Computing and Informatics, Vol. 27, 2008, 21–36

DEBUGGING ONTOLOGY MAPPINGS: A STATIC APPROACH

Peng WANG, Baowen XU

School of Computer Science and Engineering
Southeast University
#2, Si Pai Lou
Nanjing, 210096, China
e-mail: pwangseu@gmail.com, bwxu@seu.edu.cn

Revised manuscript received 11 January 2007

Abstract. Ontology mapping is the bottleneck in solving interoperation between Semantic Web applications using heterogeneous ontologies. Many mapping methods have been proposed in recent years, but in practice, it is still difficult to obtain satisfactory mapping results having high precision and recall. Different from existing methods, which focus on finding efficient and effective solutions for the ontology mapping problem, we place emphasis on analyzing the mapping result to detect/diagnose the mapping defects. In this paper, a novel technique called *debugging ontology mappings* is presented. During debugging, some types of mapping errors, such as redundant and inconsistent mappings, can be detected. Some warnings, including imprecise mappings or abnormal mappings, are also locked by analyzing the features of mapping result. More importantly, some errors and warnings can be repaired automatically or can be presented to users with revising suggestions. The experimental results reveal that the ontology debugging technique is promising, and it can improve the quality of mapping result.

Keywords: Ontology mapping, debugging, algorithm

1 INTRODUCTION

Ontologies clarify the structure of domain knowledge and enable knowledge sharing, and they play a crucial role in dealing with heterogeneous and computer-oriented huge amount of data. Ontologies have been used popularly in many fields such as knowledge representation, information retrieval, natural language understanding, biology and e-science. In recent years, the Semantic Web [1], which aims at providing high-quality intelligent ser-

vices on the Web, exploits ontlogies to model the knowledge of various semantic web applications. In turn, the Semantic Web promotes the researches of ontology greatly.

Usually, ontologies are distributedly used and built by different communities. That causes many heterogeneous ontologies in same or relative domains, which is the major obstacle to realize semantic information sharing. Ontology mapping is the main approach to solve the problem through capturing the communication rules between heterogeneous ontologies. Current mapping methods usually employ the technologies such as literal or structure similarity matching [2], machine learning [3], or combining several technologies [4], to compute the similarity between the corresponding entities in different ontologies.

In fact, although many mapping systems have been developed, when they are used in practical applications, they often can't work well as expected, and the precision and recall of mapping result are not always high [5]. In our opinions, there are two main reasons. First, for the variety of the representations and random modeling perspective of actual ontologies, it often lacks enough information for discovering correct mappings. Second, a certain mapping algorithm is often just effective for some types of ontologies but not for all.

Till now, most existing mapping approaches focus on the mapping skills and techniques. However, we find that finding out such mapping result can not be the end of an ontology mapping process. Through examining some initial mapping results, we find an interesting fact: the mapping result often includes error mappings (such as redundant and inconsistent mappings), imprecise mappings (mappings are not the best ones) and abnormal mappings (the behavior of a mapping is strange). Therefore, we want to compensate such gap by detecting the *mapping bugs* and even repairing them if possible. We call such idea *debugging ontology mappings*. Notice that this novel technique is not a new mapping approach but a beneficial complement to the existing ontology mapping methods.

The original contributions of this paper are the following:

1. The ontology mapping debugging idea is proposed for the purpose of improving the quality of mapping result;
2. The efficient methods for detecting and diagnosing mapping bugs (including errors and warnings) are presented;
3. Some bugs would be repaired automatically or be presented to users with generated repair suggestions for the final decision. The suggestions can help users improve or modify the mapping algorithms to avoid such types of errors.
4. Experimental results demonstrate that the debugging technique is promising.

The rest of this paper is organized as follows: After a brief overview of related work (Section 2), we give general ideas about ontology mapping debugging in Section 3. In Section 4 we describe the methods for debugging ontology mapping results. Some experimental results and discussions are presented in Section 5. Section 6 is conclusion.

2 RELATED WORK

Ontology mapping is an open problem. Some ontology mapping solutions have been proposed in recent years, whereas we don't review them here but refer the readers to some comprehensive surveys [6–8]. Most existing works on ontology mapping focus on the mapping algorithms. There are only a few works that address the issues related to the mapping result, but most of them pay attention to evaluating or reusing mappings. Hess showed how to use the given mappings to a third ontology as training data or background knowledge to improve mapping accuracy [9]. Euzenat and Ehrig proposed a more reasonable precision and recall measure to evaluate the mapping result [10, 11]. These works did not discuss the bugs in the mapping result.

Hanif et al. described an approach of detecting and eliminating misalignment at the time of aligning two different ontologies [12]. In their method, two mapping results extracted by different mapping techniques from the same pair of ontology were fed to the misalignment detection and elimination process to produce better alignments. This approach just combines different mapping results as a new one, so it can reduce limitation of a specific technique of ontology alignment. Obviously, this work can not find the mapping bugs in a single mapping result. Furthermore, it did not give a clear definition of what is a misaligned mapping. According to its misalignment definition, a mapping m_i could be regarded as aligned under alignment sets A_S and A_T, but could be misaligned under alignment sets A'_S and A'_T. Therefore, this approach did not adapt to our problem.

The heart of this paper is how to detect and diagnose the bugs of ontology mapping result. To our surprise, we have not found any work that directly addresses how to deal with the problem in the published literature. The most related work may be [14], in which Stuckenschmidt et al. proposed a theory for reasoning about ontology mappings. This work identified four properties that reflect the quality of a mapping, namely *containment*, *minimality*, *consistency* and *embedding*. Then these properties can be decided based on existing reasoning services for distributed description logics. The theory could detect some unsatisfactory and inconsistent mappings, but the authors did not declare their theory would be applied to debug ontology mapping bugs. From the web site of DRAGO project [13], we find a brief introduction of an ontology mapping debugging tool, which can debug the mappings created using CtxMatch matching tool[1]. According to its experimental results, the debugger can detect and remove malicious bridge rules between ontologies. It did not tell what a malicious bridge rule is, and whether the tool can repair some malicious bridge rules. For the reason that there is no corresponding literature we can not know more detail information about this ontology mapping debugging tool. However, we are sure that this work has the similar goal to ours.

Chiticariu, Alex and Tan developed SPIDER [15, 16], which was the first prototype tool for debugging schema mappings. The heart of SPIDER was a data-driven facility for understanding a schema mapping through the display of routes. A route essentially described the relationship between source and target data with the schema mapping. SPIDER was also equipped with "standard" debugging features such as breakpoints, step-by-

[1] `http://dit.unitn.it/~zanobini/downloads.html`

step computation of routes and a "watch" window for visualizing data exchanging and variables used in a dependency at each step. However, SPIDER can not discover and diagnose the bugs in schema mappings. Hence, SPIDER's goal is not the same as ours.

Debugging idea has been used in ontology building process. In order to detect and diagnose the cause of errors in ontologies, some debugging methods based on logic reasoning or heuristic ruls are proposed. Kalyanpur and Parsia et al. integrated a number of simple debugging cues generated from their description logic reasoner, Pellet, in their ontology development environment, Swoop. They aimed to debug unsatisfiable classes and repair them in OWL ontologies [17–19]. Schlobach proposed a technique called pinpointing, and it could significantly improve the quality of semantic clarification, a process which in itself was useful for quality assurance of ontologies [20]. Wang et al. presented a "black boxed" heuristic approach based on identifying common errors and inferences to diagnose unsatisfiable classes in OWL ontologies [21]. In ontology mapping debugging, both heuristic approach and logic reasoning would be useful. Logic reasoning approach for mapping debugging needs distributed description logic (DDL) reasoner, and we will discuss it in our other work. This paper discusses how to utilize heuristic rules to solve ontology mapping debugging problem.

3 THE GENERAL IDEA

Creating original mapping result should not be the end of ontology mapping process. In this section, we will examine several mapping cases to demonstrate that error mappings, imprecise mappings and abnormal mappings could exist in the mapping result. We should compensate these bugs before providing mappings to users.

3.1 Mapping Cases Study

Similarly to the work in [8], we define an ontology mapping as a 4-tuple: $m_i = \langle e, e', s, r \rangle$, where i is a unique identifier of the given mapping element; e and e' are the entities of the first and the second ontology, respectively; s is a confidence measure in some mathematical structure (typically in the $[0, 1]$ range) holding for the correspondence between e and e'; r is the relation holding between e and e'. We just consider the equivalence ($=$) and generic/specific ($\sqsupseteq/\sqsubseteq$) relation (*is-a* relation) in this paper. If $s > \varepsilon$, where ε is a predefined threshold, we simplify a mapping by $\langle e\ r\ e' \rangle$, such as $O_1 : A \sqsupseteq O_2 : B$.

We assume that the ontologies discussed in this paper are presented in OWL, and e and e' are concepts, i.e. we just consider the mappings between concepts.

We use some mapping examples shown in Figures 1 and 2[2], which include part of mappings between two publication and university ontologies, respectively. For the sake of simplicity in the following discussions, we mark the left ontology as O_1, and the right one as O_2.

[2] In the process of generating Figures 1 and 2, we used an ontology visualization tool: RDF Gravity++ (`http://www.salzburgresearch.at`).

Mapping Case 1. From the mapping result set:

$$\{O_1 : Thesis = O_2 : Thesis, O_1 : DoctoralThesis \sqsubseteq O_2 : Thesis,$$

$$O_1 : MasterThesis \sqsubseteq O_2 : Thesis\},$$

we can obviously obtain that the last two mappings are redundant, because the $O_1 : Thesis = O_2 : Thesis$ can deduce the other two mappings. Similar examples are:

$$\{O_1 : Article = O_2 : Article, O_1 : BookArticle \sqsubseteq O_2 : Article\}$$

and

$$\{O_1 : TeachingAst = O_2 : TeachAssistant, O_1 : Assistant \sqsupseteq O_2 : TeachAssistant\},$$

where $O_1 : BookArticle \sqsubseteq O_2 : Article$ and $O_1 : Assistant \sqsupseteq O_2 : TeachAssistant$ are redundant mappings.

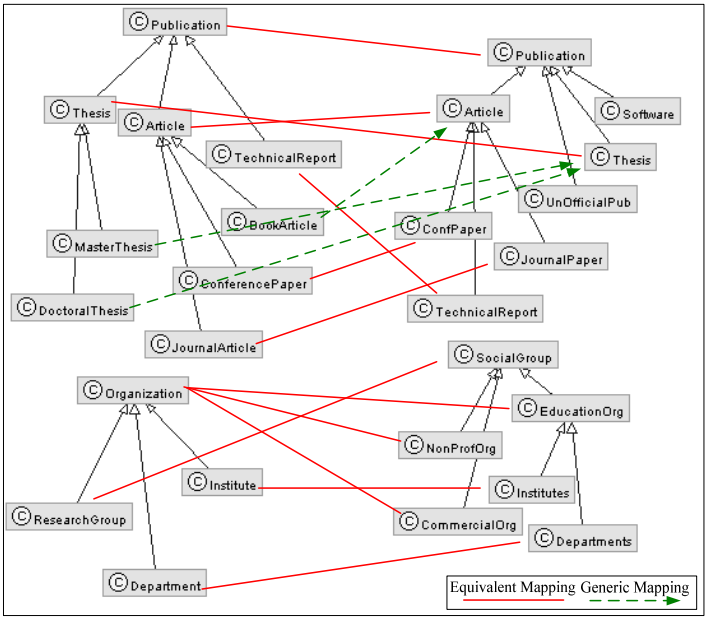

Fig. 1. Part of mappings between two publication ontologies

Mapping Case 2. Given the mapping result set:

$$\{O_1 : Organization = O_2 : CommericalOrg,$$

$$O_1 : Organization = O_2 : EducationOrg,$$

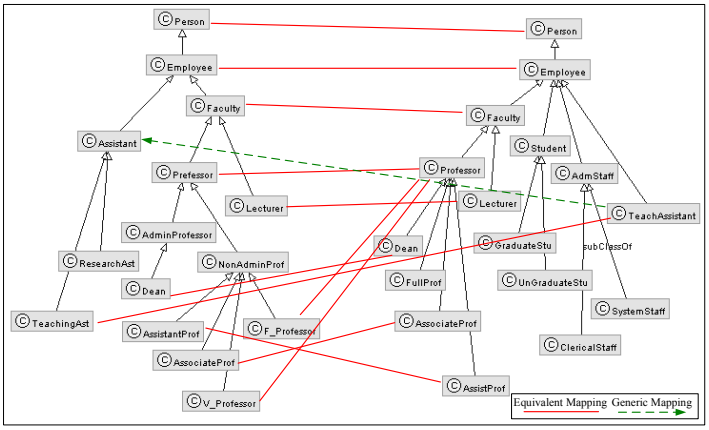

Fig. 2. Part of mappings between two university ontologies

$$O_1 : Organization = O_2 : NonProfOrg\},$$

we can obtain:

$$O_2 : CommericalOrg = O_2 : EducationOrg = O_2 : NonProfOrg.$$

Obviously, it is an unreasonable conclusion, especially when there is a *disjointWith* axiom declared between the three concepts. So the above mappings about $O_1 : Organization$ are imprecise. There are two potential solutions:

1. Importing a complex concept and representing the mappings as

   $$O_1 : Organization = O_2 : CommericalOrg \vee O_2 : EducationOrg \vee O_2 : NonProfOrg,$$

   and

2. Considering two sub-concepts of $O_1 : Organization$ are mapped to the sub-concepts of $O_2 : EducationOrg$, we also can simply treat the mapping of $O_1 : Organization$ as:

   $$O_1 : Organization = O_2 : EducationOrg.$$

   It needs users to determine which choice is better.

Mapping Case 3. Let's notice a mapping:

$$\{O_1 : ResearchGroup = O_2 : SocialGroup\}.$$

We can doubt it from two aspects. First, we observe the behavior of the mapping: the siblings of concept $O_1 : ResearchGroup$ are mapped to the children of $O_2 : EducationOrg$,

but $O_1 : ResearchGroup$ is mapped to the parent of $O_2 : EducationOrg$, so the behavior of the mapping is strange. Second, the mapping can also cause the inconsistency:

$$\left\{\begin{array}{ll} O_1 : ResearchGroup & = O_2 : SocialGroup \\ O_2 : SocialGroup & \sqsupseteq O_2 : EducationOrg \\ O_1 : Organization & = O_2 : EducationOrg \end{array}\right\} \Rightarrow \begin{array}{l} O_1 : ResearchGroup \\ \sqsupseteq O_1 : Organization. \end{array}$$

It apparently clashes with $O_1 : Organization \sqsupseteq O_1 : ResearchGroup$. Therefore, this mapping is error and should be removed.

Mapping Case 4. The two mappings:

$$\{O_1 : F\_Professor = O_2 : Professor\}$$

and

$$\{O_1 : V\_Professor = O_2 : Professor\}$$

are similar to Case 3, but we inspect them from other perspective. We know the mapping should not destroy the hierarchy structure (*is-a* structure) in ontology, but the two mappings will cause *is-a* circles as follows:

$$\text{Circle 1: } O_1 : F\_Professor \sqsubseteq O_1 : NonAdminProf \sqsubseteq O_1 : Professor$$
$$= O_2 : Professor = O_1 : F\_Professor$$
$$\text{Circle 2: } O_1 : V\_Professor \sqsubseteq O_1 : NonAdminProf \sqsubseteq O_1 : Professor$$
$$= O_2 : Professor = O_1 : V\_Professor$$

Here, the equivalent mappings are treated as bidirectional *is-a* relations. The two *is-a* circles destroy the hierarchy of ontology.

Maybe we have not listed all the problems in mapping results, but we think the cases we discussed have proved that there are some defects in original mapping results; and furthermore, most of them can be discovered and could be avoided; that is the goal of our work.

3.2 Bugs in Ontology Mapping Results

According to the mapping cases mentioned before, some common types of bugs may appear in the mapping result. We divide them into four categories.

1. Redundant Mappings. Some mappings are redundant if they can be deduced from the existing ones. Redundant mapping is one of the most common bugs. However, deleting all redundant mappings is not the best choice, because some redundant mappings can be useful in applications. For instance, storing redundant mappings before could be beneficial to the query efficiency and avoid the burden of reasoning them again. Another strategy is discovering all redundant mappings (including existing ones) and then storing them; whereas, in fact, the number of such redundant mappings may be large, not all of them will be useful.

2. **Imprecise Mappings.** Due to the limitation of the mapping algorithm, imprecise mappings would occur. Namely, the algorithm has not found the best answer but the approximate answer. We could revise these approximate mappings or combine them into more precise ones.
3. **Inconsistent mappings.** Some mappings could destroy the structure of original ontologies and disobey the declared axioms. We call such mappings *inconsistent mappings*. Both the mappings causing *is-a* circle and those disobeying the axioms (typically *equivalentClass* and *disjointWith* axioms) are inconsistent.
4. **Abnormal mappings.** Some suspicious mappings do not belong to the above three categories. It is difficult to find them in ordinary ways. However, such kind of mappings often shows the abnormal behavior. Namely, if two entities are close (such as siblings) in O_1, but they are mapped to O_2's two concepts, which are far away from each other. The errors caused by homonymy phenomenon, often have abnormal mapping behaviors.

Perhaps the four categories do not cover all the suspicious and erroneous mappings, but they certainly occur in mapping results frequently. Notice that the four categories are not mutually exclusive, they may intersect.

4 DEBUGGING ONTOLOGY MAPPINGS

In this section, we will define the types of mapping bugs, and then discuss the methods for detecting, repairing and diagnosing them.

4.1 Errors and Warnings in Mappings

Like programming debugging, we treat all suspicious mappings as two categories: *errors* and *warnings*. Apparently, errors are the confirmed wrong mappings, but warnings are the ones which may be wrong, right or imprecise.

Before defining the mapping defects, we give some basic symbols and definitions used throughout the paper.

Let T denote the concept set in ontology, T_1 and T_2 be the concept sets in O_1 and O_2, respectively. The set T_1^m (or T_2^m) denotes the concepts in O_1 (or O_2) used by mapping. Given a mapping $m_i = \langle e, e', s, r \rangle$, let $Lft(m_i) = e \in T_1$, $Rit(m_i) = e' \in T_2$, $Rel(m_i) = r$, $M = \{m_i | 0 \leqslant i \leqslant N_m\}$ denote the mapping set, where N_m is the number of mappings.

We first define *mapping implication*, and then use it to explain redundant mappings.

Definition 1 (Mapping Implication)**.** If a mapping m_1 can deduce another mapping m_2, we call m_1 implies m_2, and denote as $m_1 \sqsupseteq m_2$. If $m_1 \sqsupseteq m_2$ and $m_2 \sqsupseteq m_1$, they are equivalent, i.e. $m_1 \equiv m_2$.

Then we obtain the redundant mappings by the following theorem.

Theorem 1 (*Redundant Mappings*)**.** Given a mapping m_i, if $\exists m_j \in M$ and $m_j \sqsupseteq m_i$, m_i must be redundant.

Theorem 1 holds apparently, so we omit the proof.

There are two types of redundant mappings. One type is caused by *is-a* mapping and equivalent mapping as *Mapping Case* 1 shows. The other type is caused by *equivalentClass* axiom, which states two concepts are equivalent. If both equivalent concepts are mapped to a same concept, a redundant mapping occurs.

In debugging, we will not delete the redundant mappings, so we treat them as warnings and list them to the users.

There is no error in some mappings, but they are not the best answers we expect. Determining whether a mapping is the best answer is not easy. However, we can obtain more precise mapping by reusing existing mappings. In fact, when a concept C is mapped to multiple concepts $\{D_i\}$ in another ontology, we can combine the multiple concepts into a complex concept by conjunction or disjunction operations, such as $C{=}\bigvee D_i$ or $C \sqsubseteq \bigvee D_i$. Stuckenschmidt has used this idea for approximate information filtering [22].

Definition 2 (Imprecise Mappings)**.** Given a concept C, C is mapped to multiple concepts $\{D_i\}(2 \leqslant i \leqslant k)$ by relation r. if $\forall l, m\,(2 \leqslant l, m \leqslant k$ and $l \neq m)$, $D_l \sqsubseteq D_m$ does not exist, then we obtain the following rules: (1) if r is $\sqsupseteq$, more precise mapping is $C \sqsupseteq \bigvee D_i$; (2) if r is $\sqsubseteq$, more precise mapping is $C \sqsubseteq \bigwedge D_i$; (3) if r is $=$, we need more knowledge (such as the mappings of C's child and parent) to decide which D_i is the precise mapping, or we can simply use $C \sqsupseteq \bigvee D_i$.

The imprecise mappings are the warnings. When we obtain more precise mappings, the imprecise ones should be replaced.

The inconsistent mappings may destroy the ontology structure or violate axioms. For the sake of convenience, we use graph to represent the ontology with mappings.

Definition 3 (Ontology Graph with Mappings)**.** Let $G = (V, E)$ denote an ontology graph, where V is the set of concepts and E is the set of *is-a* edges. If $(v_1, v_2) \in E$, v_1 is the direct child of v_2. Two ontology graphs G_1 and G_2 can be connected by the mappings between them: the equivalent mappings are treated as bidirectional *is-a* edges, and the *is-a* mappings are transformed into *is-a* edges.

We assume that the original ontology is sound and does not include circles. When two ontology graphs are connected by mappings, that should not cause circles in such graph. In this way, we can identify the inconsistent mappings easily.

Definition 4 (Inconsistent Mappings Causing is-a Circles)**.** Combining two ontology graphs G_1 and G_2 with the mappings M into a single graph G. $p = \{v_1, v_2, \ldots, v_m\}$ is a path in G, and if $v_1 = v_m$, we call p a circle. For all paths p_x in G, if p_x is a circle, p_x must contain inconsistent mappings.

Notice that not all mappings in path p_x are inconsistent.

In ontology debugging, the inconsistent mappings causing *is-a* circles are errors.

The other inconsistent mappings may violate the axioms in original ontology. We just consider *equivalentClass* and *disjointWith* axioms in this paper.

Definition 5 (Inconsistent Mappings Violating Axioms)**.** Given concepts $\{A_i\}$, which satisfy *equivalentClass* axiom, mappings $\{m_k\}$ mapped $\{A_i\}$ to concepts $\{B_j\}$ in another ontology, if $\{B_j\}$ fail to satisfy *equivalentClass* axiom, $\{m_k\}$ must have inconsistent mappings. Similarly, if $\{A_i\}$ satisfy *disjointWith* axiom, but $\{B_j\}$ fail to satisfy the same axiom, mappings $\{m_k\}$ have inconsistent mappings.

In debugging, the inconsistent mappings violating axioms are errors.

The last kind of bug is abnormal mapping. In order to describe the abnormal mapping, we define mapping behavior first.

Intuitively, when close concepts in O_1 are mapped to O_2, the counterpart in O_2 would likely be close too. We use this intuition to define the mapping behavior.

Definition 6 (Mapping Behavior)**.** Given two mappings $m_i = \langle c_i, b_i, s, r_i\rangle$ and $m_j = \langle c_j, b_j, s, r_j\rangle$, let $r_i = r_j$, the mapping behavior of m_i on c_i relative to m_j on c_j is

$$Bh(m_i|(m_j, c_j)) = \frac{d(c_i, c_j)}{d(b_i, b_j)},$$

where $d(c_i, c_j)$ and $d(b_i, b_j)$ denote the distance between concepts.

Apparently, the mapping behavior is a relative value. We also can compute the mapping behavior of m_i on c_i relative to c_i's neighbors:

$$Bh(m_i|(m_1, c_1), \ldots, (m_k, c_k)) = \frac{1}{k}\sum_{t=1}^{k}\frac{d(c_i, c_t)}{d(b_i, b_t)}, (t \neq i).$$

The closer to 1 the mapping behavior value is, the more normal the mapping behavior is.

Definition 7 (Abnormal mappings)**.** If the mapping behavior of m_i relative to its neighbors satisfies

$$|Bh(m_i|(m_1, c_1), \ldots, (m_k, c_k)) - 1| > \theta,$$

m_i is a abnormal mapping. θ is a predefined threshold.

In debugging, we treat abnormal mappings as warnings as well, because we need to judge whether an abnormal mapping is an error mapping.

4.2 Detecting and Diagnosing Mappings

During ontology mapping debugging, the most important process is detecting and diagnosing the suspicious mappings. We present the heuristic approach debugging algorithms. The algorithms should lock the suspicious mappings and output useful suggestions.

For the redundant mappings, we first deal with the redundancies caused by *equivalentClass* axiom, and then check other redundancies caused by mapping implication. Algorithm 1 shows the process for debugging redundant mappings.

Algorithm 1 Debugging redundant mappings

1: **for all** concept pair $\{A_i, A_j\}$ involving *equivalentClass* axiom **do**
2: **if** $\exists m_i, m_j \, AND \, Rel(m_i) == Rel(m_j)$ **then**
3: **if** $(Lft(m_i) == A_i$ AND $Lft(m_j) == A_j$ AND $Rit(m_i) == Rit(m_j))$ OR $(Rit(m_i) == A_i$ AND $Rit(m_j) == A_j$ AND $Lft(m_i) == Lft(m_j))$ **then**
4: Output: [Warning] m_i or m_j is redundant
5: **end if**
6: **end if**
7: **end for**
8: **for all** mapping m_i **do**
9: **for all** mapping $m_j (m_i \neq m_j)$ **do**
10: **if** $(m_i \sqsupseteq m_j)$ **then**
11: Output: [Warning]m_j is redundant
12: **else if** $m_i \sqsubseteq m_j$ **then**
13: Output: [Warning]m_i is redundant
14: **end if**
15: **end for**
16: **end for**

Using mapping implication, the redundant mappings can be re-organized into the structure like hierarchy.

The inconsistent mappings caused by *is-a* circles can be found according to Definition 4, and detecting other inconsistent mappings needs to check the axioms (see Algorithm 2).

Algorithm 2 Debugging inconsistent mappings

1: Combine ontology O_1, O_2 and mappings M into a graph G
2: Traverse G to find all circles P
3: **for all** circle p_i **do**
4: Output:[Error]Inconsistent mappings in p_i
5: **end for**
6: **for all** concepts A_i satisfy *equivalentClass/disjointWith* axiom **do**
7: **if** m_k map A_i to B_j AND B_j fail to satisfy *equivalentClass/disjointWith* axiom **then**
8: Output:[Error] Inconsistent mappings in m_k
9: **end if**
10: **end for**

The users can repair the inconsistent mappings according to debugging information until no other inconsistent mappings appear.

According to Definition 2, we design an algorithm to debug imprecise mappings. For imprecise mappings are often ambiguous, we debug them with some heuristic rules.

Algorithm 3 Debugging imprecise mappings

for all concept $C_i \in T_1 \cup T_2$ **do**
  m_k map C_i to D_j,$(2 \leqslant |m_k|, |D_j| \leqslant n)$
  if $\forall s,t, D_s \sqsubseteq D_t$ does not exist **then**
    Select Case (the mapping relation of m_k)
    Case $\sqsupseteq$:
    Output:[Warning] m_k may be imprecise
    Output:More precise mapping may be: $C_i \sqsupseteq \bigvee D_j$
    Case $\sqsubseteq$:
    Output:[Warning] m_k may be imprecise
    Output:More precise mapping may be: $C_i \sqsupseteq \bigwedge D_j$
    Case $=$:
    Output:[Warning] m_k may be imprecise
    Suggestion 1: More precise mapping may be: $C_i \sqsupseteq \bigvee D_j$
    Suggestion 2: Check the mappings of C_i's children/parents for final decision;
    End Select
  end if
end for

For the abnormal mappings, we calculate their behavior value. For a given concept and related mappings, we consider the relative mapping behavior to the concept's neighbors. The algorithm is as follows:

Algorithm 4 Debugging abnormal mappings

for all mapping m_i **do**
  $C_i = Lft(m_i)$ or $C_i = Rit(m_i)$
  D_i denotes C_i's neighbors within distance δ
  $\{E_i\}\subseteq\{D_i\}$, $|\{E_i\}|=k$, and $\forall E_i$, $\exists m_t$ and $Rel(m_t) = Rel(m_i)$
  Calculate m_i's behavior as: $Bh(m_i|\{E_i\}) = \frac{1}{k}\sum_{t=1}^{k}\frac{d(c_i,ct)}{d(e_i,e_t)}$
  if $|Bh(m_i|\{E_i\}) - 1| > \theta$ **then**
    Output:[Warning]m_i is abnormal mapping
  end if
end for

5 EXPERIMENTS AND DISCUSSIONS

5.1 Experiment Results

We report some results we have obtained. In our experiment, we use five-pair ontologies, which describe university, food, publication, travel and film domains, respectively. All of the ontologies are collected through the semantic search engine Swoogle[3]. Before using

[3] `http://swoogle.umbc.edu/`

the five-pair ontologies, we preprocess them with several steps, which include translating DAML format to OWL format, removing some complex classes and revising some concepts and properties. In order to create the original mapping result, we utilize several simple mapping techniques:

1. **Literal similarity** computes Levenshtein distance between concepts information (including local-names, labels and comments) to estimate the similarity;
2. **Structure Similarity** examines concepts' neighbors, siblings, domain and range to measure the similarity.

The ontology mapping debugging algorithms are implemented using Java JDK1.5 and Jena-2.4[4] API. The input data are a pair of ontologies and the mappings between them, and the output data is debugging information. According to output information in debugging, the user can accept or reject the suggestions generated by debugging algorithm. When a suggestion is accepted, the user can examine the warnings/errors, and then repair them manually. Notice that the mapping debugging is an iterative process, which can avoid that new change on the mappings would cause new warnings/errors.

	Redundant Mappings		Inconsistent Mappings				Imprecise Mappings		Abnormal Mappings	
			Causing Circles		Violating Axiom					
	Num	P/R	Num	P/R	Num	P/R	Num	P/R	Num	P/R
University	12	1.00/ 1.00	7	1.00/ 1.00	7	1.00/ 1.00	5	1.00/ 0.83	5	0.80/ 1.00
Food	23	1.00/ 0.92	12	1.00/ 1.00	3	1.00/ 1.00	9	0.77/ 1.00	4	0.50/ 1.00
Publication	30	0.93/ 0.88	6	1.00/ 1.00	0		5	1.00/ 1.00	7	0.71/ 1.00
Travel	10	0.90/ 0.82	3	1.00/ 1.00	3	1.00/ 1.00	2	1.00/ 0.67	0	
Film	5	1.00/ 1.00	0		2	1.00/ 1.00	6	0.83/ 0.63	2	1.00/ 0.50

Table 1. Ontology mapping debugging results

We also use *precision* and *recall* to evaluate the performance of our debugging technique. If a suggestion is accepted by the user, or the user agrees that an output error/warning is useful, then the suggestion/error/warning is right. Table 1 shows the debugging results in our experiment.

After debugging, some mapping errors may be repaired. We re-calculate the precision and recall of the mapping result, and compare with the original one (Figure 3). Figure 3 demonstrates that the debugging process is useful, and the quality of mapping result is improved significantly.

[4] `http://jena.sourceforge.net/index.html`

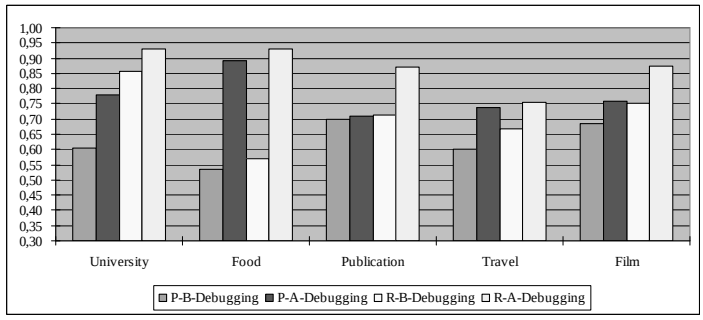

Fig. 3. Mapping results comparison (before debugging and after debugging)

5.2 Discussions

In fact, the debugging technique we discussed is just a static process. If we can import some dynamic techniques, such as setting breakpoints in the debugging process, we could control the mapping process and choose the best parameters.

The abnormal behavior of mappings is based on the intuition. In debugging process, the abnormal mappings algorithm is very useful for dealing with homonymy concepts. However, when two ontologies have divarication for the same concepts, the abnormal mappings debugging could return wrong results.

The mapping bugs defined in this paper probably do not cover all kinds of mapping defects. Moreover, some intensive experiments are needed to verify this technique.

We have not used reason methods in the debugging process. We believe the distributed description logic (DDL) reason will be of benefit to the ontology mapping debugging.

Fortunately, to the best of our knowledge, some companies are adopting(or will adopt) the mapping debugging idea in their semantic information integration product. The mapping debugging could be a useful tool/component for the ontology mapping systems.

6 CONCLUSIONS

A novel technique called ontology mapping debugging technique is proposed. Four types of mapping bugs are defined, and the methods for detecting and diagnosing them are proposed as well. The experiment results demonstrate that the technique is promising and can improve the mapping result quality.

Acknowledgements

This work was supported in part by the NSFC(60425206, 90412003), and Excellent Ph. D. Thesis Fund of Southeast University (YBJJ0502). The authors thank Yimin Wang and Jie Bao for the early discussions on this paper. Jin Zhou read and polished the entire paper.

The authors would also like to thank the anonymous reviewers for their helpful comments and suggestions for improving the manuscript.

REFERENCES

[1] Berners-Lee, T.—Hendler, J.—Lassila, O.: The Semantic Web. Scientific American, Vol. 284, 2001, No. 5, pp. 34–43.

[2] Natalya, F. N.—Mark, A.: The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. Int. J. Hum.-Comput. Stud, Vol. 59, 2003, No. 6, pp. 983–1024.

[3] AnHai, D.—Jayant, M.—Robin, D. et al.: Learning to Match Ontologies on the Semantic Web. The VLDB Journal. Vol. 12, 2003, No. 4, pp. 303–319.

[4] Ehrig, M.—Staab, S.: QOM – Quick Ontology Mapping. Proceedings of the Third International Semantic Web Conference, ISWC2004, Hiroshima, Japan, 2004.

[5] Euzenat, J.—Stuckenschmidt, H.—Yatskevich, M.: Introduction to the Ontology Alignment Evaluation 2005. Proceedings of the K-CAP 2005 Workshop on Integrating Ontologies, Banff, Canada, 2005.

[6] Erhard, R.—Philip, A. B.: A Survey of Approaches to Automatic Schema Matching. The VLDB Journal, Vol. 10, 2001, No. 4, pp. 334–350.

[7] Kalfoglou, Y.—Schorlemmer, M.: Ontology Mapping: The State of the Art. The Knowledge Engineering Review, Vol. 18, 2003, pp. 1–31.

[8] Shvaiko, P.—Euzenat, J.: A Survey of Schema-Based Matching Approaches. Journal on Data Semantics, Vol. 4, 2005, pp. 146–171.

[9] Hess, A.: An Iterative Algorithm for Ontology Mapping Capable of Using Training Data. Proceedings of the 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, 2006.

[10] Ehrig, M.—Euzenat, J.: Relaxed Precision and Recall for Ontology Matching. Proceedings of the K-CAP 2005 Workshop on Integrating Ontologies, Banff, Canada, 2005.

[11] Euzenat, J.: Semantic Precision and Recall for Ontology Alignment Evaluation. Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI2007, Hyderabad, India, January 6–12, 2007.

[12] Hanif, S.—Seki, Y.—Aono, M.: Automatic Alignment of Ontology Eliminating the Probable Misalignments. Proceedings of the 1st Asian Semantic Web Conference, ASWC 2006, Beijing, China, 2006.

[13] Distributed Reasoning Architecture for Galaxy of Ontologies Mapping Debugging. `http://sra.itc.it/projects/drago/applications-debugging.html`.

[14] Stuckenschmidt, H.—Serafini, L.—Wache, H.: Reasoning about Ontology Mappings. In ECAI 2006 Workshop on Context Representation and Reasoning, Ria del Garda, Italy, 2006.

[15] Chiticariu, L.—Tan, W.: Debugging Schema Mappings with Routes. Proceedings of International Conference on Very Large Data Bases, VLDB 2006, Seoul, Korea, 2006.

[16] Alexe, B.—Chiticariu, L.—Tan, W.: SPIDER: A Schema mapPIng DEbuggeR. Proceedings of International Conference on Very Large Data Bases, VLDB2006, Seoul, Korea, 2006.

[17] Kalyanpur, A.—Parsia, B.—Sirin, E. et al.: Debugging Unsatisfiable Classes in OWL Ontologies. Journal of Web Semantics, Vol. 3, 2005, No. 4, pp. 268–293.

[18] Parsia, B.—Sirin, E.—Kalyanpur, A.: Debugging OWL Ontologies. Proceedings of the 14th Inter-national World Wide Web Conference, WWW 2005, Chiba, Japan, 2005.

[19] Kalyanpur, A.—Parsia, B.—Sirin, E. et al.: Repairing Unsatisfiable Concepts in OWL Ontologies. Proceedings of the 3rd European Semantic Web Conference, ESWC2006, Budva, Montenegro, 2006.

[20] Schlobach, S.: Debugging and Semantic Clarification by Pinpointing. Proceedings of the Second European Semantic Web Conference, ESWC 2005, Heraklion, Greece, 2005.

[21] Wang, H.—Horridge, M.—Rector, A.: Debugging OWL-DL Ontologies: A Heuristic Approach. Proceedings of the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, 2005.

[22] Stuckenschmidt, H.: Approximate Information Filtering with Multiple Classification Hierarchies. International Journal of Computational Intelligence and Applications, Vol. 2, 2002, No. 3, pp. 295–302.

Peng Wang received his bachelor and master degrees from Northwestern Polytechnical University, P. R. China in 2000 and 2003, respectively. Now, he is a Ph. D. candidate in the School of Computer Science and Engineering, Southeast University. His current research interests include Semantic Web, ontology, and information retrieval on the web.

Baowen Xu is a Professor of School of Computer Science and Engineering, Southeast University. His research areas include programming languages, software engineering (software analysis and testing, re-engineering), formal software techniques, web information analysis and testing techniques, knowledge and information retrieval techniques. He has published more than 20 books and more than 200 papers in scientific journals and international conferences/workshops in the above research areas. He has been the general chair, program chair or PC member of more than 20 international conferences.