yes (Concurrent access control with transaction oriented locking and rollback.)	Yes	Yes	Yes
Collaborative conflicts handling	No	No	No	No	No

There are some drawbacks of these methods as follows. Some of them (e.g. [7]) provide discussion thread for users to communicate. KAON[5] provides concurrent access control with transaction oriented locking, and in some cases, even rollback. Almost all current tools provide functionality for consistency checking. They accept all works that participators have done until inconsistency detected. However, few of these methods distinguish collaborative conflicts from logical inconsistency. It is helpful to differentiate them from each other, because some collaborative conflicts may not result in ontology logical inconsistency, but violate user defined rules or other exception (we will give further explanation in *section III B*). Furthermore, concurrent access control with transaction oriented locking as database is not suitable for ontology, for the rich semantic relationships among ontology entities.

Table 1 makes a simple comparison among these tools. We can see that almost all tools check ontology consistency, but none of them handles collaborative conflicts. Therefore, we need new approach to overcome these deficiencies

III. OVERVIEW OF OUR SYSTEM

A. Ontology Command

In collaborative ontology development, each participator can browse the model and choose some segments to edit. When participators need to revise the content, we provide a set of commands for them. Each command is composed of a operation and some operands (ontology entity). For example, *AddClass*/*DelClass*, *AddInstance*/*DelInstance* etc. In our system, we identify a set of atomic commands as table 2(A) lists some of them and their operands. We name these commands as OntoCommand.

Definition 1 (OC---OntoCommand). OC=<*Name,E,V*> is a function such that E→E', where, *Name* is the command name, *E* is an ontology entity set , *V* is the value of parameter.

With these basic commands, complex ontology editing can be represented as a sequence of atomic editing operations. For example, there is an original ontology model showed in Fig.1 (a). Suppose that a participator want to add a new concept hierarchy, named *Carnivore*, between *Animal* and *Tiger*, he should do the following sequence of operation:

1)add new class command *AddClass('Carnivore')*,

2)perform *AddSubClass(Carnivore, Animal)* to let *Carnivore* be a subclass of *Animal,*

3)perform *DelSubClassRelation(Tiger,Animal),*

4)perform *AddSubClass(Tiger,Carnivore).* The participator also can add concept *OmnivorousAnimal* between *Rat* and *Animal*.

These series of operations can be represented a commands list as listed in table 2(B). The result of executing these commands is showed in Fig.1(b).

TABLE 2(A) : SOME ONTOCOMMANDS

Name	**Parameter (E)**
AddClass/DelClass	(aClass)
AddSubClass/DelSubClassRelation/	(aClass,supClass)
DelEquClass/DelSameClass	(aClass1,aClass2)
AddDataProperty	(aProperty)
AddObjProperty/DelObjProperty	(aObjProperty)
AddRest/DelRest	(aRest, aProperty)
AddDifferentClassRelation/	(aClass1,aClass2)
AddEquivalentClass/AddSameClass	(aClass1,aClass2)

TABLE 2 (B). COMMAND SEQUENCE PARTICIPATOR PERFORM

	Command Sequence
1.	*AddClass('Carnivore')*
2.	*AddSubClass(Carnivore, Animal)*
3.	*DelSubClassRelation(Tiger,Animal)*
4.	*AddSubClass(Tiger,Carnivore)*
5.	*AddClass('OmnivorousAnimal')*
6.	*AddSubClass(OmnivorousAnimal, Animal)*
7.	*DelObjProperty(prey,Tiger),*
8.	*AddObjProperty(prey,Carnivore, Animal)*
9.	*DelSubClassRelation(Rat,Animal)*
10.	*AddSubClass(Rat, OmnivorousAnimal)*

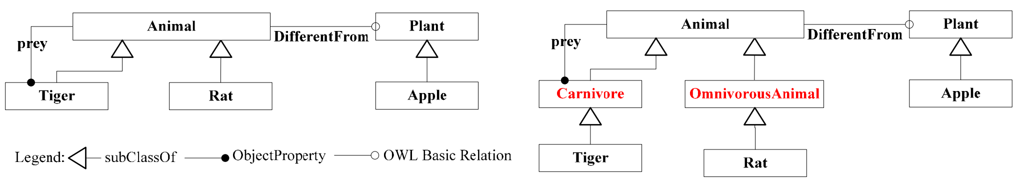


Fig. 1.(a) Initial Ontology Segment. (b) New Ontology Segment

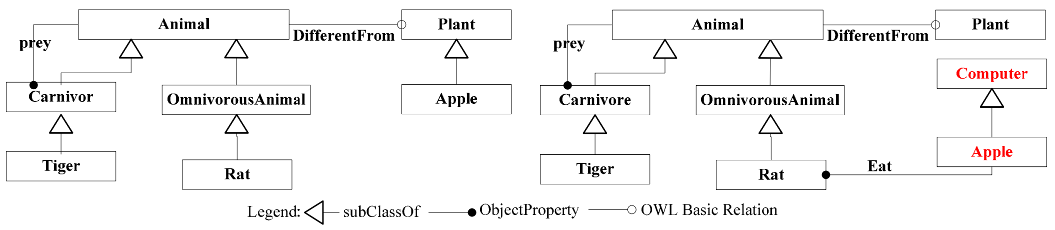


Fig. 2.(a) Original Ontology Segment. (b)Result of Accepting All Revisions

B. *Three Categories of Conflicts*

As mentioned in *section II*, few of current ontology construction tools distinguish collaborative conflict from consistency conflict. Some collaborative conflicts may not always result in ontology logical inconsistency, but violate user-defined rules or get result unexpected. For example, Fig.2 (a) shows an original ontology segment, which depicts the relation between *Animal* and *Plant* as well as their subclasses. If one participator adds *Eat* object property between *Rat* and *Apple,* meanwhile the other participator move *Apple* to be a subclass of *Computer,* then Fig.2 (b) shows the result of executing all operations. There is no inconsistency at all in Fig.2 (b), but it is not the expected result of the first participator. What he (or she) expects is '*Rat eats a kind plant',* instead of '*Rat eats a kind Computer*'. Therefore, in our opinion, collaborative conflict should be handled in a way that is different from inconsistency handling. In our system, we identify three categories of conflicts, i.e. collaborative conflict 、consistency conflict and anomalistic conflict. For each kind of conflict, we adopt different mechanisms to detect and resolve.

Definition 2(Collaborative Conflict): We call two revisions performed by different participators on the same ontology entity conflict with each other, if one changes the semantic of the entity, while the other still uses it with the assumption that the original semantic does not change.

Definition 3 (**Consistency Conflict**) We call a revision on ontology O, conflicts with a set of consistency conditions K iff $\exists k \in K$, such that O' does not satisfies the consistency condition k(O'), where O' is the result from O by the revision.

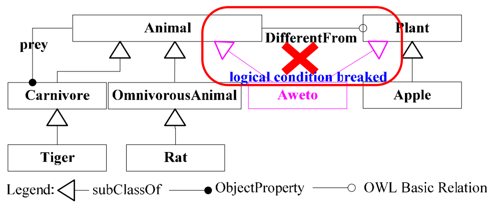


Fig.2.(c) An Example of Consistency Conflict.

For example, suppose one participator adds a concept *Aweto*, and then makes it as a subclass of both *Animal* and *Plant*. It is obvious that these changes breaks the logical condition '*Animal owl:differentFrom Plant*' as fig.2(c) shows.

Besides collaborative conflict and consistency conflict, there is still one kind of mistakes that is difficult to detect automatically. These mistakes are "outside" of the ontology language itself, and may make none logically consistent. For example, type error is a typical conflict of this kind. At this point domain experts must involve in the checking process manually

Definition 4 (**Anomalistic Conflict**): We call all mistakes different from collaborative conflict and consistency conflict as **Anomalistic Conflict**.

C. *Our System Framework*

Fig.3 shows the overview of our system. We use OWL [14] as our ontology language, and adopt B/S architecture, i.e. each user works in his (or her) own private work (Client), and all works done will be submitted to public workspace (Server). The following paragraphs will drift into the detail of the process.

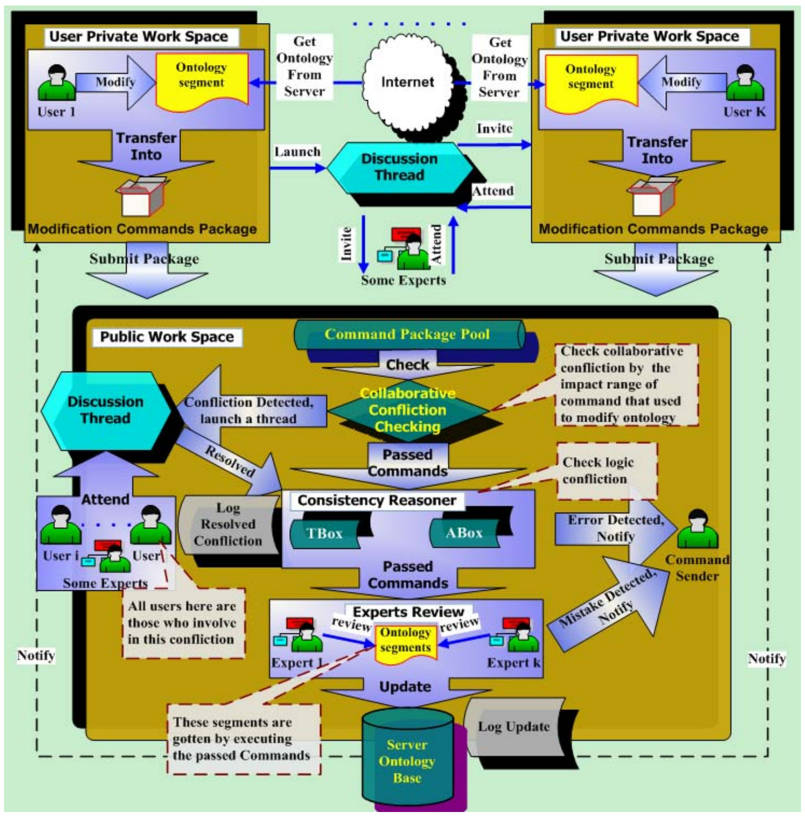


Fig. 3. Overview of Collaborative Ontology Building Process

When a user U1 logins, the user get an ontology segment he (or she) wants from server. Then user can modify it through the UI interface we provide, e.g user can add remove classes. All changes happen here will be transferred into a set of commands that can be executed in the server. We wrap these commands into a Command Package **(**CP) and submit it to server. Notice that other users can also download a segment that may be the same or intersects with the segment of U1, and modify it independently. By this way, it is easy for modifications on ontology from different users conflict with each other.

For there are collaborative conflicts and other conflicts may exists in the operation list submitted by users, we must detect and resolve them before updating the ontology in server. Therefore, in the public workspace, there are 5 main steps mainly focused on detecting and resolving conflicts, as following.

1) For all commands submitted by participators, a command package pool is used to stores them. It has a interface for users to interact with our system.

2) For every period, all commands received will be sent for the first kind of checking, i.e. collaborative conflicts checking, in order to detect collaborative conflicts. If such kind of conflicts is detected, system will launch a discussion thread, and invite all participators who involve in this conflict with some experts to attend the discussion until the conflicts are resolved.

3) All commands passed step 2 will be submitted to next module for consistency checking, which is used to deal with consistency conflicts, for there are some mistakes users made break ontology consistency. In this step, we use Pellet[4] to check logical consistency. If such kind error detected, the commands will be rejected and notify the command sender.

4) After 3 preceding processes, there are still some mistakes remain, e.g type mistake or ontology concept classification mistake, these type mistakes need experts check it manually. If any such kind of mistake found, expert may revise it directly or refused and returned it to the command sender.

5) Lastly, all correct commands will be executed to update ontology base and notify all users.

D. A Simple Case Of Conflicts Detecting

This section addresses a simple case of detecting conflicts among commands submitted by two participators (U1 and U2). They work independently, the ontology segment they revise is showed in figure 1(b), and operations they perform list in table 3 (a). After they submit their works, a series of checking will be launched in the public server workspace, and table 3(b) shows the checking result.

TABLE 3 (A). A SIMPLE CASE: COMMANDS USERS SUBMITTED

Commands Sequence			
	Commands U1 Performs		Commands U2 Performs
1	DelClass ('Rat')	1	AddClass('Cat')
2	AddClass ('Mouse')	2	AddSubClass('Carnivore',' Cat')
3	AddSubClass('Plant', 'Aweto')	3	AddObjProperty ('Cat', 'FavoritesFood', 'Rat')
4	AddSubClass('Animal', 'Aweto')	4	AddSubClass('Tiger', 'SouthChianTiger')

TABLE 3 (B). A SIMPLE CASE: CONFLICTS DETECTED

Conflicts Checking and Result	
Checking	Conflicts Detected
Collaborative Conflicts Checking	1st command of U1 conflicts with 3rd command of U2
Consistency Conflicts Checking	Aweto cannot be subclass of Plant and subclass of Animal, for Plant is different from animal.
Anomalistic Conflicts Checking	Type mistake 'SouthChianTiger', correct type is 'SouthChinaTiger'

Commands passed the three kinds of checking are: *AddClass ('Mouse')* 、 *AddClass('Cat')* 、 *AddSubClass(Carnivore, Cat)*. Then the commands executor in our system will execute these commands, while other commands that may result in conflicts should be return to user or domain experts for further confirmation.

IV. COLLABORATIVE CONFLICT DETECTION

As stated in *section III B*, collaborative conflict should be differentiated from inconsistency. This section addresses how to detect collaborative conflicts in Command Package Pool (**CPP**) automatically.

Look back on the first example in *section III.B*, suppose that the second participator only inserts new concept *Fruit* between *Apple* and *Plant,* instead of moving *Apple* to be a subclass of *Computer*, then the result is acceptable. Fig.4 shows the comparison, where (a) shows the acceptable result, and (b) shows unacceptable one. In this case, it is not easy to judge whether the changes on concept *Apple* is acceptable or unacceptable, for lacking absolute standard. Therefore, heuristic methods must be adopted.

In the following, we firstly introduce command Impact Range (IR). Base on it, we propose **OntoSIM,** which is an extensible set of heuristic matchers. Each matcher adopts different point of view to calculate the similarity value between two entities.

A. Command Impact Range

In the process of collaborative ontology development, subtle semantic relationship exists among entities in ontology, change one entity may produce chain reaction. This means that for each operation, there is a set of entities may be impacted. Therefore, given a command oc we define a function:

IR(Impact Range) calculates the impact range of a oc, denote as IR(oc). For each atomic OntoCommand we define a unique IR value, table 4 lists some of them.

For example, if a user uses *DelClass* command oc to delete class 'Carnivore'. According to table 4, **IR(oc)**={*Carnivore, prey,Tiger* }.

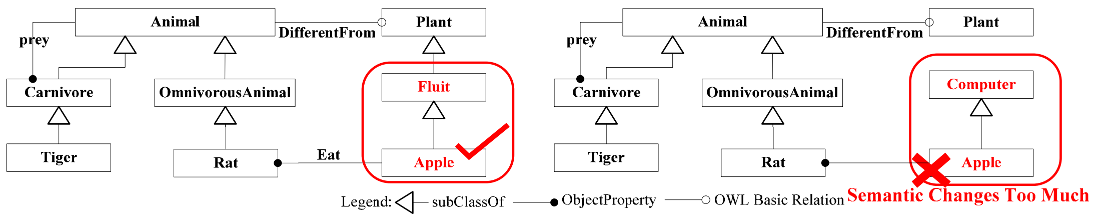


Fig. 4. (a) Acceptable Result. (b)Unacceptable Result.

TABLE 4: SOME IR VALUE

Command	E	IR(oc)
DelClass	(aClass)	Return **E** ⋃ {o| o.Domain=aClass, o is ObjectProperty} ⋃ { o| o.Range=aClass, o is ObjectProperty } ⋃ {ins| ins.instanceOf(aClass), ins is an individual} ⋃ {class| (aClass. directSubclassOf (aClass1)}
DelInstance	(aIndividual)	Return **E** ⋃ { ins| ins is the same as aIndividual }
DelProperty	(aProperty)	Return **E** ⋃ {r| r.actOn(aProperty)= true, r is Restriction }
AddEquClass	(aClass1,aClass2)	Return **E** ⋃ {class| (aClass. directSubclassOf (aClass1) } ⋃ {class| (aClass. directSubclassOf (aClass2)
AddObjProperty	(aClass1,aClass2)	Return **E** ⋃ {class| (aClass.directSubclassOf(aClass2)}…

TABLE 5 . SOME CHARACTERISTIC AND HEURISTIC SIMILARITY MEASURES

Measure type	Measure	Heuristic matcher Description
Semantical	Entity Name	Compare the string or label of e1 and e2
	Class Property	Compare matched property of e1 and e2
	Property Domain & Range	Compare matched domain and range of e1 and e2
	Class Restriction	Compare matched restriction of e1 and e2
	Restriction OnProperty	Compare matched property of e1 and e2
	Individual Type	Compare the types of e1 and e2
Structural	Direct Sup/Subproperty	Compare matched direct sup/subproperty of e1 and e2
	Direct Sup/Subclsass	Compare matched direct sup/subclass of e1 and e2
	Depth Distance	Compare hierarchy distance of e1 and e2

B. Heuristic Similarity and OntoSIM

Definition 5 (Heuristic Similarity): Given an entity e and two commands oc and oc' , where, $e \in IR(oc) \cap IR(oc')$. **HSim**(e1,e2) is a heuristic matcher returns the similarity between e1 and e2, where e1 is the entity produced from e by oc, e2 is the entity produced from e by oc' independently.

We differentiate HSim into two kinds, the 1st is semantical which calculates similarity based on semantic information, denoted as **HSemSIM**; the 2nd is structural which is based on the structure information, denoted as **HSturSIM**. Table 5 lists the some heuristic matchers used in our system. We will address three of them in detail as follows.

(1)Name: It is a semantical matcher, which calculates the SIM between e1 and e2 by label string.

$$HSemSim_{string}(e1, e2) = \begin{cases} 1, \text{synonymous or abbreviation} \\ StrSim(nameof(e1), nameof(e2)), \text{otherwise} \end{cases}$$

If e1 is synonymous of e2 or abbreviation relationship between two labels then return 1, otherwise use StrSim[16] to compare two string and return a similarity value.

(2)Depth Distance: It is structural matcher deals with ontology class and objects property, otherwise returns -1. It calculates the SIM value base on the fact that the deeper the two entities locate in one hierarchy, the higher the similarity is. Therefore,

$$HStruSim_{DD}(e1, e2) = \frac{2 \times depth(NA(e1, e2))}{depth(e1) + depth(e2)}$$

where, NA(e1,e2) gets the nearest ancestor of e1 and e2.

(3)Matched Subclasses: It is a structural matcher deals with classes, otherwise returns -1. It calculates the SIM value base on the fact that the more subclasses matched, the higher the similarity is. Therefore,

$$HStruSim_{subclass}(e1,e2) = \begin{cases} 1, n = N1 \\ \sqrt{n/N1}, N2 \geq N1 \\ n / N1, \text{otherwise} \end{cases}$$

where n is the number of matched subclasses, N1 is the number of direct subclasses of e in original ontology, N2 is mean number of direct subclasses of all classes in original ontology segment. The reason we use square root of (n/N1) is to decrease the SIM value error when N1 is too small.

Similarity Aggregation: For all matchers used in our system, we adopt formula **OntoSIM** to combine all of the together.

$$OntoSIM(e1,e2) = \begin{cases} 0, OntoSemSIM(e1,e2)<t_1 \\ 1, OntoSemSIM(e1,e2)>t_2 \\ \lambda_1 OntoSemSIM(e1,e2)+\lambda_2 OntoStruSIM(e1,e2), \text{otherwise} \end{cases}$$

where,

(1) $OntoSemSIM(e1,e2) = Min(HSemSim_0(e1,e2), HSemSim_1(e1,e2)),....HSemSim_N(e1,e2))$

where, N is the number of semantical matchers that return SIM value bigger than -1 in OntoSIM.

(2) $OntoStruSIM(e1,e2) = \frac{\sum_{i=0}^{i=M} W_i \times HStruSim_i(e1,e2)}{\sum_{i=0}^{i=M} W_i}$

where, M is the number of structural matchers that return SIM value bigger than -1 in OntoSIM. W_i is the weight for each individual structural measure.

(3) $t_1 = 0.3,\ t_2 = 0.95,\ \lambda_1 = 0.6,\ \lambda_2 = 0.4$.

The OntoSIM formula shows the point of view that there is no need to check structural information if the semantic SIM value is too low or high enough.

Definition 6 (Command Conflict '@'): Given 2 commands oc and oc' submitted by different users. oc@ oc' holds, if

(1) $DR(oc,oc') <> NULL, where, DR(oc,oc') = IR(oc) \cap IR(oc)$

(2) $\exists e \in DR(oc,oc'), s.t.\ OntoSIM(e1,e2)<t,\ t \in [0,1]$. where e1 is the entity produced from e by oc, e2 is the entity produced from e by oc' independently.

C. Checking Algorithm

This section presents two algorithms for detecting collaborative conflicts. The first one is simple. The second is high effective. We will give their performance comparison in *section V. B.*

Algorithm1 lists the simple algorithm, the main effort locates in line 0, scan the pool by two loops, whose complexity is O(n ^{2}). Other works' complexity are all O(C), therefore the total complexity is O(n ^{2}).

Algorithm 1. Simple Algoritm

Input Data: All Commands in **Command-Pool**

Result: three conflict sets with initialization *conflicSet*={},

```
0   scan Command-Pool by two loops and get all possible command pair <c,c'> where c is different from c'
1   for each command pair <c,c'> do
2       calculate intersection entity set $S$:= (IR($c$) $\cap$ IR($c'$))      /* IR is the impact range of a command */
3       for each entity $e$ in $S$ do
4           if (OntoSIM($c$.execute($e$) , $c'$.execute($e$))<$T$) conflicSet:= conflicSet $\cup$ {<c,c'>}
5       end;
6   end ;
```

Algorithm 2. Collaborative Conflict Detector (CCD)

Input Data: All Commands in **Command-Pool**

Result: three conflict sets with initialization *conflicSet*

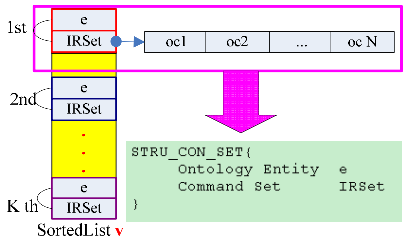
```
0   create and init a Sorted-List<STRU_CON_SET > v.
1   for each command c in Command-Pool do
2       for each entity e in IR(c) do                          /* IR is the impact range of command */
3           STRU_CON_SET scs :=do binary search in v by e, if not found create a new and insert into v
4           for each command c' in.the IRSet of scs do
5               if (OntoSIM(c.execute(e) , c'.execute(e))<T) conflicSet:= conflicSet $\cup$ {<c,c'>}
6           end
7           scs.IRSet:= scs.IRSet $\cup$ {oc}
8       end
9   end
```

In order to detect conflicts effectively, we define a data structure STRU_CON_SET as Fig. 5 shows. Each instance of it holds an ontology entity e, and a commands set IRSet, where, $\forall oc \in \mathrm{IRSet} : e \in \mathrm{IR}(oc)$, i.e IRSet is a set that stores all commands whose impact range include e. All STRU_CON_SET objects are stored in SortedList v. Given a STRU_CON_SET *scs*, we have:

Theorem 1: $\forall oc, oc' \in \mathrm{scs.IRSet}$, oc@oc' if OTC(oc,oc') not holds and OntoSIM(e1,e2)<t, where e1 is the entity produced by oc from scs.e, e2 is the entity after executing oc' from scs.e independently.

Therefore, Algorithm 2 presents a new algorithm that is high effective named Collaborative Conflict Detector (CCD).

Complexity Analysis: As we can see, there are 2 loops. The complexity of first loop is O(n). While in the second loop, 1) based on our experiments, the mean size of **IRSet** is closes to n'= Log_{10} (n). 2)the binary search algorithm's complexity is O(Log_2 (n)). Therefore, the total complexity is O(n)*O(Log_2 (n)). In the worst case is O(n^2). The cost is the extra spending of sorted List whose space complexity is O(n).



```
STRU_CON_SET{
     Ontology Entity  e
     Command Set      IRSet
}
```

Fig. 5. New Data Structure

V. Case Study and Experimental Design

Table 6. Some Detail of Experiment Data

Entity	Number	Entity	Number
Class	82	Data-Property	105
Individual	233	Data-Range	176
Restriction	113	Object-Property	212
FunctionalProperties	13	TransitiveProperties	14
SymmetricProperties	12	InverseFunctionalProperties	9

We developed our system by Java1.5 and MySql. Client runs on web, and the server was conducted on Intel Centrino Duo T2400 1.83GHz PC with 2GB RAM, running WindowsXP sp2. In order to evaluate, we organize 20 students who know some knowledge about agriculture to do tests without communication. Table 6 lists the detail of the experiment data, which comes from FAO (Food and Agriculture Organization).

A. Experiment 1

In order to evaluate we compare the manually determined real conflicts(R) against the detected result P returned by our method, and determine the true positives, I, as well as the false positives, F=P-I. According to the cardinalities of these sets, we have two quality measures, and Fig. 6 shows the result.

(1) $\mathrm{Precision} = \frac{|I|}{|P|} = \frac{|I|}{|I|+|F|}$ estimates the reliability of the detection predictions.

(2) $\mathrm{Recall} = |I| / |R|$ specifies the share of real conflicts that is found.

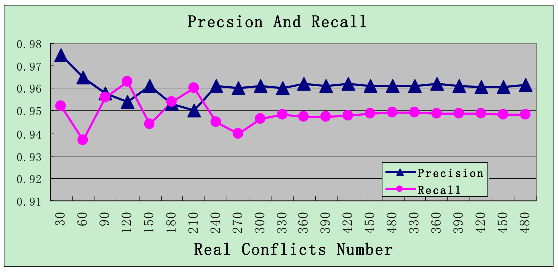


Fig. 6. Conflicts Detection Precision and Recall

Analysis: From fig 5, we see that at the start of X axes, both precision and recall change radically, we believe that is because of the data for analysis is not enough. With conflicts number increasing, the results converge to a constant, and we consider the constant is the real data we need. The recall and precision cannot reach 100%, we believe that is because of Impact Range functions we define are not enough.

B. *Experiment 2*

This experiment compares the running of two algorithms by increasing command numbers, as fig.7 shows.

Analysis: 1) It clearly shows that with commands increasing, the running time of CCD increases flatly, while the performance of optimized simple algorithm is far worse than CCD. 2) According to the analysis in *section IV.C*, the running time of Simple algorithm should be (N/ Log ^{2} (N)) times of the time of CCD. From this figure, we can see that the experiment result complies with it perfectly.

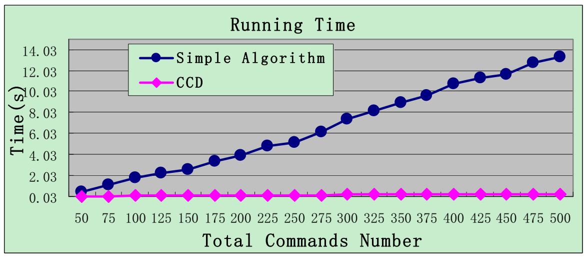


Fig. 7. Running Time Comparison Graph

VI. Conclusion and Future Work

In this paper, we establish a tool for collaborative ontology construction with collaborative conflict detection, the main contributions are: 1) differentiate three categories of conflicts in the collaborative process, and deal with them by with different methods. 2) A formal method to detect collaborative conflicts with a high effective algorithm. 2) Provide evaluation of our algorithm to show its efficiency. In the future, we will adapt the method to be a real time agent for more efficiency.

Acknowledgment

This work is supported by the National High Technology Development (863) Program of China under Grant No. 2007AA01Z179.

The authors would like to thank Dr. Guilin Qi, Institute AIFB University of Karlsruhe, for his insightful suggestions and help on an earlier draft of the paper.

References

[1]. M. Denny. Ontology building: A survey of editing tools. Technical report, O'Reilly XML.com, November 06, 2002.
[2]. Ontoweb. Deliverable 1.3: A survey on ontology tools. http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OntoWeb_Del_1-3.pdf
[3]. Chengzheng Sun etc. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. 1998 ACM Transactions on Computer-Human Interaction
[4]. Pellet : http://pellet.owldl.com/docs/
[5]. E. Bozsak etc. Kaon - towards a large scale semantic web. In K. Bauknecht, A. M. Tjoa, and G. Quirchmayr, editors, E-Commerce and Web
[6]. Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence,France, September 2-6, 2002, Proceedings, volume 2455 of Lecture Notes in Computer Science, pages 304{313. Springer, 2002.
[7]. FIDGE, C. 1988. Timestamps in message-passing systems that preserve the partial ordering. In Proceedings of the 11th Australian Computer Science Conference. 56–66.
[8]. N. F. Noy etc. Musen. Creating Semantic Web Contents with Protege-2000. IEEE Intelligent Systems 16(2):60-71 2001.
[9]. Suk-Hyung Hwang, etc. A FCA-Based Ontology Construction for the Design of Class Hierarchy. ICCSA 2005, Springer LNCS 3482, 2005.
[10]. Clyde W. Holsapple, K. D. Joshi. A Collaborative Approach to Ontology Design. Communications of the ACM, Vol. 45, No. 2, 2002.
[11]. J. Domingue, Tadzebao and Webonto: Discussing, Browsing and Editing Ontologies on the Web. In Proceedings of the Eleventh Knowledge Acquisition Workshop (KAW98, Banff, 1998).
[12]. J. Bao, V.G. Honavar, "Collaborative Ontology Building with Wiki@nt" Third International Workshop on Evaluation of Ontology Building Tools, Hiroshima 2004.
[13]. ELLIS, C. A.. etc J. 1989. Concurrency control in groupware systems. In Proceedings of the ACM SIGMOD Conference on Management of Data (May). 399–407.
[14]. Ontology editor survey results. http://www.xml.com/2002/11/06/Ontology_Editor_Survey.html.
[15]. Web—ontology working group.OWL Web ontology Language Overview. http://www. w3. org/TR/2003/PR--- owl-- features –20031215/(Accessed Feb.02，2004)
[16]. Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string metrics for matching names and records. In: Proc. KDD-2003 Workshop on Data Cleaning and Object Consolidation. (2003)
[17]. H.Zhuge, The Knowledge Grid, World Scientific Publishing Co., Singapore, 2004. H.Zhuge, China's E-Science Knowledge Grid Environment, IEEE Intelligent Systems, 19 (1) (2004) 13-17。