Reconciling Ontologies for Coordination among E-business Agents

Jingshan Huang
Computer Science and Engineering Department
University of South Carolina
Columbia, SC 29208, USA
+1-803-777-3768
huang27@sc.edu

Jiangbo Dang
Computer Science and Engineering Department
University of South Carolina
Columbia, SC 29208, USA
+1-803-777-3768
dangj@engr.sc.edu

Michael N. Huhns
Computer Science and Engineering Department
University of South Carolina
Columbia, SC 29208, USA
+1-803-777-5921
huhns@sc.edu

ABSTRACT

E-business is heralded as having one of the most significant impacts on the Internet. It is well recognized that to meet the demands of typically dynamic and open e-markets requires e-businesses to cooperate and coordinate their activities. The first step towards this coordination is for e-business agents to understand each other's service description. Using ontologies can aid in this understanding. However, independently designed ontologies usually have heterogeneous semantics, so it is likely that each agent would have its own unique semantics. We first briefly present an automated schema-based approach to reconcile the ontologies from communicating agents as a basis for coordination of e-business. Then we introduce compatibility vectors as a means of measuring and maintaining ontology quality. In cases where resources are limited, those e-businesses with ontologies of good quality will be chosen to participate in a business transaction. Both experiments and proofs verify the benefits of our approach.

1. INTRODUCTION

Due to its ability of providing new opportunities and unparalleled efficiencies, e-business is being utilized by an increasing number of enterprises. They are discovering that exposing formerly internal activities, e.g., flight ticket purchase orders and automated order enactment, to external business collaborators can yield increased value. Although there is value in accessing the service provided by a single e-business through a semantically well-founded interface, greater value is derived through enabling a flexible composition of e-businesses, which not only creates new services, but also potentially adds value to existing ones [11]. In addition, in a dynamic environment such as the Web, to make use of agents acting on behalf of a human is bound to increase the ability to enable sharing tasks and automating processes, which eventually result in automated and seamless interoperation among e-businesses. As the first step of communication and integration of e-business activities, mutual understanding of semantics among e-business agents plays an important role in the entire coordination process.

An ontology serves as a declarative model for the knowledge and capabilities possessed by an agent or of interest to an agent. It forms the foundation upon which machine understandable service descriptions can be obtained, and as a result, it makes automatic coordination among agents possible. By providing a more comprehensible and formal semantics, the use of and reference to ontologies help the functionalities and behaviors of agents to be described, advertised, discovered, and composed by others. Eventually, these agents would be able to interoperate with each other, even though they have not been designed to do so.

However, because it is impractical to force all agents to adopt a global ontology that describes every concept that is or might be included as part of their services, ontologies from different agents typically have heterogeneous semantics. Due to this basic characteristic, agents need to reconcile ontologies and form a mutual understanding when they interact with each other. Only in this sense are agents able to comprehend and/or integrate the information from different sources, and enhance process interoperability thereafter.

In this paper, we first present an automated schema-based ontology merging algorithm to align heterogeneous ontologies. Then we focus on an important but mostly neglected research topic – how to select suitable agents with which to interact. Communicating agents have ontologies of different quality, in that those agents with high quality ontologies are more likely to understand and be understood by other agents, and this kind of mutual understanding is the prerequisite for interoperation. In this sense, ontology quality is used to measure the compatibility between agents. Notice that the quality of the service provided by agents is a separate research topic, which is not the focus of this paper. Based on this insight, we present a compatibility representation system built upon compatibility vectors to measure and maintain the ontology quality, and therefore the compatibility for each agent. Therefore, compatibility vectors form a basis for ontology quality differentiation and thereby help in choosing agents with good compatibilities. Our approach is able to create and adjust dynamically the agent compatibility with regard to the quality of its underlying ontology.

This paper advances the state of the art by (1) introducing the ontology issue into quality of service (QoS) and (2) exploring the use of a compatibility system to speed up the discovery of suitable agents. The rest of the paper is organized as follows. Section 2 briefly introduces related work in ontology matching, ontology application in e-service, and QoS. Section 3 discusses ontology heterogeneities in an e-business domain and outlines our solution. Sections 4 and 5 present our ontology merging algorithm and compatibility vector system, respectively. Section 6 briefly reports on our 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, such as GLUE [2], PROMPT [5], Cupid [4], COMA [3], Similarity Flooding [6], S-Match [7], and PUZZLE [1]. All of these systems take a schema-based approach, except for GLUE, which is an instance-based system.

PROMPT is a tool that uses 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 good performance in terms of precision and recall. However, user intervention is required, which is not always available in many applications.

Similarity Flooding utilizes a hybrid matching technique based on a measure of similarity spreading from similar nodes to their adjacent neighbors. Before a fix-point is reached, alignments between nodes are refined iteratively. This algorithm considers only simple linguistic similarity between node names, ignoring node properties and inter-node relationships.

Cupid 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, because a tree cannot represent multiple-inheritance, an important characteristic in ontologies.

COMA 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 other measures. Although being a composite schema matching tool, COMA does not integrate reasoning and machine learning techniques.

S-Match 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. Giunchiglia et al. claim that S-Match outperforms Cupid, COMA, and Similarity Flooding in measurements of precision, recall, overall, and F-measure. However, 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 an insufficient number of instances or no instance at all.

2.2 Related Work in Ontology Application in E-Service

The application of ontology in e-service environment has been studied widely. In [12], Honavar et al. describe several challenges in information extraction and knowledge acquisition from heterogeneous, distributed, autonomously operated, and dynamic data sources when scientific discovery is carried out in data rich domains. They outline the key elements of algorithmic and systems solutions for computer assisted scientific discovery in such domains, including ontology-assisted approaches to customizable data integration and information extraction from heterogeneous and distributed data sources. Ontology-driven approaches to exploratory data analysis from alternative ontological perspectives are also discussed.

An ontology-based information retrieval model for the Semantic Web is presented in [13]. The authors generate an ontology through translating and integrating domain ontologies. The terms defined in the ontology are used as metadata to markup the Web's content; these semantic markups are semantic index terms for information retrieval. The equivalent classes of semantic index terms are obtained by using description logic reasoner. It is claimed that the logical views of documents and user information needs, generated in terms of the equivalent classes of semantic index terms, can represent documents and user information needs well, so the performance of information retrieval can be improved effectively when suitable ranking function is chosen.

Tijerino et al. introduce an approach (TANGO) to generate ontologies based on table analysis [14]. TANGO aims to understand a table's structure and conceptual content; discover the constraints that hold between concepts extracted from the table; match the recognized concepts with ones from a more general specification of related concepts; and merge the resulting structure with other similar knowledge representations. The authors claim that TANGO is a formalized method of processing the format and content of tables that can serve to incrementally build a relevant reusable conceptual ontology.

2.3 Related Work in QoS

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. [8] 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, they extend the UDDI registry to a service repository by combing a relational database and an attribute ontology.

Zhou et al. [9] 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, they present a matchmaking algorithm for QoS properties.

One widely used QoS attribute is *user rating*, but it is subjective to the perception of an end user and is limited by the lack of an objective representation of performance history. Kalepu et al. [10] 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. ONTOLOGY HETEROGENEITY AND OUR SOLUTION

3.1 Ontology Heterogeneity in E-business

In order to collaborate with the services rendered by other agents, an e-business agent 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 an e-business environment can be envisioned as follows:
1. A number of agents form an e-business community (EBC) within which services provided by different agents might be integrated and have the ability to render a more complete and functional service. This integration requires the mutual understanding of the individual ontologies underlying each agent.
2. The agents outside this EBC 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 agent(s), especially under the situations where resources are limited.

Because of the fact that there is no global, common, and agreed-upon ontology, any agent can maintain and use ontologies according to its own conceptual view of the world. Consequently, ontological heterogeneity among different agents becomes an inherent characteristic in an EBC. The heterogeneity can occur in two ways: (1) Different ontologies could use different terminologies to describe the same conceptual model. That is, different terms could be used for the same concept or the same term could be used for different concepts. (2) Even if two ontologies use the same name for a certain concept, its corresponding properties and relationships with other concepts can be different.

Therefore, two major problems are envisioned here. First, during the formation of an EBC, how can it be ensured that all agents within the community have no problem in understanding each other's ontology? Second, an agent seeking coordination from outside this community would like to choose those agents that understand its ontology best. How can it ensure this selection is a correct one?

3.2 Overview of Our Solution

In order to solve the first problem – mutual understanding of ontologies within an EBC – we need an approach to match/align ontologies from different agents. By this means, concepts from communicating agents can be mutually understood, and possible integration of related services can be achieved. In the next section, we present an ontology merging algorithm to reconcile heterogeneous ontologies.

To tackle the second problem – the correct selection of agents that are most acquainted with the ontologies from outside the EBC – we introduce compatibility vectors as a means of measuring and maintaining ontology quality. By determining the compatibility for each constituent agent along with the formation of an EBC, not only the agents outside this community are able to select the best agent(s) with ease, but also a better mutual understanding of ontologies within the EBC is obtained.

4. A 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 an EBC. Our methodology, based on the ontology merging algorithm presented in [1], is summarized next.

4.1 Top-Level Procedure

The ontology merging is carried out at the schema level, that is, we concentrate on the structure (schema) information of ontologies. 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 two 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 "built-in" root. In order to merge two ontologies, G_1 and G_2, we try to relocate each concept (node) from one ontology into the other one. 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 put into the suitable place in the destination graph G_2 before the relocation of C itself. The pseudocode in Figure 1 describes this top-level procedure, whose time complexity is obviously $O(n^2)$, with n the number of concepts in the merged ontology.

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 the relationship between C and p_j's direct child(ren) in the following descending priorities: *equivalentclass*, *superclass*, and

Top-Level Procedure – *merge*(G_1, G_2)

Input: Ontology G_1 and G_2
Output: Merged Ontology G_2
begin
  new location of G_1 's root = G_2 's root
  for each node C (except for the root) in G_1
    Parent(C) = C's parent set in G_1
    for each member p_i in Parent(C)
      p_j = new location of p_i in G_2
      relocate(C, p_j)
    end for
  end for
end

Figure 1. Top-Level Procedure

subclass. Because *equivalentclass* has most significant and accurate information, it is straightforward that *equivalentclass* has been assigned the highest priority. For *superclass* and *subclass*, since we adopt a top-down procedure to relocate concepts, the former has been given a higher priority than the latter. If we cannot find any of these three relationships, the only option is for us to let *C* be another direct child of p_j.

4.3 Features of Our Merging Algorithm

Comparing to the research work mentioned in Section 2.1, our approach advances the state of the art of ontology merging techniques by including the following features.

- We carry out ontology merging at the schema level, and separate the performance of the merging algorithm from the availability of a large volume of instance data. As a result, it is more practical than GLUE in cases where there is no enough data to carry out instance-based matching.
- Our approach is fully automated. This feature is necessary, especially in terms of the seamless coordination and cooperation among workflows. Some semi-automated systems, PROMPT for example, require user intervention, which is not always available in a dynamic environment.
- We treat graph-structured ontologies, which are not only more complex than tree-structured ones (as in Cupid and S-Match), but also more realistic, because multiple-inheritance cannot be represented by a tree.

5. COMPATIBILITY VECTORS

We introduce compatibility vectors as a means of measuring and maintaining ontology quality, which determines the compatibility of the associated agent with regard to the mutual understanding of each other's service description. Along with the formation of an EBC, we create a *center* ontology by merging all the original ontologies; then the distances from the latter to the center ontology are suitably encoded in the compatibility vectors, and can be adjusted efficiently and dynamically during the period in which the EBC is formed. Based on the information contained in the vectors, agents are supposed to understand the ontology from each other without trouble. In addition, the agent from outside this community will have no difficulty choosing the agents with good compatibilities, which is, in an objective sense, with no bias.

5.1 Center Ontology Formation

The center ontology is generated by merging all original ontologies, step by step, as each new agent joins an EBC. At the beginning, when there is only one agent, its ontology is regarded as the center. With new agents join the community, the new ontologies are merged with the current center one. The resultant merged ontology is the newly obtained center ontology.

5.2 Ontology Distance and Compatibility Vectors

5.2.1 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 ontology, the original ontology can understand it, but possibly with less accurate and/or complete 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 $d_{concept}$:

$$d_{concept} = w_1 * n_{sub\text{-}super} + w_2 * n_{other},$$

with the constraint of ($w_1 + w_2 = 1$). $n_{sub\text{-}super}$ 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 assigned to different kinds of relationship, including *subclass*, *superclass*, *disjointWith*, *parts*, *owns*, *contains*, and *causes*, etc. Because the sub/superclass relationship is the most important one in an ontology schema, w_1 will be given a greater value than w_2.

Consider the ontologies in Figures 2(a) and 2(b). In ontology_1, concept "Intangible" has one *superclass* ("AbstractThing"); four *subclasses* ("TemporalThing", "SpatialThing", "Mathematical", and "IntangibleIndividual"); and one *disjointWith* relationship (with "PartiallyTangible"). In the merged center_1, the concept "Intangible" has more information from the other ontologies: one more *superclass* ("PartiallyIntangible"); one more *disjointWith* relationship (with "Tangible"); and one more *subclass* ("OtherIntangibleStuff"). Thus, the concept distance from "Intangible" in ontology_1 to "Intangible" in center_1 is $w_1 * 2 + w_2 * 1$. Also notice that the concept distance formula is suitable for both situations, i.e., independent of whether the original ontology recognizes that concept or not. For example, if in ontology_1 there is no concept "Intangible", then the distance becomes $w_1 * 7 + w_2 * 2$.

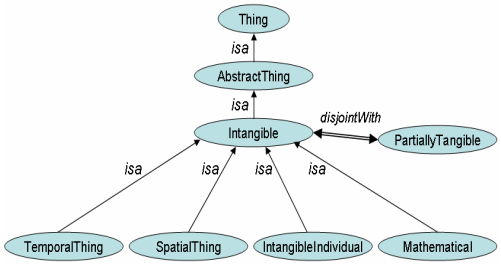

Figure 2(a). Graphical Representation for Ontology_1

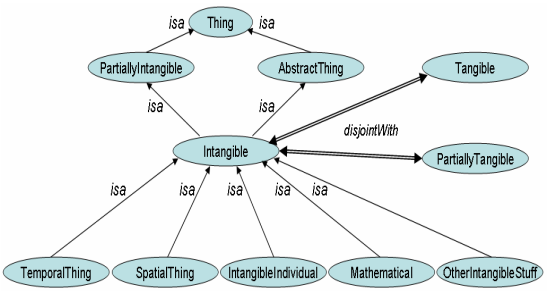

Figure 2(b). Graphical Representation for Center_1

5.2.2 *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.

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

Recall that the concept set of the original ontology is a subset of that of the center ontology, and the concept distance is encoded by the missing relationships in the original ontology compared to the center. The above formula 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.

5.2.3 *Compatibility Vectors*

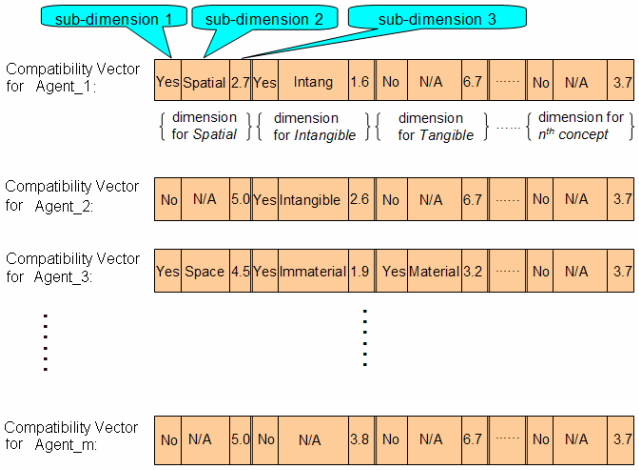

Figure 3. Compatibility Vectors

Inside the center ontology, 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 ontology. Therefore, all compatibility vectors have identical number of dimensions, i.e., equaling to the number of the concepts in the center ontology. Each dimension has three sub-dimensions. The first sub-dimension tells us whether the original ontology understands this concept or not; the second sub-dimension records the concept name in the original ontology if the latter does recognize that concept; the third sub-dimension encodes the distance from the concept of the original ontology to the concept of the center ontology. An example of compatibility vectors is shown in Figure 3.

For the first concept ("Spatial") in the center ontology, provider_1 knows it as "Spatial" and has a concept distance of 2.7; provider _3 also understands this concept, but with a different name ("Space") and a bigger concept distance of 4.5; neither provider _2 nor provider _m recognizes concept "Spatial", therefore, they have the same concept distance (5.0).

5.3 Dynamically Adjusting Compatibility Vectors

As mentioned before, when there is only one agent, its compatibility is perfect. In the compatibility vectors stored in the center ontology, each concept distance has a value of zero. However, with the adding of new agents into this EBC, the compatibilities for existing agents might be changed because newly joined agents could contain ontologies with more accurate and/or complete information.

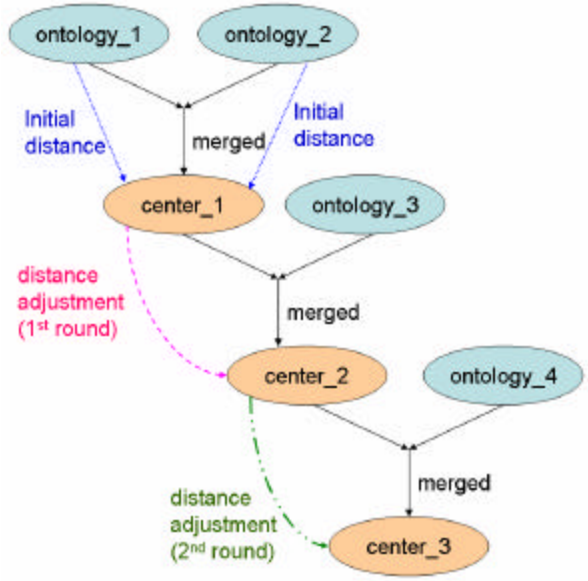

Figure 4. Dynamic Adjustment of Compatibility Vectors

An example is shown in Figure 4, demonstrating the process of dynamic distance adjustment. After ontology_1 and ontology_2 are merged to generate center_1, the distance between these two original ontologies and the merged one (center_1) is calculated and stored in the compatibility vectors of center_1. Upon the joining of ontology_3 and the generation of center_2, the compatibility vector for center_1 in center_2 is calculated and integrated with the compatibility vectors for ontology_1 and ontology_2 in center_1; then we generate the compatibility vectors for ontology_1 and ontology_2 in center_2. This is explained in detail next.

For example, we have compatibility vectors in both center_1 and center_2. Now we want to update the compatibility vectors in center_2.

Originally there are two compatibility vectors in center_2: one for ontology_3, and the other for center_1. The former will remain the same as is; while the latter will be replaced by several new vectors, the number of which is determined by the number of the vectors in center_1 (two in our example).

Remember that center_1 has one vector for each agent when center_1 is generated. Each vector in center_1 will be integrated with the vector for center_1 in center_2, therefore creating a new

vector correspondingly in center_2. The following procedure describes the generation of such a new vector.

```
Input:
  -   compatibility vector v for center_1
      in center_2
  -   compatibility vector u for agent_i
      in center_1
Output:
  -   compatibility vector w for agent_i
       in center_2

begin
 for each dimension d in v
   yn = d's first sub-dimension's value
   nm = d's second sub-dimension's value
   dis = d's third sub-dimension's value

   create a new dimension nd in w
   if yn = "Yes"
    find in u the dimension od for concept nm
    yn_old = od's first sub-dimension's value
    nm_old = od's second sub-dimension's value
    dis_old = od's third sub-dimension's value

    nd's first sub-dimension = yn_old
    nd's second sub-dimension = nm_old
    nd's third sub-dimension = dis + dis_old
   else (yn = "No")
    nd's first sub-dimension = yn
    nd's second sub-dimension = nm
    nd's third sub-dimension = dis
   end if
 end for
end
```

Pseudocode for New Vector Generation

It is not difficult to figure out that the time complexity for the above procedure is *O(n * log n)*, 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(log n)*.

Figure 5 exemplifies how the above pseudocode works. There are two source vectors, *u* and *v*, and we traverse the second one, one dimension each time.

1. The values for the first dimension are "Yes", "Intangible", and "2.3". We then find the dimension for "Intangible" in *u*, and obtain ("Yes", "Intang", and "1.6"). Finally we calculate the values for the new dimension in the resultant vector *w*, which are "Yes", "Intang", and "3.9" (the result of 1.6 + 2.3).
2. The values for the second dimension are "Yes", "Tangible", and "1.7". After we obtain the values for dimension "Tangible" in *u* ("No", "N/A", and "6.7"), we figure out the values for the new dimension in *w* are "No", "N/A", and "8.4" (the result of 6.7 + 1.7).
3. The values for the third dimension are "No", "N/A", and "5.9". We simply copy these three values into the new dimension in *w*.
4. This continues until we finish the traverse of all dimensions in *v*.

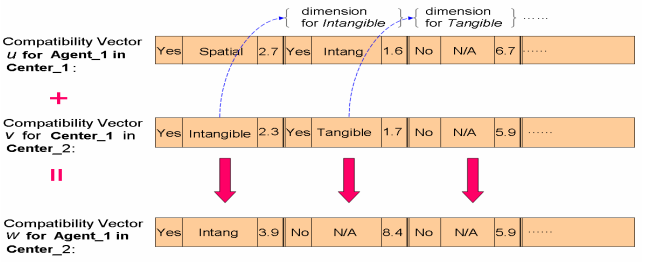

Figure 5. Example of New Vector Generation

5.4 Understanding Ontologies through Compatibility Vectors

The center ontology maintains the compatibility vectors for all original ontologies; in addition, the vectors themselves contain such information as whether an original ontology understands a specific concept or not, what is the concept name in the original ontology, and so on. Therefore, if two agents would like to try to understand each other's ontology, they can simply refer to the center ontology and obtain the corresponding compatibility vectors. By this means, compatibility vectors help agents in their mutual understanding of ontological concepts.

5.5 Selecting Agents through Compatibility Vectors

When an agent from outside this EBC requests for agent(s) to coordinate with, it would like to choose those that understand its ontology best. The requesting agent first compares its own ontology with the center ontology, and then searches in the compatibility vectors to find all agents understanding the concept of its interest. If there is more than one candidate, the coordination request will be sent to those with good compatibilities, that is, with concept and/or ontology distance below a certain threshold. Such a threshold could be either specified by the requesting agent, or otherwise determined by the center ontology. Because the compatibility vectors are stored and maintained by the center ontology, the agents have no way to modify or manipulate the vectors. In this sense, the selection of agent(s) is objective and without bias.

5.6 Features of Compatibility Vectors

5.6.1 Correctness of Compatibility Vectors – A Precise Approach

In this section, we prove that our approach obtains correct compatibilities for agents. To record and maintain the proper compatibility of each agent inside an EBC, the key is to obtain a correct center ontology by which to evaluate the distance from it to each original ontology, and thereby acquire the corresponding compatibility vector. When a new agent joins the EBC, instead of communicating with each existing agent, it only talks with the center ontology. Therefore, if we can prove that the newly merged ontology is a correct new center, the correctness of compatibility vectors is guaranteed.

First, we point out that according to the merging algorithm in Section 4, each time we merge two ontologies, the resultant one

will contain all information from both original ones. Next, we introduce Lemma 1 and Theorem 1.

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 merge B into A.

Proof by induction:

1. Base Case: when both A and B contain two concepts, i.e., besides one common built-in root, "Thing", A contains C_1 and B contains C_2.
   If we merge A into B according to the Top-Level Merging Procedure in Section 4, "Thing" in A is considered equivalent with "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 inside the Top-Level Merging Procedure. On the contrary, if we merge B into A, "Thing" in B is considered equivalent with "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 is 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 also 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 in both merging orders. Similarly, the $(j+1)^{th}$ concept in B will also be put into the same position in M in both merging orders. So in the case where A and B contain (i+1) and (j+1) concepts, respectively, we still have the same resultant ontology regardless of the merging order taken.

Theorem 1. The final result of merging a number of ontologies is identical no matter by which order the original 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 are 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 will 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 order_1 and order_2 are in common (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 order_1 and order_2 and will get the same result.

5.6.2 Complexity of Compatibility Vectors – An Efficient Approach

The time complexity of establishing an EBC, along with the achievement of mutual understanding of ontological concepts, is on the order of $O(n^2 * m)$, with n the number of the concepts in the center, and m the number of original ontologies. For the ontology merging, $O(n^2 * m)$ is needed, because we need to merge m ontologies, and each merging procedure takes time $O(n^2)$ as described in Section 4. In addition, in order to dynamically update the compatibility vectors, extra time will be spent. According to the previous analysis, $O(n * log\ n)$ is needed for updating one agent, so the extra time for all agents is $O(n * log\ n * m)$. Therefore, the total time complexity becomes $O((n^2 + n * log\ n) * m)$, which is on the same order of $O(n^2 * m)$.

For agent selection, the time complexity is $O(n^2)$. We only need to compare the ontology from the requesting agent 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 omitted 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 an EBC 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 have been conducted. We first fixed one original ontology as the one from the coordination-requesting agent, and simulated an EBC out of the remaining 5, 10, and 15

ontologies as three experimental settings; then for each EBC setting we did the following in two groups. In the first group the requesting ontology always interacted with the ontology with the best quality, while in the second group the interaction happened with a randomly chosen ontology. We compared the resultant merged ontologies from two groups. The result is shown in Figure 6. 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 agents can be chosen for coordination, our approach increases the efficiency by choosing suitable agents.

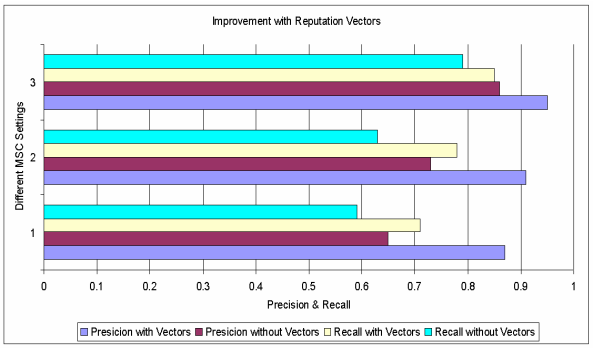

Figure 6. Improvement with Compatibility Vectors

7. CONCLUSION AND FUTURE WORK

E-business has a significant impact on the internet revolution. In order to meet the new demands of highly dynamic environment and open e-markets, there has been a growing need for e-businesses to coordinate their activities. The first step towards this coordination is for e-business agents to understand each other's service description. Although using ontologies can aid in this comprehension, independently designed ontologies usually have heterogeneous semantics, resulting in each agent having its own unique semantics. To tackle this emerging challenge, we present an automated approach carried out at the schema level to reconcile ontologies as a basis for agent communication and coordination; then we introduce compatibility vectors as a method to evaluate and maintain ontology qualities, thereby handling the problem of how to choose ontologies with high quality, and therefore choose agents with good compatibilities. We not only prove that our approach is both precise and efficient, but also show promising results experimentally.

Some future work is envisioned here: (1) our current approach makes use of a center ontology, but introduces the problem of how to handle the vulnerability issue inherent in this centralized solution, (2) how to maintain compatibility vectors when existing agents modify their corresponding ontologies, 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] 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.

[7] 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.

[8] 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.

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

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

[11] Singh, M. P. and Huhns, M. N. *Service-Oriented Computing Semantics, Processes, Agents*. 1st edition. Wiley Bowman, B., 2005.

[12] Honavar, V., Andorf, C., Caragea, D., Silvescu, A., Reinoso-Castillo, J., and Dobbs, D. *Ontology-Driven Information Extraction and Knowledge Acquisition from Heterogeneous, Distributed Biological Data Sources*, In: *Proceedings of the IJCAI-2001 Workshop on Knowledge Discovery from Heterogeneous, Distributed, Autonomous, Dynamic Data and Knowledge Sources*.

[13] Song, J., Zhang, W., Xiao, W., Li, G., Xu, Z. *Ontology-Based Information Retrieval Model for the Semantic Web*, In: *Proceedings of IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, IEEE Computer Society Press, Washington DC, USA, 2005, pp. 152-155.

[14] Tijerino, Y. A., Embley, D. W., Lonsdale, D. W., Ding, Y., and Nagy, G. *Towards Ontology Generation from Tables*, In: *World Wide Web: Internet and Web Information Systems, Vol. 8, Issue 3*, Kluwer Academic Publishers, Hingham, MA, USA, 2005, pp. 261-285.