An Approach to Detect Collaborative Conflicts for Ontology Development

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

Abstract. Ontology has been widely adopted as the basis of knowledge sharing and knowledge-based public services. However, ontology construction is a big challenge, especially in collaborative ontology development, in which conflicts are often a problem. Traditional collaborative methods are suitable for centralized teamwork only, and are ineffective if the ontology is developed and maintained by mass broadly distributed participators lacking communications. In this kind of highly collaborative ontology development, automated conflicts detection is essential. In this paper, we propose an approach to classify and detect collaborative conflicts according to some mechanisms: 1) impact range of a revision, 2) semantic rules, and 3) heuristic similarity measures. Also we present a high effective detecting algorithm with evaluation.

1 Introduction

Ontology has been widely adopted as the basis of knowledge sharing and knowledge-based public services. In this kind of knowledge-based systems, a rich and high-quality ontology model is the premise of satisfactory knowledge services. Therefore, besides ontology-based knowledge service, it is also important to get support for ontology creation and maintenance. In fact, ontology construction itself is a big challenge, especially when the ontology is expected to have relevance and value to a broad audience[1].

In most cases, the ontology is constructed manually by centralized teams, which is a complex, expensive and time-consuming process (e.g. [10]). In recent years, some methods and tools 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.

However, collaborative developments are often accompanied by conflicts. In traditional way, collaborative conflicts are usually handled by mechanisms of locking or branching/merging provided by version management systems (e.g. CVS). The conflicts occur when two developers revise the same file in a same phase. However, in ontology, the basic units are concepts and relationships between them, without explicit file-based artifact. Therefore, conflicts in this environment much more likely exist. The rich semantic relationships in ontology model exacerbate the situation, since different parts of the ontology model have much more complicated impacts on each other. On the other hand, in large-scale collaboration-based ontology development, there are many participators, most of them are not aware of the

existences of their co-workers at all, let alone communicate enough, this exacerbates the problem. Therefore, it is obvious that effective conflicts detection and resolution are essential for large-scale collaborative ontology development. For these, in this paper, we propose an approach to classify and detect three kinds of collaborative conflicts according to some mechanisms. Also we present a high effective detecting algorithm with evaluation.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 addresses the overview of our framework. Section 4 proposes an approach to detect collaborative conflicts. Section 5 shows experiments with evaluation. The last section is our conclusion.

2 Related Works

KAON [5] focuses on that changes in ontology can cause inconsistencies, and proposes deriving evolution strategies in order to maintain consistencies. Protégé [8] is an established line of tools for knowledge acquisition, which is constructed in an open, modular fashion. OntoEdit [7] supports the development and maintenance of ontologies, with an inferencing plugin for consistency checking, classification and execution of rules. OntoWiki [3] fosters social collaboration aspects by keeping track of changes, allowing 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. OILEd [12] is a graphical ontology editor that allows the user to build ontologies with FACT reasoner [2] to classify ontologies and check consistency.

As far as conflicts detection and resolve are concerned, some works, such as [5] provides concurrent access control with transaction oriented locking, Noy[8] provides discussion thread for users to communicate. In some cases, even rollback. SCROL[11] provides a systematic method for automatically detecting and resolving various semantic conflicts in heterogeneous databases with a dynamic mechanism of comparing and manipulating contextual knowledge of each information source. Cupid [6] is an approach uses a thesaurus to identify linguistic matching and structure matching. Peter Haase[4] discusses the consistent evolution of OWL ontologies, and presents a model considering structural, logical, and user-defined consistency.

Almost all current ontology construction tools provide functionality for consistency checking. But based on our humble knowledge, none of them distinguishes collaborative conflicts from logical inconsistency. It is helpful to differentiate them from each other, because some collaborative conflicts may not result in logical inconsistency, but violate user defined rules or other exception (we will explain in section 3.3). Furthermore, concurrent access control with transaction oriented locking as database is not suitable for ontology, for the rich semantic relationships among ontology entities. Therefore, we need new approach to overcome these deficiencies.

3 Overview of Our Approach

3.1 System Framework

Fig.1 shows a framework of our collaborative ontology building tools. It adopts B/S architecture, and uses OWL[9] as ontology language. There are two parts, one is

private workspace, each user has his own private workspace, and the other is public workspace. When a user logins to his own private workspace successfully, he could select an ontology segment from server. All revisions the user performs will be transferred into a Command-Package and stored until submission. In the public workspace, there are 5 main processes: 1) a command package pool stores all packages received from each user. 2) For every period, the collaborative conflicts checking process, which includes three sub-processes, i.e. hard soft and latent conflicts checking. All conflicts will be handled by Conflicts Handler, in this paper we omit the detail of the process. 3) The 2nd checking is consistency checking, which deals with logical inconsistency, and all inconsistency will be handled by another handler which is also omitted in this paper. 4) After 2 proceeding checking, some mistakes still remain for they can not be found automatically. Therefore, experts' review is necessary. 5) Lastly, all correct revisions will be executed to update ontology base. In this paper, we focus on collaborative conflicts detection.

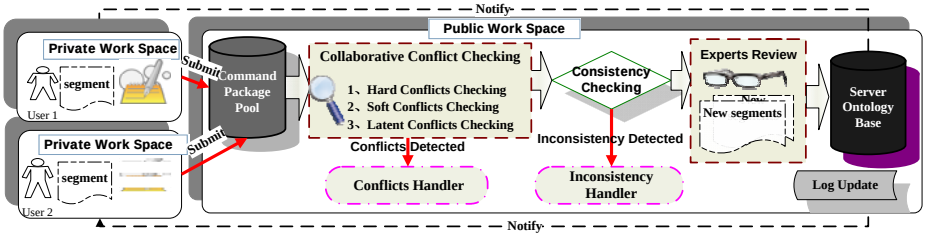

Fig. 1. Overview of Collaborative Ontology Building Process

3.2 Ontology Commands

In our method, we provide a set of commands named Onto-Command for users to modify the ontology content, each of which is composed of an operation and some operands (ontology entity). Table 1 lists a part of atomic commands.

Definition 1(OC:Onto-Command). OC=<*Name*,*E*,*V*> is a function s.t. E→E', where, *Name* is the command name, *E* is an ontology entity set , *V* is the value of parameter.

Our system differentiates these commands into three kinds:

$$\mathrm{OCKind}(oc) = \begin{cases} \mathrm{DEL}, oc \in \{\mathrm{DelClass}, \mathrm{DelIndividual}, \mathrm{DelRestriction}, \mathrm{DelProperty}\} \\ \mathrm{ADD}, oc \in \{\ \mathrm{AddClass},\ \mathrm{AddIndividual}, \mathrm{AddRest}, \mathrm{AddProperty}\ \} \\ \mathrm{MOD}, \mathrm{otherwise} \end{cases}$$

where, *DEL* means commands of this kind is for deleting entity, *ADD* is for adding, and *MOD* is for modifying.

Table 1: A Part of Ontology Commands

Name	Parameter (E)
AddClass/DelClass	(aClass)
AddSubClass/DelSubClassRelation/DelEquClass/DelSameClass	(aClass,supClass)
AddDataProperty/AddObjProperty/DelObjProperty	(aProp)
AddDifferentClassRelation/AddEquivalentClass/AddSameClass	(aClass1,aClass2)
AddDifferentIndividual/AddEquivalentIndividual	(aInd1,aInd2)

3.3 Collaborative Conflicts and Logical Inconsistency

As mentioned in the last paragraph of section 2, some 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 moves *Apple* to be a subclass of *Computer*, as showed in fig.2(b). There is no inconsistency at all in fig.2(b), but what the 1st participator wants is '*Rat eats a kind plant*', instead of '*Rat eats a kind Computer*'. Therefore, in our opinion, collaborative conflicts should be handled different from logical inconsistency.

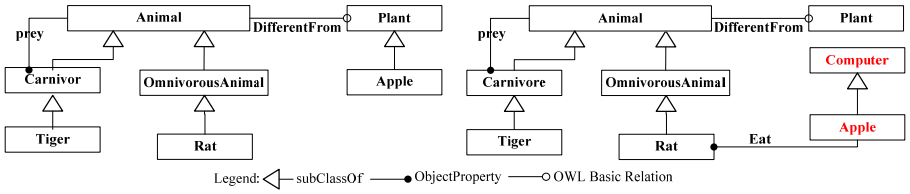

Fig. 2 (a) Original Ontology Segment. (b)Result of Accepting All Revisions

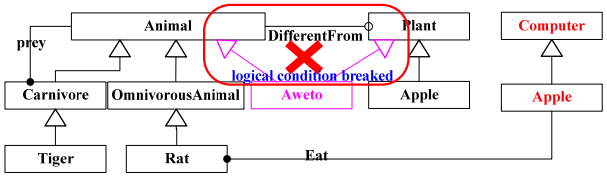

Fig 3. An Example of Logical Inconsistency.

Definition 2(Collaborative Conflicts): 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 its original semantic.

Definition 3 (Logical Inconsistency): We call a revision on ontology O, is inconsistent 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.

For example, suppose one participator adds a concept *Aweto*, and then makes it as a subclass of both *Animal* and *Plant* as fig.3 shows. It is obvious that these changes breaks the logical condition '*Animal owl:differentFrom Plant*'.

4 Collaborative Conflicts Detection

In this section we address how to detect collaborative conflicts. Firstly we introduce Command Impact Range, and base on it we classify 3 kinds of collaborative conflicts, and finally propose two algorithms with comparison between them.

4.1 Command Impact Range

Because subtle semantic relationship exists among entities in ontology, change one entity may produce chain reaction. This means that for each revision, there is a set of entities may be impacted. Therefore, for each atomic Onto-Command we define a unique **IR(Impact Range)** value, table 2 lists some of them.

Look back on fig.2(b), for example, if a user uses *DelClass* command *oc* to delete class '*Carnivore'*. According to table 2, **IR(***oc***)**={*Carnivore*, *prey*, *Tiger* }.

Table 2. A Part of IRs Value

OC Name	E	IR(oc)
DelClass	(aClass)	Return **E** ⋃ {o| o.Domain=aClass } ⋃ { o| o.Range=aClass } ⋃ …
DelInstance	(aIndividual)	Return **E** ⋃ { ins| ins is the same as aIndividual }
DelProperty	(aProperty)	Return **E** ⋃ {r| r.actOn(aProperty)= true, r is Restriction }
AddObjProperty	(class1,class2)	Return **E** ⋃ {class| (aClass.directSubclassOf(aClass2)} …

4.2 Three Kinds of Collaborative Conflicts

In our method, according different criterions, we classify three kinds of collaborative conflicts, i.e. hard soft and latent conflicts. They are following.

4.2.1 Hard Conflicts

Hard conflicts are easy to detect according to the impact range and command type. Before go on we will introduce two functions, which are used to judge whether it is possible for two commands conflict with each other.

OTC (OC Type Conflicts Table) is a function table returns whether it is possible for two commands (oc and oc') conflict with each other. As table 3 lists, F/T is false/true, which means that it is impossible/possible for oc directly conflicts with oc'.

In our opinion, if two participators perform the same operation on the same entity with same value, we will unite two operations as one. Therefore, there is no conflict if both participators perform commands to delete or add the same class. i.e., *OTC(DEL,DEL)=F* and *OTC(ADD,ADD)=F*.

Table 3. Onto-Command Type Conflicts Rules

OTC(oc,oc')		OCKind(oc')		
		MOD	ADD	DEL
OCKind(oc)	MOD	**OTC_MOD**(oc,oc')	F	T
	ADD	F	F	T
	DEL	T	T	F

Table 4. A Part of OTC_MOD Conflicts Rules

oc	oc'	OTC_MOD(oc,oc')
AddEquivalentIndividual	AddDifferentIndividual	If (oc.E=oc'.E) T Else F
AddEquivalentClass	AddDifferentClassRelation	If (oc.E=oc'.E) T Else F
AddEquivalentClass	AddDisjointWithRelation	If (oc.E=oc'.E) T Else F
AddEquivalentProperty	AddDifferentPropertyRelation	If (oc.E=oc'.E) T Else F

OTC_MOD (MOD OC Conflicts Table) is a function table returns whether it is possible for two *MOD* commands (i.e. OCKind(*oc*) =*MOD*) directly conflict with each other. Table 4 lists a part of them.

For example, if one participator use command *AddEquivalentClass* oc1 to make *class1* and *class2* equivalent, while the other use command *AddDifferentClass* oc2 to make *class1* and *class2* different, then *OTC_MOD(oc1,oc2)=T*.

Definition 4 (Hard Conflicts '@'): Given 2 commands *oc* and *oc'* performed by different users, *oc@oc'* if **OTC**(*oc,oc'*) $\wedge$ $(\mathrm{IR}(\mathrm{oc}) \cap \mathrm{IR}(\mathrm{oc}')) \neq \mathrm{NULL}$

For example, look back on fig.2(b), if user U1 performs a *DelCalss* command oc for delete class *Carnivore*, and user U2 performs *AddSubClass* command oc' to add a subclass *Lion* to *Carnivore*. In this case, $\mathrm{DR}(\mathrm{oc}, \mathrm{oc}') = \mathrm{IR}(\mathrm{oc}) \cap \mathrm{IR}(\mathrm{oc}') = \{Carnivore\}$ and OTC(oc,oc') holds, then oc@oc', that means oc conflicts with oc' directly.

4.2.1 Soft Conflicts

In some cases, there is nothing wrong if both revisions happen alone, but would result in inconsistency if both were accepted. Therefore, we define **Semantic-Rule-Set** which includes a set of semantic rules, in order to deal with these cases. Table 5 lists a part of these rules.

Definition 5 (Semantic Rules): Given an entity *e* and two commands *oc* and *oc'*, where, $\mathrm{e} \in \mathrm{IR}(\mathrm{oc}) \cap \mathrm{IR}(\mathrm{oc}')$. **SEM** is a semantic rule judges whether *e1* is inconsistent with *e2*, where *e1* and *e2* originate from *e* by executing *oc* and *oc'* respectively.

Table 5. A Part of Semantic Rules in Semantic-Rule-Set

SEM	Description
Same VS Different	Check whether there is an entity is the same as e1 but different from e2
Same VS DisjointWith	Check whether there is an entity is the same as e1 but disjoint with from e2
Functional Same VS Different	Check whether there is an entity is the same as e1 but different from e2 according the principle of functional object property

For example, ***Functional Same VS Different*** is one semantic rule, whose principle is that for a functional object property **P** in OWL, **Y=Z** if **P(X, Y)** and **P(X, Z)**. Therefore,

$$\mathrm{SEM}_{\mathrm{functional\&diff}}(\mathrm{e,oc,oc'}) = \begin{cases} 0, \exists \mathrm{c,s.t.\ FunctionalSameAs(c,e1)} \wedge \mathrm{c.differentFrom(e2)} \\ 0, \exists \mathrm{c,s.t.\ FunctionalSameAs(c,e2)} \wedge \mathrm{c.differentFrom(e1)} \\ 1, \text{ otherwise} \end{cases}$$

, where,

(a) differentFrom means '*owl:differentFrom' or 'owl:disjointWith'*.

(b) *e1* and *e2* originate from *e* by executing *oc* and *oc'* respectively.

(c) $\mathrm{FunctionalSameAs(c1,c2)} = \begin{cases} 1, \exists \mathrm{p}, \exists \mathrm{c}, \text{ s.t., } \mathrm{p(c, c1)} \wedge \mathrm{p(c, c2)}, \text{ where, p is a functional property} \\ 0, \text{ otherwise} \end{cases}$

Definition 6 (Soft Conflicts '#'): Given 2 commands oc and oc' performed by different users, oc#oc' if $\neg \mathrm{OTC(oc,oc')} \wedge (\exists \mathrm{SEM}, \exists \mathrm{e}, \text{ s.t. } \mathrm{SEM(e,oc,oc')}=0)$, where, $\mathrm{e} \in \mathrm{IR}(\mathrm{oc}) \cap \mathrm{IR}(\mathrm{oc}'), \mathrm{SEM} \in$ **Semantic-Rule-Set**.

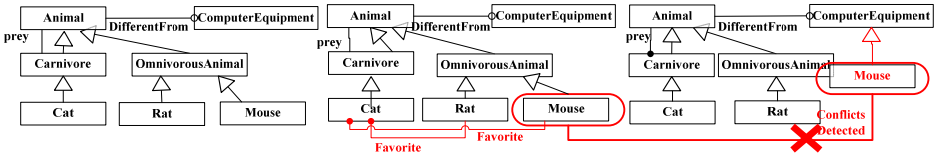

Fig. 4. (a) Another Segment. (b)Result Revised By U1. (c) Result Revised By U2

For example, fig. 4 (a) shows another ontology segment. If participator U1 adds a functional object property *Favorite* from *Cat* to *Rat* and *Mouse* by command *oc*, according to the principle of functional object property, we can infer that *Rat* is the same as *Mouse*. Meanwhile participator U2 moves *Mouse* to be a subclass of *ComputerEquipment* that is different from *Animal* by command *oc'*. We can infer that it is also different from *Rat*. i.e. $SEM_{functional\ \&diff}(e, oc, oc') = 0$, and $\neg$ OTC(oc,oc') therefore, *oc#oc'* . Fig.4 (b) and (c) show the result.

4.2.2 Latent Conflicts

Different from hard and soft conflicts, which can be detected explicitly, some conflicts are not so easy to judge. Look back on the first example in section 3.3, 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. 5 shows the comparison, where (a) shows the acceptable result, and (b) shows unacceptable case. In this case, it is not easy to judge whether the changes on concept *Apple* is acceptable or not, for lacking absolute standard. Therefore, heuristic measures must be adopted to deal. In our system, we define a set of similarity measures, and differentiate them into two types. Table 6 lists a part of them.

Two Types of Heuristic Similarity Measures:

(1) **HSemSIM**: calculates similarity based on semantic information.

(2) **HStruSIM**: calculates similarity based on the structure information.

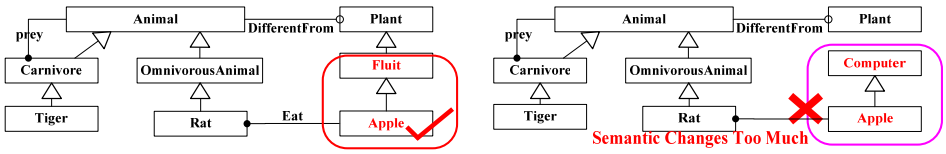

Fig. 5. (a) Acceptable Result. (b) Unacceptable Result.

Table 6. A Part of Characteristic and Heuristic Similarity Measures

Measure type	Measure	Heuristic matcher Description
Semantical	Entity Name	Compare the name 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 On Property	Compare matched property of e1 and e2
Structural	Direct Sup/Subproperty	Compare matched direct sup/subproperty of e1 and e2
	Direct Sup/Subclass	Compare matched direct sup/subclass of e1 and e2
	Depth Distance	Compare hierarchy distance of e1 and e2

For example, ***Depth Distance*** is one structural measure that calculates similarity between two ontology classes or two object properties, based on the fact that the deeper the two entities locate in one hierarchy, the higher the similarity is. Therefore,

$$\mathrm{HStruSim}_{\mathrm{DepthDistance}}(e1,e2) = \frac{2 \times \mathrm{depth}(\mathrm{LCA}(e1,e2))}{\mathrm{depth}(e1) + \mathrm{depth}(e2)}$$

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

Similarity Aggregation: for all measures, **OntoSIM** is given:

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

where, $\mathrm{OntoSemSIM}(e1,e2) = \mathrm{Min}(\mathrm{HSemSim}_0(e1,e2), \mathrm{HSemSim}_1(e1,e2)), \ldots \mathrm{HSemSim}_N(e1,e2))$

$$\mathrm{OntoStruSIM}(e1,e2) = \frac{\sum_{i=0}^{i=M} W_i \times \mathrm{HStruSim}_i(e1,e2)}{\sum_{i=0}^{i=M} W_i}$$

$$t_1 = 0.3, \quad t_2 = 0.95, \quad \lambda_1 = 0.6, \quad \lambda_2 = 0.4$$

where, N is the number of semantical matchers. M is the number of structural matchers. W_i is the weight of each individual structural measure.

Definition 7 (Latent Conflicts'!!'): Given 2 commands *oc* and *oc'* performed by different users. *oc*!!*oc'* if $\neg$ OTC(*oc*,*oc'*) $\wedge$ $\exists e \in (\mathrm{IR}(oc) \cap \mathrm{IR}(oc'))$,s.t. OntoSIM(*e1*,*e2*)<t, $t \in [0,1]$, where *e1* and *e2* originate from *e* by executing *oc* and *oc'* respectively.

4.3 Checking Algorithm

This section presents two algorithms for collaborative conflicts detection. The 1st is simple with high complexity that is unacceptable. The 2nd is high effective named CCD. We will give their performance comparison with experiments in section 5.3.

4.3.1 Simple Algorithm

Algorithm 1 lists the simple algorithm. It scans command-pool by 2 loops and gets all possible command pairs. For each pair, we do three kinds of conflicts checking, and each detected conflict will be added to appropriate conflicts set.

Algorithm 1. Simple Conflicts Detector Pseudo Code

Input Data: All Commands in **Command-Pool**
Result: three conflicts sets with initialization *hardConflicSet*=**{},** *softConflictSet*=**{},** *latentConflictSet*=**{}**

```
0   scan Command-Pool by 2 loops and get all possible 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)∩IR(c'))
3     if (OTC(c, c') and S is not empty) hardConflictSet := hardConflicSet∪{<c,c'>}
4     for each entity e in S do
5       for each semantic rule SEM in Semantic-Rules-Set do
6         if (SEM (e,c,c') is zero) softConflictSet := softConflictSet∪{<c,c'>}
7       end
8       if (OntoSIM(c.execute(e) , c'.execute(e))<T) latentConflictSet:= latentConflictSet∪{<c,c'>}
9     end;
10  end ;
```

Complexity Analysis: Because 1) in line 0, it scans pool by two loops which makes complexity $O(n^2)$. 2) The number of entities that one command may impact is finite (by our statistic, none exceeds 100), i.e. O(IR)=O(C). 3) The number of *SEMs* in Semantic-Rules-Set and the number of heuristic similarity measures in OntoSIM are both finite. Therefore, the total complexity is $O(n^2)$, this is unacceptable.

4.3.2 High Effective Collaborative Conflicts Detection Algorithm (CCD)

In order to detect conflicts effectively, we define a structure STRU_CON_SET (see fig.6). Each instance of this type holds an entity *e*, and a command set *IRSet* , where, $\forall oc \in IRSet$ s.t. $e \in IR(oc)$, i.e *IRSet* stores all commands whose impact range includes *e*. All instances are stored and sorted by *e* in Sorted-List v.

Theorem 1: $\forall oc, oc' \in scs.IRSet$, oc@oc' if OTC(oc,oc') holds.

Theorem 2: $\forall oc, oc' \in scs.IRSet$, oc#oc' if $\neg OTC(oc, oc) \wedge \exists SEM, s.t. SEM(scs.e, oc, oc') = 0$.

Theorem 3: $\forall oc, oc' \in scs.IRSet$,oc!!oc' if $\neg$ OTC(oc,oc') $\wedge$ OntoSIM(e1,e2)<t,where *e1* and *e2* are produced from *e* by executing *oc* and *oc'* respectively.

Algorithm 2 presents a high effective checking algorithm named **CCD**. It scans Command-Pool only once, for each command *c* in the pool, we compare it with candidate commands that may conflict with *c* in Sorted-List *v* by binary searching. The remainder is similar with Algorithm 1.

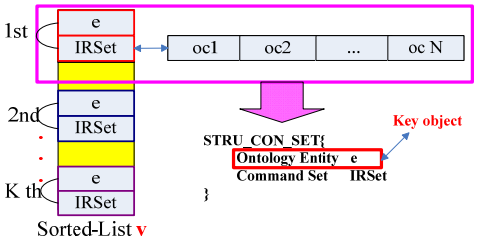

Fig. 6. STRU_CON_SET Data Structure

Algorithm 2. CCD Pseudo Code

Input Data: All Commands in **Command-Pool**
Result: three conflicts sets with initialization *hardConflicSet*=**{},** *softConflictSet*=**{},** *latentConflictSet*=**{}**

```
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
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. IRSet of scs do
5               if (OTC(c, c')) do hardConflictSet := hardConflictSet⋃{<c,c'>} and get next c'
6               for each semantic rule SEM in Semantic-Rules-Set do
7                   if (SEM (e,c, c') is zero) softConflictSet := softConflictSet⋃{<c,c'>} and get next c'
8               if (OntoSIM(c.execute(e) , c'.execute(e))<T) latentConflictSet:= latentConflictSet⋃{<c,c'>}
9           end
10          scs.IRSet:= scs.IRSet ⋃{oc}
11      end
12  end
```

Complexity Analysis: As stated in Algorithm 2, in line 1 it scans the pool by a loop, whose complexity is O(n). In line 3 the binary retrieval's complexity is $O(\mathrm{Log}_2(n))$. Other loops' complexities are all O(C), just the same as Algorithm 1. Therefore, the total complexity of CCD is $O(n)*O(\mathrm{Log}_2(n))$. In the worst case is $O(n^2)$. The cost is the extra spending of *v* whose space complexity is O(n).

5 Case Study and Experimental Design

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. Table 7 lists the detail of the experiment data, which comes from FAO (Food and Agriculture Organization).

Table 7. Experiment Data for Collaborative Conflicts Detection

Entity type	Number	Entity type	Number	Entity type	Number
Class	82	TransitivePropertiy	14	FunctionalProperty	13
Individual	233	FunctionalProperty	13	InverseFunctionalProperty	9
Restriction	113	Data-Property	105	SymmetricProperty	12
ObjectProperty	212	Data-Range	176	InverseFunctionalProperty	9

5.1 Experiment 1

In order to evaluate, we compare the manually determined real conflicts(R) against the detected result P returned by **CCD**, and determine the true positives, I, as well as false positives, F=P-I. Then we have: (1) **Recall**=|**I**|/|**R**|, (2) **Precision** =|**I**|/|**P**|.

5.1.1 Hard and Soft Conflicts Detection Result

This experiment is to evaluate the hard and soft conflicts checking algorithm, fig. 7 (a) and (b) show the result.

Analysis: From fig.7 we can infer, 1) the algorithm is good enough for detecting soft conflicts. 2) As far as the hard conflicts are concerned, at the start of X axes, both precision and recall change radically, because the data for analysis is not enough. 3) With number increasing, the precision/recall of hard conflicts converge 95%/94%, and we consider the constant is the real data we need.

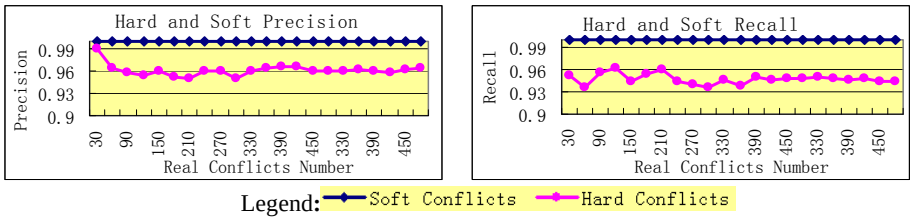

Fig.7.(a) Hard and Soft Precision Graph (b) Hard and Soft Recall Graph

5.1.2 Latent Conflicts Detection Result

This experiment is to analysis the impact of heuristic matchers on latent conflicts detection result. The threshold is 0.65. Firstly there are few matchers in **OntoSIM**, and then add a number, finally we add enough. Fig. 8 (a) and (b) show the result.

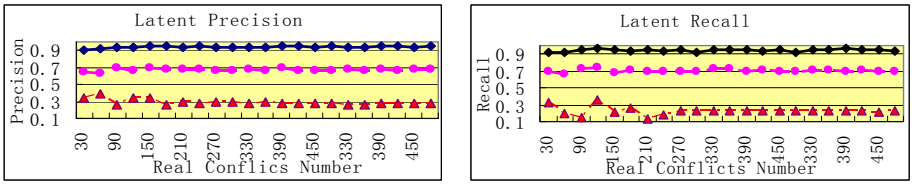

Fig.8. (a) Latent Precision Graph (b) Latent Recall Graph

Analysis: From the two figures, we can infer, 1) at the start of X axes, the result of '*Few Matcher*' changes radically. We also believe that is because of the data for analysis is not enough, 2) with more and more conflicts coming, the results converge to a constant, and we consider the constant is the real data we need, too. 3) '*Enough Matchers*' produces higher precision and recall than '*Few Matcher*' and '*A Number of Matchers*'. Therefore, we can say that heuristic matchers play an important role in the collaborative conflicts detecting process.

5.2 **Experiment 2**

In this experiment, we adopt 'Enough Matchers' in OntoSIM with threshold 0.65, and make a running time comparison between **Simple** and **CCD** algorithm. Table 8 and fig.9 present the performance comparison.

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 4.3.1 and 4.3.2, the running time of Simple algorithm should be $(N/\text{Log}_2(N))$ times of the time of CCD. From the last two lines of table 8, we can see that the experiment result complies with it perfectly. 3) Notice that the running time of Simple algorithm does not increase by the way of N^2 parabola, this is because we optimize the checking algorithm.

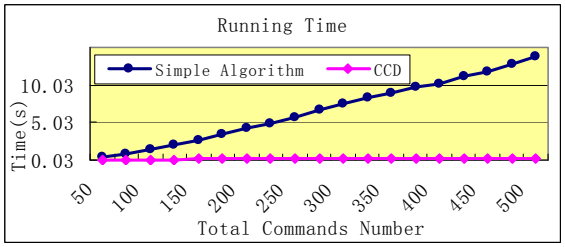

Fig. 9. Running Time Comparison between Simple Algorithm and CCD

Table 8. Running Time and Comparison between Two Algorithms (time is in seconds)

Comparison	N (Commands Number)						
	200	250	300	350	400	450	500
Running time of Simple(s)	4.6079	5.9016	7.4763	8.9776	10.1837	11.492	13.869
Running time of CCD (s)	0.1605	0.1746	0.1976	0.2134	0.2192	0.2237	0.2374
Time(Simple)/Time(CCD)	28.70	33.79	37.81	42.05	46.44	51.36	58.41
$N/\text{Log}_2(N)$	26.16	31.51	36.46	41.41	46.27	51.06	55.76

6 Conclusion and Future Work

Ontology construction itself is a big challenge, especially in large-scale collaboration-based ontology development, because, 1) in ontology, the basic units are concepts and relationships between them, without explicit file-based artifact, which makes conflicts much more likely exist; 2) Most of participators are not aware of the existences of their co-workers at all, let alone have enough sufficient communications. Therefore, it is obvious that effective conflicts detection and resolution are essential for large-scale collaborative ontology development.

In this paper, we propose a novel approach to deal with collaborative conflicts in our ontology construction system. The main contributions are: 1) differentiate collaborative conflicts from logical inconsistency. 2) Classify three kinds of collaborative conflicts, i.e. hard conflicts soft conflicts and latent conflicts. 3) Propose a formal method to detect collaborative conflicts with an effective algorithm and evaluation.

In the future, we will adapt the method to be a real time agent for more efficiency.

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

Reference

1. Natalya Fridman Noy, Abhita Chugh, Harith Alani: The CKC Challenge: Exploring Tools for Collaborative Knowledge Construction. IEEE Intelligent Systems 23(1): 64-68 (2008)
2. Horrocks, U. Sattler, S. Tobies. Practical reasoning for expressive description logics. 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99).
3. S. Auer, S. Dietzold, and T. Riechert. OntoWiki—A Tool for Social, Semantic Collaboration, ISWC 06.
4. Peter Haase, Ljiljana Stojanovic. Consistent Evolution of OWL Ontologies. ESWC 2005.
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. J. Madhavan, P.A. Bernstein, and E. Rahm, Generic Schema Matching with Cupid, Proc. VLDB Conf., pp. 49-58, Sept. 2001.
7. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. OntoEdit: Collaborative ontology development for the semantic web. In Proceedings of the first International Semantic Web Conference 2002 (ISWC 2002).
8. N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, & M. A. Musen. Creating Semantic Web Contents with Protege-2000. IEEE Intelligent Systems 16(2):60-71, 2001.
9. Web—ontology working group. OWL Web ontology Language Overview. http://www. w3. org/TR/2003/PR--- owl-- features –20031215
10. Suk-Hyung Hwang, Hong-Gee Kim, and Hae-Sool Yang. A FCA-Based Ontology Construction for the Design of Class Hierarchy. ICCSA 2005.
11. Sudha Ram; Jinsoo Park. Semantic conflict resolution ontology (SCROL): an ontology for detecting and resolving data and schema-level semantic conflicts. Knowledge and Data Engineering, IEEE Transactions on Volume 16, Issue 2, Feb. 2004.
12. Sean Bechhofer, Ian Horrocks, Carole Goble, Robert Stevens. OILEd: a Reason-able Ontology Editor for the Semantic Web. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence.