Ontology Reconciliation for Service-Oriented Computing

Jingshan Huang, Jiangbo Dang, and Michael N. Huhns
Computer Science & Engineering Dept., University of South Carolina, Columbia, SC 29208, USA
{huang27, dangj, huhns}@engr.sc.edu

Abstract

Service-oriented computing (SOC) is viewed as the computing paradigm of the near future, allowing for the dynamic interaction of services provided by distributed business partners. Being a declarative knowledge representation model, ontologies serve as a foundation for SOC. Due to the heterogeneous nature of independently designed ontologies, it is problematic for partners to understand the concepts adopted in ontologies from other sources. In order for partners to achieve seamless collaboration of services, they need to reconcile their ontologies with each other. During the alignment process and the following service interactions, compatibility is an important measurement that has been neglected in most research work. We extend a vector system to encode ontology compatibility. In addition, we present a new model – probabilistic center ontology – for better recording and maintenance of ontology alignment results. Our precise and efficient approach is verified by both theoretic proofs and experimental results.

1. Introduction

Service-oriented computing (SOC) is the emerging cross-disciplinary paradigm for distributed computing: it is changing the way business applications are designed, architected, delivered, and consumed so as to better support interoperability and dynamism in meeting changing business needs. Initially, business automation required that partners predefine the terminology of their interaction using EDI and XML standards, such as ebXML. In this sense, the automation activities were tightly coupled, which is a major disadvantage if we would like to enable sharing tasks and automating processes. In contrast, services are autonomous and platform-independent computational elements that can be described, published, discovered, orchestrated, and deployed using standard protocols (UDDI for discovery, WSDL for description, BPEL4WS for coordination, and SOAP for communication). Through the methodology proposed by SOC, we are able to build networks of collaborating applications distributed within and across organizational boundaries. By providing the automated support needed for e-business collaboration and integration both at the data and business logic levels, the visionary promise of SOC is a world where application components are assembled with little effort into a network of loosely coupled services spanning organizations and computing platforms.

Ontologies serve as a declarative knowledge representation model and form a foundation for SOC. However, because it is impractical to have an agreed-upon, unique, and global ontology that includes every concept that is or might be adopted as part of the services, distributed partners typically have heterogeneous semantics in services rendered. Due to this basic characteristic, partners need to align their ontologies and form a mutual understanding among each other automatically. Only in this sense are partners able to integrate the information from different sources autonomously, and then facilitate service-based application interoperability.

During the ontology alignment process and the following service interactions, quality is an important measurement in selecting partners with which to communicate, especially in cases where resources are limited. In addition, compatibility is an important component of quality. Communicating partners have ontologies of different compatibility, in that those partners with high compatibility ontologies are more likely to understand and be understood by other partners, and this kind of mutual understanding is the prerequisite for interoperation. Notice that the quality of the service provided by partners is a separate research topic not covered in this paper. Based on the above insight, we extend a vector system in [12] to support ontology compatibility encoding. In addition, we introduce a new model – a probabilistic center ontology – for more suitable recording and maintenance of ontology matching results.

This paper advances the state of the art by (1) introducing ontology quality issues into QoS, (2) exploring the use of a compatibility system to speed up the discovery of the partner(s) of interest, and (3) proposing a probabilistic ontology model for aligning ontologies. In the rest of our paper, Section 2 introduces related work, Section 3 discusses ontology heterogeneities in SOC and outlines our solution, Section 4 presents a schema-based ontology merging algorithm, Section 5 introduces the probabilistic center ontology and an extended vector system, Section 6 reports on the experimental results and Section 7 concludes with future work.

2. Related work

2.1. Related work in ontology matching

The need for the automatic or semi-automatic mapping, matching, and merging of ontologies from different sources has prompted considerable research. All of the following systems take a schema-based approach, except for GLUE [2], which is an instance-based system.

0-7695-2670-5/06 $20.00 © 2006 IEEE

PROMPT [5] is a tool that makes use of linguistic similarity matches between concepts for initiating the merging or alignment process, and then uses the underlying ontological structures of the Protégé-2000 environment to inform a set of heuristics for identifying further matches between the ontologies. PROMPT has a good performance in terms of precision and recall. However, user intervention is required, which is not always available in real world applications.

Similarity Flooding [7] utilizes a hybrid matching technique based on the idea that similarity spreading from similar nodes to the adjacent neighbors. Before a fix-point is reached, alignments between nodes are refined iteratively. This algorithm only considers the simple linguistic similarity between node names, leaving behind the node property and inter-node relationship.

Cupid [4] combines linguistic and structural schema matching techniques, as well as the help of a precompiled dictionary. But it can only work with a tree-structured ontology instead of a more general graph-structured one. As a result, there are many limitations to its application.

COMA [3] provides an extensible library of matching algorithms, a framework for combining results, and an evaluation platform. According to their evaluation, COMA is performing well in terms of precision, recall, and overall measures.

The work in [6] investigates a probabilistic framework for ontology mapping. Ontologies are first translated into Bayesian networks, and then the concept mapping is realized as evidential reasoning. The probabilities needed in both translation and mapping can be obtained by using text classification programs.

S-Match [8] is a modular system into which individual components can be plugged and unplugged. The core of the system is the computation of relations. Five possible relations are defined between nodes: equivalence, more general, less general, mismatch, and overlapping. Like Cupid, S-Match uses a tree-structured ontology.

GLUE introduces well-founded notions of semantic similarity, applies multiple machine learning strategies, and can find not only one-to-one mappings, but also complex mappings. However, it depends heavily on the availability of instance data. Therefore, it is not practical for cases where there is not a significant number of instances or no instance at all.

2.2. Related work in QoS

Quality of service (QoS) is becoming a significant factor with the widespread deployment of Web services. By QoS, we refer to the non-functional properties of services, such as reliability, availability, and security. [9] proposes a Service Query and Manipulation Language (SWSQL) to maintain QoS attribute ontologies and to publish, rate, and select services by their functionality as well as QoS properties. Based on SWSQL, a UDDI registry is extended to a service repository by combing a relational database and the attribute ontology.

Zhou et al. [10] provide a DAML-QoS ontology as a complement to a DAML-S ontology in which multiple QoS profiles can be attached to one service profile. In addition, a matchmaking algorithm for QoS properties is presented.

One widely used QoS attribute is user rating, but it is subjective to the perception of the end user and is limited by the lack of an objective representation of the performance history. Kalepu et al. [11] introduce reputation, a composition of user rating, compliance, and verity as a more viable QoS attribute. Ontologies are applied to QoS-aware service selection, execution, and composition. A selected ontology itself can adopt some QoS measures to facilitate mutual ontology understanding, as discussed in this paper.

3. Ontological heterogeneity and our solution

3.1. Ontological heterogeneity in SOC

In order to collaborate with the services rendered by other partners, a business partner must first be able to comprehend the descriptions about those services. Being a formal knowledge representation model, ontologies can aid in this comprehension by providing the necessary semantics during collaboration.

An example scenario of the interaction within a SOC environment is envisioned as follows.

1. A number of business partners form a SOC community (SOCC) within which services provided by different partners might be integrated into a single equivalent service that is more complete and functional. This integration requires the mutual understanding of the individual ontology underlying each partner.
2. The partners outside this SOCC can request help from the community and make use of its services, either the original ones or the integrated one. This request requires not only an understanding of the related ontologies, but also the ability to choose suitable service provider(s), especially under the situations where resources are limited.

Because there is no global, common, and agreed-upon ontology, any partner can base its service on an ontology that reflects its own conceptual view of the world. Consequently, ontological heterogeneity among different partners becomes an inherent characteristic of a SOCC. The heterogeneity can occur in two ways: (1) different ontologies could use different terminologies to describe the same conceptual model, and (2) even if two ontologies use the same name for a concept, its corresponding properties and relationships with other concepts can be different. Therefore, two major problems must be confronted: (1) during the formation of a SOCC, how can it be ensured that all partners within the community have no problem in understanding each other's ontology? And (2) how can an external partner seeking collaboration with a SOCC select those partners that understand its ontology best?

3.2. Overview of our solution

In order to solve the first problem – mutual understanding of ontologies within a SOCC – we need an

approach to reconcile ontologies from different partners through an alignment process. By this means, concepts from communicating partners can be related, and possible integration of related services can be achieved. Our main idea is to form a *center* ontology by merging all original ones during the generation of a SOCC. The center ontology serves as a reference through which the original ontologies are able to align with each other.

To tackle the second problem – the correct selection of partners that are most closely aligned with the ontology of an external partner – we introduce compatibility vectors as a means of measuring and maintaining ontology compatibility. By setting up the compatibility for each partner during the formation of a SOCC, not only a better mutual understanding of ontologies within the SOCC is obtained, but also the partners outside this community are able to select the best partner(s) with ease.

4. Schema-based ontology merging algorithm

Our goal is to develop a methodology for constructing a merged ontology from two original ones. The methodology can then be applied iteratively to merge all ontologies within a SOCC. Our methodology, based on the ontology merging algorithm presented in [1], is summarized next.

4.1. Top-level procedure

Ontology merging is carried out at the schema level. Internally we represent an ontology using a directed acyclic graph *G (V, E)*, where *V* is a set of ontology concepts (nodes) and *E* is a set of edges between concepts, i.e., $E = \{(u, v) \mid u$ and v belong to V and u is a superclass of $v\}$. In addition, we assume that all ontologies share *Thing* as a common root. To merge two ontologies, G_1 and G_2, we relocate each concept from one ontology into the other. We adopt a breadth-first order to traverse G_1 and pick up a concept *C* as the target to be relocated into G_2. Consequently, at least one member of *C*'s parent set Parent(*C*) in the original graph G_1 has already been placed into its proper location in the destination graph G_2 before the relocation of *C* itself. The pseudocode below describes this procedure, whose time complexity is $O(n^2)$, with *n* the number of concepts in the merged ontology.

```
Input: Ontology G1 and G2
Output: Merged Ontology G2
begin
  new location of G1's root = G2's root
  for each node C (except for the root)in G1
  Parent(C) = C's parent set in G1
  for each member pi in Parent(C)
          pj = new location of pi in G2
          relocate(C, pj)
  end for
  end for
end
```

Top-level procedure for ontology merging

4.2. Relocate function

The *relocate* function in the top-level procedure is used to relocate *C* into a subgraph rooted by p_j. The main idea is: try to find one of three relationships (*equivalentclass*, *superclass,* or *subclass*) between *C* and p_j's direct child(ren). If we cannot find any, the only option is for us to let *C* be another direct child of p_j.

5. Center ontology and compatibility vectors

All original ontologies are merged into a center ontology, which has built-in probabilities recording the similarity degrees for concepts. We use compatibility vectors to represent the compatibility of the constituent ontologies. Compatibility vectors are stored in the center, encoding a measure of distance from an original ontology to the center. The distances can be adjusted efficiently during and after a SOCC is formed. Based on the information contained in the vectors, partners can straightforwardly understand ontologies from each other. In addition, the partners from outside this community will be able to choose the partner(s) with more compatible ontologies. We adopt compatibility instead of reputation when choosing suitable services, because the former is based on an objective calculation, thus avoiding the drawbacks of subjective reputation.

5.1. Probabilistic center and ontology distance

5.1.1. Center formation and its role in ontology matching. The center ontology is generated by merging all original ontologies, step by step, as each new partner joins a SOCC. Initially, when there is only one partner, its ontology is regarded as the center ontology. Each time a new partner joins the community, the new ontology is merged with the current center to obtain the new center ontology. If there is no center ontology, we need $\frac{n(n-1)}{2}$ pairwise alignments among *n* ontologies. In addition, whenever an original ontology changes, new alignments are needed between this modified ontology and all remaining ones. After the construction of a center ontology, an alignment will be built between each original ontology and the center, so only *n* alignments are required. For any pair of original ontologies, the related alignment against the center is able to provide enough information for these two ontologies to align with each other. Moreover, in cases when modifications are made in original ontologies, only the comparison between the center and the modified ontology is necessary.

5.1.2. Probabilistic center. We introduce the idea of a probabilistic center ontology, i.e., a concept *C* in one original ontology (e.g., ontology_1) is equivalent to a concept *C'* in the center by some probability *p*. Therefore, *C* is a child of the parent of *C'* by *p*. This creates a center ontology with probabilities in parent-child hierarchy.

Initially no ambiguity would result, because ontologies are often created by definition. For example, there is no doubt that a *lion* is a *mammal*, because biologists have defined it to be that way. However, ambiguity can arise in two ways.

(1) *Object Classification* – a given object (instance) might or might not be a member of a class (concept). For

example, the animal one sees walking through the bushes might or might not be a *lion*.

(2) *Ontology Merging* – a concept *C* in ontology_1 *might* be equivalent to a concept *C'* in the center. Even an ontology expert cannot always be completely sure about such an equivalence, let alone an automated procedure. Therefore, we need a measure of equivalence; and probability can record a degree of similarity.

The ambiguity from object classification belongs to the problem domain of instances, and will not be considered in our solution, which deals with ontology schemas alone.

For the ambiguity resulting from ontology merging, there are two possible solutions. The first solution is an instance-based one. The similarity degree between two concepts can be measured by the instances that are members of both concepts versus those instances that are members of just one concept. For example, the concepts *BrownFurryAnimals* and *DangerousAnimals* would have all of the *lions* in common, but a *brown furry dog* is not dangerous and a *rattlesnake* is dangerous but not brown and furry. The second solution is a schema-based one, which is essentially the same as the instance-based one, just from a different viewpoint. For example, the facts that "a *brown furry dog* is not dangerous" and "a *rattlesnake* is not brown and furry" result from the different properties in the concepts *BrownFurryAnimals* and *DangerousAnimals*. Therefore, instead of counting the instances, we can represent this difference by properties at a higher level – the schema level. Basically, if we could enumerate exhaustively all associated instances, the percentage of common properties over the union of properties should be represented by the ratio of corresponding instances. For example, suppose in the center ontology there is a concept *B_D_Animals* which includes all and only the properties from both *BrownFurryAnimals* in ontology_1 and *DangerousAnimals* in ontology_2. Furthermore, suppose that these two original concepts contribute 70% and 75% respectively for the properties of *B_D_Animals*; then we are 70% and 75% sure, respectively, that "*BrownFurryAnimals* is equivalent to *B_D_Animals*" and "*DangerousAnimals* is equivalent to *B_D_Animals*". In this sense, the merged center ontology is a probabilistic one with probabilistic alignment for each original ontology.

5.1.3. Concept distance. The center ontology contains information from all original ontologies, because the former is the result of the merging of the latter. Therefore, with respect to whether a specific original ontology understands each concept in the center ontology or not, there are two situations. One situation is that for one specific concept in the center, the original ontology can understand it, but possibly with less accurate information. The other situation is that the original ontology is not able to recognize that concept at all. In either case, the concept distance is represented by the amount of information missing, i.e., the number of relationships not known in the original ontology. The following equation formalizes the concept distance, $d_{concept}$:

$d_{\text{concept}} = w_1 * n_{sub\text{-}super} + w_2 * n_{other}$, with $(w_1 + w_2 = 1)$.

$n_{sub\text{-}super}$ is the number of sub/superclass (isa) relationships not known in the original ontology, and n_{other} is the number of other relationships not known in the original ontology. w_i is the weight given to different relationship types, including *subclass*, *superclass*, *equivalentclass, disjointWith*, *parts*, *owns*, *contains*, and *causes*. Because the sub/superclass relationship is the most important one in an ontology, w_1 will be given a greater value than w_2.

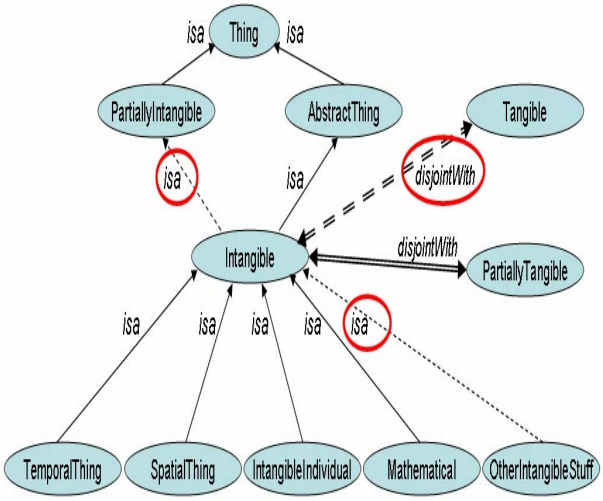

Figure 1. Merged center ontology

Consider the ontology in Figures 1. Suppose the center is merged from ontology_1 and others. The relationships from ontology_1 are represented by solid lines, while those from others are represented by dashed lines and circled. Thus, the concept distance from *Intangible* in ontology_1 to *Intangible* in the center is $(w_1 * 2 + w_2 * 1)$. From another viewpoint, concept distance can also be encoded as the *similarity degree* between concepts from the original and center ontology.

$similarity\ degree = w_1 * p_{sub\text{-}super} + w_2 * p_{other}$, with w_i having the same meaning as in equation for concept distance.

$p_{sub\text{-}super}$ is the percentage of sub/superclass relationships known in the original ontology over those in the center, and n_{other} is the percentage of other relationships.

5.1.4. Ontology distance. After each concept distance has been calculated as shown above, we can continue to figure out the ontology distance, d_{ontology}, between the original ontology and the center.

$d_{\text{ontology}} = \sum_{i=1}^{n} w_i * d_{\text{concepti}}$, where d_{concepti} is the distance between a pair of concepts, n is the number of concepts in the center, and w_i is explained next.

Recall that the concept set of the original ontology is a subset of that of the center, and the concept distance is encoded by the missing relationships in the original ontology compared to the center. The above equation shows that the ontology distance is obtained by the weighted sum of the concept distances between two ontologies. How much a concept contributes to the ontology distance is determined by the importance of that concept in its ontology. We use the percentage of the number of relationships to represent this measurement. For example, if ontology_1 has 100 relationships in total, and concept *Spatial* has 15 relationships, then the weight for this concept in ontology_1 is 0.15.

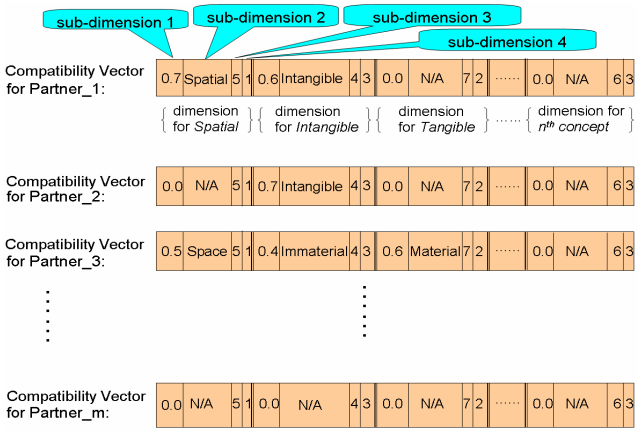

Figure 2. Compatibility vectors

5.2. Compatibility vectors

We extend the vectors presented in [12]. Inside the center, there is a set of compatibility vectors, one for each original ontology. A compatibility vector consists of a set of dimensions, each corresponding to one concept in the center. Therefore, all compatibility vectors have identical number of dimensions (the number of the concepts in the center). Each dimension has four sub-dimensions. The 1st sub-dimension encodes the similarity degree, associated with two numbers (the number of sub/superclass relationships and the number of other relationships in the original ontology); the 2nd sub-dimension records the concept name in the original ontology if the latter does recognize that concept; and the 3rd and 4th sub-dimensions keep track of the numbers of sub/superclass relationships and other relationships in the center, respectively. An example of compatibility vectors is shown in Figure 2.

For the first concept (*Spatial*) in the center, partner_1 knows it as *Spatial* and has a similarity degree of 0.7; partner_3 also understands this concept, but with a different name (*Space*) and a smaller similarity degree of 0.5; neither partner_2 nor partner_m recognizes concept *Spatial*, therefore, they have the same similarity degree (0.0).

5.3. Dynamic adjustment of ontology distance

5.3.1. During the formation of a SOCC. As mentioned above, when there is only one partner in a SOCC, its compatibility is perfect. In the compatibility vectors stored in the center, each similarity degree has a value of 1.0. However, with the addition of new partners into the SOCC, the compatibilities for existing partners might be changed because newly joined partners could contain more accurate information.

An example is shown in Figure 3, demonstrating the dynamic adjustment of the ontology distance (encoded in compatibility vectors) along with the formation of a SOCC.

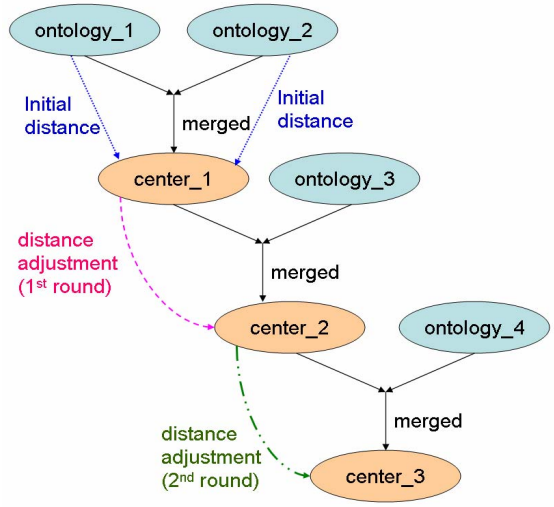

Figure 3. Dynamic adjustment of ontology distance

After ontology_1 and ontology_2 are merged to generate center_1, the compatibility vectors of these two ontologies in center_1, v_1 and v_2, are calculated. Upon the joining of ontology_3 and the generation of center_2, the compatibility vector of center_1 in center_2, $v_{center1}$, is calculated and then integrated with v_1 and v_2 to form the compatibility vectors of ontology_1 and ontology_2 in center_2.

Before the distance adjustment, there are two compatibility vectors in center_2: one for ontology_3, and the other for center_1. The former will remain as is; while the latter will be replaced by two new vectors. The following procedure describes the generation of a new vector.

```
Input:
  -   compatibility vector v for center_1
      in center_2
  -   compatibility vector u for partner_i
      in center_1
Output:
  -   compatibility vector w for partner_i
      in center_2
begin
 for each dimension d in v
   sd = d's 1st sub-dimension's value
   nm = d's 2nd sub-dimension's value
   n1 = d's 3rd sub-dimension's value
   n2 = d's 4th sub-dimension's value
   create a new dimension nd in w
   nd's 3rd sub-dimension = n1
   nd's 4th sub-dimension = n2
   if sd > 0
     find in u the dimension od for concept nm
     sd_old = od's 1st sub-dimension's value
     nm_old = od's 2nd sub-dimension's value
     nd's 1st sub-dimension = getSD(sd_old, n1, n2)
     nd's 2nd sub-dimension = nm_old
   else (sd = 0)
     nd's 1st sub-dimension = sd
     nd's 2nd sub-dimension = nm
   end if
 end for
end
```

Pseudocode for new vector generation

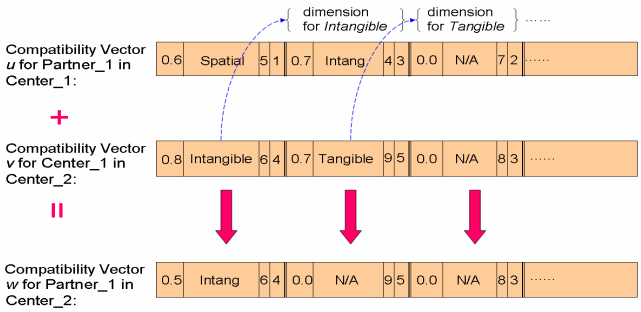

Figure 4. Example of new vector generation

The time complexity for the above procedure is *O(nlogn)*, because there are *n* dimensions in each vector, requiring *n* steps for the loop. Within each loop, all steps take constant time, except for the one finding dimension in *u*. Suppose in *u* the dimensions are indexed by the concept names, then a binary search is able to locate a specific dimension within *O(logn)*. Figure 4 exemplifies the operation of the above pseudocode.

5.3.2. After a SOCC is created. In cases where existing original ontologies change their schema information after a SOCC is created, we need to modify the compatibility vectors accordingly. There are several situations.

1. One or more new concepts are added.
2. No new concept is added, but new relationships are added.
3. No new concept is added, but existing relationships are removed.
4. No new concept is added, but existing relationships are modified.

An outline of our solution to the dynamic adjustment of a SOCC is as follows.

1. For case 1, a subgraph of the modified ontology needs to be merged with the center. A subgraph consists of the new concept and all its ancestors and descendants, together with all concepts having relationships with the new concept. After this merging, we relocate the new concept in the center. If the concept already exists in the center and no new information (relationship) is added, then only the compatibility vector for the modified ontology needs to be updated; otherwise all vectors need to be updated.
2. For case 2, if the new relationships already exist in the center, then only the compatibility vector for that modified ontology needs to be updated; otherwise all vectors need to be updated.
3. For case 3, if the removed relationships come from both the modified ontology and other original ontologies, then only the compatibility vector for this modified ontology needs to be updated; otherwise all related vectors (for those ontologies providing the removed relationships) need to be updated.
4. Case 4 is a combination of cases 2 and 3.

5.4. Features of compatibility vectors

When a partner from outside a SOCC requests partner(s) to collaborate with, it would like to choose those understanding its ontology best. The requesting partner first compares its own ontology with the center, and then searches in the compatibility vectors to find all partners understanding the concept of its interest and/or having a small ontology distance. If there is more than one candidate, the collaboration request will be sent to those with good qualities. Because the compatibility vectors are stored and maintained by the center, partners have no way to modify or manipulate the vectors. In this sense, the selection of partner(s) is objective and done with no bias.

5.4.1. Correctness of compatibility vectors – a precise approach. In this section, we prove that our approach obtains a correct compatibility for each partner. To record and maintain the proper compatibility of each partner inside a SOCC, the key is to obtain a correct center by which to evaluate the distance from it to each original ontology, and thereby acquire the corresponding compatibility vector. When a new partner and its associated ontology join the SOCC, instead of communicating with each existing partner, it only talks with the center. Therefore, if we can prove that the newly merged ontology is a correct new center, the correctness of compatibility vectors is guaranteed.

Lemma 1. When we merge two ontologies A and B using the algorithm in Section 4, the result is the same regardless of whether we merge A into B or B into A.

Proof by induction:

1. Base Case: when both A and B contain two concepts, i.e., besides the common root, *Thing*, A contains C_1 and B contains C_2.

   If we merge A into B according to the merging procedure in Section 4, *Thing* in A is considered equivalent to *Thing* in B; then C_1 is compared with all the direct children of the root in B, in this case C_2, to determine where to put C_1 in B. This is based on the *relocate* function. On the contrary, if we merge B into A, *Thing* in B is considered equivalent to *Thing* in A; then C_2 is compared with C_1 to determine where to put C_2 in A. Obviously, we obtain the same merged ontology in both cases.
2. Induction: Assume that Lemma 1 holds for all cases where the number of concepts contained in A and B are less than (i+1) and (j+1), respectively. Now consider the case where A and B contain (i+1) and (j+1) concepts, respectively.

   Suppose the superclass set of the $(i+1)^{th}$ concept in A, C_{i+1}, is $P_A(C_{i+1})$ and suppose the location of $P_A(C_{i+1})$ in merged ontology M is $P_M(C_{i+1})$. The position of C_{i+1} in M is determined by the relationships between C_{i+1} and all the direct children of $P_M(C_{i+1})$. From the inductive hypothesis we know that $P_M(C_{i+1})$ is identical no matter whether we merge A into B or merge B into A. Therefore, the position of C_{i+1} in M will be the same in both situations. That is, C_{i+1}, the $(i+1)^{th}$ concept in A, will be put into the same position in M for both merging orders. Similarly, the $(j+1)^{th}$ concept in B will also be put into the same position in M for both merging orders. So in the case where A and B contain (i+1) and (j+1) concepts, respectively, we have the same resultant ontology regardless of the merging order taken.

Theorem 1. The result of merging a number of ontologies is identical no matter by which order the ontologies are merged using the algorithm in Section 4.

Proof by induction:

1. Base Case: there are two ontologies to be merged.

   According to Lemma 1, when we merge two ontologies A and B, the result is the same no matter whether we merge A into B, or merge B into A.
2. Induction: Assume that Theorem 1 holds for all cases where the number of ontologies to be merged is less than (n+1). Now consider the case where we merge (n+1) ontologies. Let the indexes of these ontologies be: 1, 2, …, (n+1).

   Consider two arbitrary orders by which we merge these (n+1) ontologies: order_1 and order_2. Suppose the last index in order_1 and order_2 is i and j, respectively.

   - If i equals j, then the first n indexes in order_1 and order_2 are the same, just in different orders. We merge the first n ontologies to get $Merged_n$. According to the inductive hypothesis, $Merged_n$ in order_1 is identical with $Merged_n$ in order_2. Then we merge $Merged_n$ with the last ontology in both order_1 and order_2 and get the same result.
   - If i does not equal j, we mutate the first n indexes in order_1 and make the n^{th} index be j; then mutate the first n indexes in order_2 and make the n^{th} index be i. Now the first (n-1) indexes in both orders are the same (possibly in different orders), and the last two are (j, i) and (i, j), respectively. Notice that this kind of mutation will not affect the merging result of the first n ontologies according to our inductive hypothesis. We then merge the first (n-1) ontologies to get $Merged_{n-1}$. According to the hypothesis, $Merged_{n-1}$ in order_1 is identical with $Merged_{n-1}$ in order_2. Finally we merge $Merged_{n-1}$ with the last two ontologies in both orders and get the same result.

5.4.2. Complexity of compatibility vectors – an efficient approach.

(1) The time complexity of establishing a SOCC, along with the achievement of a mutual understanding of ontological concepts, is on the order of $O(mn^2)$, with n the number of the concepts in the center, and m the number of original ontologies. The process of creating a SOCC is one of generating a merged center ontology. For the ontology merging, $O(mn^2)$ is needed, because we need to merge m ontologies, and each merging procedure takes time $O(n^2)$ as described in Section 4.

(2) In order to dynamically update the compatibility vectors during the formation of a SOCC, extra time will be spent. According to the previous analysis, $O(nlogn)$ is needed for updating one partner, so the extra time for all partners is $O(mnlogn)$. Therefore, the total time complexity of establishing a SOCC becomes $O(mn^2 + mnlogn)$, which is still on the order of $O(mn^2)$.

(3) For the update after a SOCC is formed, the time complexity is only $O(n^2)$, because only one merging process is carried out.

(4) For partner selection, the time complexity is $O(n^2)$, because we only need to compare the ontology from the requesting partner with the center ontology.

6. Experimental results

6.1. Experiments on merging algorithm itself

Due to limited space, the experimental results for our merging algorithm are not shown in this paper. Please refer to [1] for details. Briefly, the resultant merged ontology has a promising performance in both *precision* and *recall* measurements (0.93 and 0.81 respectively).

6.2. Experiments with compatibility vectors

6.2.1. Correctness of compatibility vectors. We simulated a SOCC out of 16 ontologies [1]. Based on calculated compatibility vectors, we sorted the original ontologies

with regard to their qualities (encoded by ontology distance). We then asked two experts to rank the qualities of these ontologies manually; the result is the same as the one from our system.

6.2.2. Efficiency of compatibility vectors. A set of experiments has been conducted. We first fixed one of the original ontologies as the one requesting interaction, and simulated a SOCC out of the remaining 5, 10, and 15 ontologies as three experiment settings; then for each SOCC setting we did the following in two groups. In the first group the requesting ontology always interacted with the ontology with the best compatibility, while in the second group the interaction happened with a randomly chosen ontology. We compared the resultant merged ontologies from the two groups. The result is shown in Figure 5. It is clear that, after adopting our compatibility vectors, both precision and recall measurements have been improved. Therefore, in cases where sufficient resources are not available and only a certain number of business partners can be chosen for collaboration, our approach increases the efficiency by choosing more suitable partners.

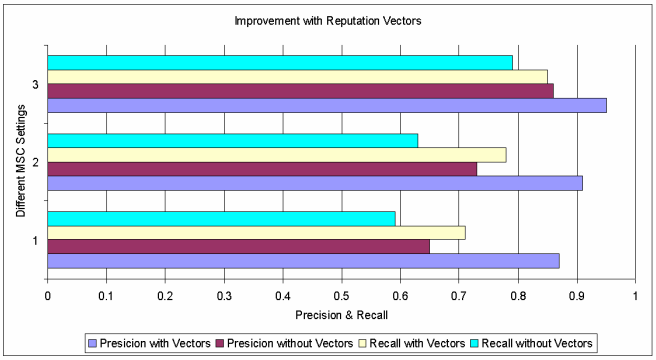

Figure 5. Improvement with Compatibility Vectors

7. Conclusion and future work

Service-oriented computing is a new paradigm for computing that supports the dynamic interaction of services from distributed business partners. Ontologies can help partners to understand the semantics of services from each other. However, alignments among ontologies are necessary to handle the inherent heterogeneity in individually developed ontologies. Ontology compatibility is an important issue during such an alignment process and the interaction among partners thereafter. To tackle this emerging challenge, we extend a previously proposed vector system to encode ontology compatibility. In addition, we present a probabilistic center ontology model for better recording and maintenance of ontology alignment results. We prove that our approach is a precise and efficient one; and show utility by a set of experiments.

We envision the following future work: (1) How to propagate the probabilities in the merged center ontology; (2) How to handle the vulnerability issue inherent in the centralized solution introduced by our use of a center ontology; and (3) What kind of mechanism is suitable if we simultaneously consider qualities of both ontologies and services.

8. References

[1] Huang, J., Zavala Gutiérrez, R., Mendoza, B., and Huhns, M. N. *Sharing Ontology Schema Information for Web Service Integration*, In: *Proceedings of 5th International Conference on Computer and Information Technology (CIT 2005)*, Shanghai, China, 2005.

[2] Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., and Halevy, A. *Learning to match ontologies on the Semantic Web*. In: *The VLDB Journal*, Vol. 12. Springer-Verlag 303 – 319, 2003.

[3] Do, H. H., Melnik, S., Rahm, E. *Comparison of schema matching evaluations*. In: *Proceedings of workshop on Web and Databases*, 2002.

[4] Madhavan, J., Bernstein, P. A., and Rahm, E. *Generic Schema Matching with Cupid*. In: *Proceedings of the 27th VLDB Conference, Springer-Verlag*, 2001.

[5] Noy, N. F., Musen, M. A. *Anchor-PROMPT: Using Non-Local Context for Semantic Matching*. In: *Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence* (IJCAI). Seattle, WA, 2001.

[6] Pan, R., Ding, Z., Yu, Y., and Peng, Y. *A Bayesian Network Approach to Ontology Mapping*. In: *Proceedings of the 4th International Semantic Web Conference*. Springer, 2005.

[7] Melnik, S., Garcia-Molina, H., and Rahm, E. 2002. *Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching*. In: *Proceedings of the 18th International Conference on Data Engineering*. IEEE Computer Society Press, 2002.

[8] Giunchiglia, F., Shvaiko, P., and Yatskevich, M. *S-Match: an algorithm and an implementation of semantic matching*. In: *Proceedings of the 1st European Semantic Web Symposium*, Vol. 3053. Springer-Verlag 61 – 75, 2004.

[9] Bilgin, A. S. and Singh, M. P. *A DAML-based repository for QoS-aware semantic web service selection*. Presented at *IEEE International Conference on Web Services*, 2004.

[10] Zhou, C., Chia, L.-T., and Lee, B. S. *DAML-QoS ontology for web services*. Presented at *IEEE International Conference on Web Services*, 2004.

[11] Kalepu, S., Krishnaswamy, S., and Loke, S. W. *Reputation = f(user ranking, compliance, verity)*. Presented at *IEEE International Conference on Web Services*, 2004.

[12] Huang, J., Dang, J., and Huhns, M. N. *Ontology Alignment as a Basis for Mobile Service Integration and Invocation*, submitted to *International Journal of Pervasive Computing and Communications* (*JPCC*).