Ontology Alignment as a Basis for Mobile Service Integration and Invocation

Jingshan Huang[1]
Jiangbo Dang
Michael N. Huhns
Computer Science and Engineering Department
University of South Carolina
Columbia, SC 29208, USA
{*huang27, dangj, huhns*}*@engr.sc.edu*

Yongzhen Shao
Software School
Fudan Unversity
Shanghai 200433, China
shaoyz@fudan.edu.cn

Received: December 31 2005; revised: May 07 2006

***Abstract*— The limited capabilities of typical mobile devices can be extended by using services from other devices. To use such services, a mobile device must be able to comprehend their descriptions. Ontologies can aid in this comprehension, but ontologies designed independently for each device would have heterogeneous semantics. This paper presents an automated schema-based approach to align the ontologies from interacting devices as a basis for mobile service invocation. When the ontologies are ambiguous about the services provided, we introduce compatibility vectors as a means of maintaining ontology quality and deciding which service to choose to reduce the ambiguity. Our approach is verified both experimentally and theoretically.**

***Index Terms*— Ontology Merging, Ontology Compatibility, Mobile Service**

I. INTRODUCTION

Mobile computing is becoming more widespread and increasingly important. Mobile portable devices already outnumber traditional desktop computers and are expected to determine the view of computers for future generations. However, mobile devices typically have rather limited capabilities. To overcome this limitation, a mobile device can make use of the functionalities and services provided by other mobile devices, and thereby extend its own capabilities. The first step for mobile devices to achieve this goal will be to understand the descriptions of services that they can provide to each other. Only in this way can the future integration and/or invocation of mobile services take place automatically and successfully.

An ontology serves as a declarative model for the knowledge and capabilities possessed by a device or of interest to a device. It forms the foundation upon which machine understandable description of services can be obtained, and as a result, automatic interaction among mobile devices is enabled. Therefore, adding ontologies into the mobile service scenario will facilitate the extension of mobile device capabilities by providing a more comprehensible and formal semantics. The functionalities and behaviors of mobile services can be described, advertised, discovered, and composed by others through the use of and reference to ontologies. Eventually, these mobile services would be able to interoperate with each other, even though they have not been designed to work together. This is the vision for pervasive computing among mobile devices.

However, because it is impractical to have a global ontology that describes every concept that is or might be included as part of the mobile services, ontologies from different mobile devices typically have heterogeneous semantics. Due to this basic characteristic, mobile devices need to reconcile ontologies and form a mutual understanding when they interact with each other. Only in this sense are mobile services able to comprehend and/or integrate the information from different sources, and enhance service 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 a mobile device can select suitable ontologies to interact with. We introduce the concept of compatibility vectors as a means of evaluating and maintaining ontology quality, and use this as a basis for suitability of ontology selection. Our approach is able to create and adjust dynamically the compatibilities of mobile devices with regard to the quality

[1]Corresponding author Tel./Fax: +1-803-777-3768/+1-803-777-3767.

of their underlying ontologies.

The rest of the paper is organized as follows. Section II briefly introduces related work in ontology matching and Quality of Service (QoS). Section III discusses the challenges after the introduction of ontologies into mobile services. Sections IV and V present our ontology merging algorithm and compatibility vector system, respectively. Section VI briefly reports on the experiment results, and Section VII concludes with future work.

II. RELATED WORK

A. Related Work in Ontology Matching

Many researchers have investigated the problem of ontology matching, mostly using one of two approaches: instance-based and schema-based. All of the following systems belong to the latter, except for GLUE [2].

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 insignificant number of instances or no instances at all.

PROMPT [5] is a tool making use of linguistic similarity matches between concepts for initiating the merging or alignment process, and then use 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.

COMA [3] provides an extensible library of matching algorithms, a framework for combining results, and evaluation platform as well. According to their evaluation, COMA is performing well in terms of precision, recall, and overall measures. Although being a composite schema matching tool, COMA does not integrate reasoning and machine learning techniques.

Similarity Flooding [6] 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, because a tree cannot represent multiple-inheritance, an important characteristic in ontologies.

S-Match [7] 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.

B. 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. [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, they extend the UDDI registry to a service repository by combining 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, they present a matchmaking algorithm for QoS properties.

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.

III. CHALLENGES OF ONTOLOGY INTEGRATION INTO MOBILE SERVICES

A. Adding Ontologies into Mobile Services

In order to integrate and invoke the services rendered by other mobile devices, a mobile device must be able to comprehend the descriptions about those services as a first step. Being a formal knowledge representation model, ontologies can aid in this comprehension by providing the necessary semantics during communications among mobile devices.

An example scenario of the interaction among different mobile devices can be envisioned as follows.

1) A number of mobile devices form a mobile service community (MSC) within which services provided by different devices might be integrated and have the ability to render better services. This integration requires the mutual understanding of the individual ontologies underlying each service.
2) The mobile devices outside this MSC can ask for help from the community and consume its services, either the original ones or the integrated one. This invocation 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.

B. Problems Encountered

Because of the fact that there is no global, common, and agreed-upon ontology, any mobile device can maintain

and use ontologies according to its own conceptual view of the world. Consequently, ontological heterogeneity among different mobile devices becomes an inherent characteristic in a mobile computing environment. 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 C, its corresponding properties and relationships with other concepts can be different.

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

C. Overview of Our Approach

In order to solve the first problem - mutual understanding of ontologies within a MSC, we need an approach to match/align ontologies from different mobile devices. By this means, concepts from communicating devices can comprehend each other, 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 mobile devices that are most acquainted with the ontologies from service-consuming devices, we introduce compatibility vectors as a means of measuring and maintaining the ontology quality. By setting up the compatibility for each mobile device along with the formation of a MSC, not only the mobile devices seeking service from this community are able to select the best service provider(s) with ease, but also a better mutual understanding of ontologies within the MSC is obtained.

IV. A SCHEMA-BASED ONTOLOGY MERGING ALGORITHM

Our goal is to construct a merged ontology from two original ones. Although there does not exist such a global and agreed-upon ontology, we do assume that there is a common metamodel, i.e., OWL DL, for the ontologies to be merged, and we also assume that natural language provides common semantics during the ontology merging process.

A. 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*)|*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 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 below describes this top-level procedure, whose time complexity is obviously *O(n^2)*, with *n* the number of concepts in the merged ontology.

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

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

An implementation detail is worth mentioning here. Because of the characteristics of traversing a directed acyclic graph, there is a possibility that one or more parents of a certain concept may not have been relocated before that concept is visited. However, at least one of the parents will have been relocated. In this case, we revisit the target concept after all its parents have been visited. Notice that progress is guaranteed, because the graphs in question are acyclic.

B. Relocate Function

The relocate function in the top-level procedure is used to relocate *C* into a subgraph rooted by p_j. The following pseudocode shows the details of this function.

Input: nodes N_1 and N_2
Output: the modified structure of N_2 according to information from N_1
begin
  if there exists any *equivalentclass* of N_1 in the child(ren) of N_2
    merge N_1 with it
  else if there exists any *subclass* of N_1 in the child(ren) of N_2
    Children(N_1) = set of such subclass(es)
    for each member c_i in Children(N_1)
      add links from N_2 to N_1 and from N_1 to c_i
      remove the link from N_2 to c_i
    end for
  else if there exists any *superclass* of N_1 in the child(ren) of N_2

```
    Parent(N1) = set of such superclass(es)
    for each member p_i in Parent (N1)
        recursively call relocate(N1, p_i)
    end for
  else
    add a link from N2 to N1
  end if
end
```

Relocate Function - *relocate*(N_1, N_2)

Our 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 *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.

Notice that there is a recursive call within relocate. This recursion is guaranteed to terminate because the number of the nodes within a graph is finite, and the worst case is to call relocate repetitively until the algorithm encounters a node without any child.

To determine the relationship between *C* and p_j's direct child(ren), we need to consider both the linguistic and the contextual features. The meaning of a concept is determined by two aspects: (1) its linguistic feature - concept name - and (2) its contextual feature - property list and the relationship(s) with other concept(s). These two features are discussed next; they together specify a concept's semantics.

C. Linguistic Matching

The name of a concept reflects the meaning that the ontology designer intended to encode. Our approach uses string matching techniques to match linguistic features. Furthermore, we integrate WordNet by using the JWNL API [8] in our system. In this way, we are able to obtain the synonyms, antonyms, hyponyms, and hypernyms of an English word. In addition, WordNet performs some stemming, e.g., the transformation of a noun from plural to singular form.

We claim that for any pair of ontology concepts *C* and *C'*, their names N_C and $N_{C'}$ have the following mutually exclusive relationships, in terms of their linguistic features (the $v_{linguistic}$ mentioned below refers to the similarity between two concept names).

- *anti-match*: N_C is a antonym of $N_{C'}$, with $v_{linguistic}$ = 0;
- *exact-match*: either N_C and $N_{C'}$ have an exact string matching, or they are the synonyms of each other, with $v_{linguistic}$ = 1;
- *sub-match*: N_C is either a postfix or a hypernym of $N_{C'}$, with $v_{linguistic}$ = 1;
- *super-match*: $N_{C'}$ is either a postfix or a hypernym of N_C, with $v_{linguistic}$ = 1;
- *leading-match*: the leading substrings from N_C and $N_{C'}$ match with each other, with $v_{linguistic}$ equaling the length of the common leading substring divided by the length of the longer string. For example, "active" and "actor" have a common leading substring "act", resulting in a leading-match value of $\frac{3}{6}$;
- *other*: $v_{linguistic}$ = 0.

When relocating *C*, we perform the linguistic matching between *C* and all the candidate concepts in the destination graph G_2, and build a list for each of three types of relationship of *C*, i.e., *equivalentclass*, *superclass*, and *subclass*. For each candidate concept *C'*, if an exact-match or a leading-match (with $v_{linguistic} \geq$ *threshold*) is found, we put *C'* into *C*'s candidate equivalentclass list; if a sub-match is found, we put *C'* into *C*'s candidate subclass list; and if a super-match is found, we put *C'* into *C*'s candidate superclass list. Then we continue the contextual matching between *C* and each concept in the three candidate lists to obtain further information.

D. Contextual Matching

In essence, the context of an ontology concept *C* consists of two parts: its relationship(s) with other concept(s), and its property list. The former include *equivalentclass*, *subclass*, *superclass*, and *sibling*, and is implicitly embodied in the graph traverse process mentioned previously. The latter is discussed next.

Considering the property lists, P(*C*) and P(*C'*), of a pair of concepts *C* and *C'* being matched, our goal is to calculate their similarity value. $v_{contextual}$ = $w_{required} \cdot v_{required}$ + $w_{non-required} \cdot v_{non-required}$, where $v_{required}$ and $v_{non-required}$ are the similarity values calculated for the *required* property list and *non-required* property list, respectively. $w_{required}$ and $w_{non-required}$ are the weights assigned to each list. Notice that $v_{required}$ and $v_{non-required}$ are calculated by the same procedure.

Suppose the number of properties in two property lists (either required or non-required ones), P_1 and P_2, is n_1 and n_2, respectively. Without loss of generality, we assume that $n_1 \leq n_2$. There are three different matching models between two properties.

1) *total-match*
   - The linguistic matching of the property names results in either an exact-match, or a leading-match with $v_{linguistic} \geq$ *threshold*; and
   - The data types match exactly.

   Let v_t = number of properties with a total-match, and f_t=$\frac{v_t}{n_1}$. Here f_t is a *correcting* factor for *name-match*, embodying the integration of heuristic reasoning. We claim that between two property lists, the more pairs of properties being regarded as total-match, the more likely that the remaining pairs of properties will also hit a match as long as the linguistic match between their names is above a certain threshold value. For example,

assume that both P_1 and P_2 have ten properties. If there are already nine pairs with a total-match, and furthermore, if we find out that the names in the remaining pair of properties are similar with each other, then it is much more likely that this pair will also have a match, as opposed to the case where only one or two out of ten pairs have a total-match.

2) *name-match*
- The linguistic matching of the property names results in either an exact-match, or a leading-match with $v_{linguistic} \geq$ *threshold*; but
- The data types do not match.

Let v_n = number of properties with a name-match, and f_n=$\frac{v_t+v_n}{n_1}$. Similarly to f_t, f_n also serves as a correcting factor for *datatype-match*.

3) *datatype-match*
Only the data types match. Let v_d = number of properties with a datatype-match.

After we find all the possible matching models in the above order, we can calculate the similarity between the property lists as $\frac{1}{n_1}(v_t \cdot w_1 + v_n(w_2 + f_t \cdot w'_2) + v_d(w_3 + f_n \cdot w'_3))$, where:

- w_i (*i* from 1 to 3) is the weight assigned to each matching model; and
- w'_i (*i* from 2 to 3) is the correcting weight assigned to the matching models of name-match and datatype-match.

E. Domain-Independent Reasoning

Remember that to merge two ontologies, we in essence are to relocate each concept from one ontology into the other one. After we obtain the linguistic and contextual similarities, we apply a domain-independent reasoning rule to infer the relationship between the target concept to be relocated and the candidate concept in the destination ontology.

1) Relationships among Property Lists: Suppose we have two ontologies A and B, each of which is designed according to the OWL DL specification. Furthermore, let *n(A)* and *n(B)* be the sets of concepts in A and B, respectively, with n_i*(A)* and n_j*(B)* be the individual concept for each set ($1 \leq i \leq |n(A)|$ and $1 \leq j \leq |n(B)|$), and P(n_i*(A)*) and P(n_j*(B)*) be the property list for each individual concept.

Consider the property lists P(n_i*(A)*) and P(n_j*(B)*), let s_i and s_j be the set size of these two lists. There are four mutually exclusive possibilities for the relationship between P(n_i*(A)*) and P(n_j*(B)*):

- P(n_i*(A)*) and P(n_j*(B)*) are consistent with each other if and only if

  i. Either $s_i = s_j$ or $\frac{abs(s_i - s_j)}{s_i + s_j} \leq$ *threshold*, and

  ii. $v_{contextual} \geq$ *threshold*

  We denote the corresponding concepts n_i*(A)* and n_j*(B)* by n_i*(A)* $\overset{p}{\leftrightarrow}$ n_j*(B)*;

- P(n_i*(A)*) is a subset of P(n_j*(B)*) if and only if

  i. $s_i \leq s_j$, and
  ii. $v_{contextual} \geq$ *threshold*

  We denote the corresponding concepts n_i*(A)* and n_j*(B)* by n_i*(A)* $\overset{p}{\rightarrow}$ n_j*(B)*;

- P(n_i*(A)*) is a superset of P(n_j*(B)*) if and only if

  i. $s_i \geq s_j$, and
  ii. $v_{contextual} \geq$ *threshold*

  We denote the corresponding concepts n_i*(A)* and n_j*(B)* by n_i*(A)* $\overset{p}{\leftarrow}$ n_j*(B)*;

- Other.

2) Relationships among Concepts: Given any two ontology concepts, we can have the following five mutually exclusive relationships between them:
- *subclass*, denoted by $\subseteq$
- *superclass*, denoted by $\supseteq$
- *equivalentclass*, denoted by $\equiv$
- *sibling*, denoted by $\approx$
- *other*, denoted by $\neq$

3) Reasoning Rule: If two classes share a same parent, then their relationship is one of: *equivalentclass*, *superclass*, *subclass*, and *sibling*.

- Preconditions:
n_{i1}*(A)* $\supseteq$ n_{i2}*(A)* and
n_{j1}*(B)* $\supseteq$ n_{j2}*(B)* and
n_{i1}*(A)* $\equiv$ n_{j1}*(B)* and
1) (the names of n_{i2}*(A)* and n_{j2}*(B)* have either an exact-match, or a leading-match with $v_{linguistic} \geq$ *threshold*) and n_{i2}*(A)* $\overset{p}{\leftrightarrow}$ n_{j2}*(B)*
2) (the name of n_{j2}*(B)* is a sub-match of the name of n_{i2}*(A)*) and n_{i2}*(A)* $\overset{p}{\rightarrow}$ n_{j2}*(B)*
3) (the name of n_{j2}*(B)* is a super-match of the name of n_{i2}*(A)*) and n_{i2}*(A)* $\overset{p}{\leftarrow}$ n_{j2}*(B)*
4) None of above three holds

- Conclusion:
1) n_{i2}*(A)* $\equiv$ n_{j2}*(B)*
2) n_{i2}*(A)* $\supseteq$ n_{j2}*(B)*
3) n_{i2}*(A)* $\subseteq$ n_{j2}*(B)*
4) n_{i2}*(A)* $\approx$ n_{j2}*(B)*

The intuition behind our reasoning rule is as follows. After the linguistic matching phase we obtain three candidate lists for target concept *C*. For each concept *C'* in these lists, we then try to find the contextual similarity between *C* and *C'* to make the final decision.

F. Features of Our Merging Algorithm

Comparing to the research work mentioned in Section II, 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 successful invocation and seamless integration of mobile services dynamically. 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.
- We exploit both the linguistic and the contextual features of a concept, and combine these two features to determine what a concept means in an ontology. It is more advanced than Similarity Flooding, which considers concept names alone and can represent only partial semantics of ontological concepts.
- We incorporate WordNet into the linguistic analysis phase, under the assumption that natural language provides common semantics; and then integrate heuristic knowledge into the contextual analysis phase.
- We apply a reasoning rule to infer new relationships among concepts. This rule is based on the domain-independent relationships subclass, superclass, equivalentclass, and sibling, together with each concept's property list.

V. COMPATIBILITY VECTORS

We introduce compatibility vectors as a means of measuring and maintaining the ontology quality, which determines the compatibilities of mobile devices providing services. Along with the formation of a MSC, 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 compatibility vectors, and can be adjusted efficiently and dynamically during the period in which the MSC is formed. Based on the information contained in the vectors, mobile devices are supposed to understand the ontology from each other without trouble. In addition, for the mobile devices seeking services from outside this community, there is no difficulty for them to choose the devices with good compatibilities, which is, in an objective sense, with no bias.

A. Center Ontology Formation

The center ontology is generated by merging all original ontologies, step by step, as each new mobile device joins a MSC. At the beginning, when there is only one mobile device, its ontology is regarded as the center ontology. Each time a new device joins the community, the new ontology is merged with the current center one. The resultant merged ontology is the newly obtained center ontology.

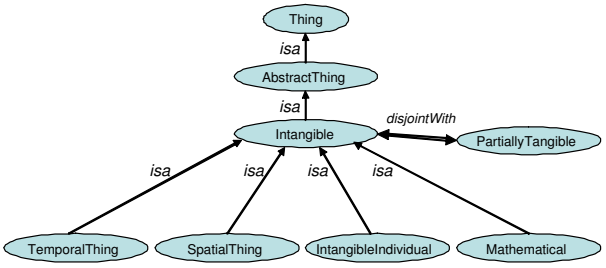


Fig. 1. Graphical Representation for $Ontology_1$

B. Ontology Distance and Compatibility Vectors

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: $d = w_1 \cdot n_{sub-super} + w_2 \cdot n_{other}$, with the constraint of ($w_1 + w_2 = 1$). $n_{sub-super}$ 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*, *equivalentclass*, *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 1 and 2. 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 ($w_1 \cdot 2 + w_2 \cdot 1$). 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 ($w_1 \cdot 7 + w_2 \cdot 2$).

2) Ontology Distance: After each concept distance has been calculated as shown above, we can continue to figure out

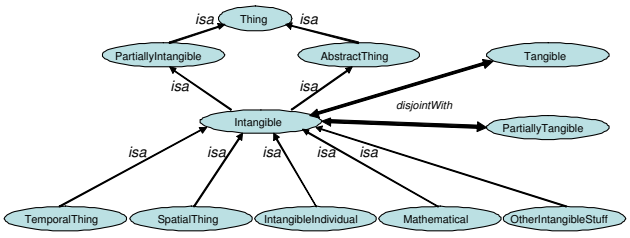


Fig. 2. Graphical Representation for $Center_1$

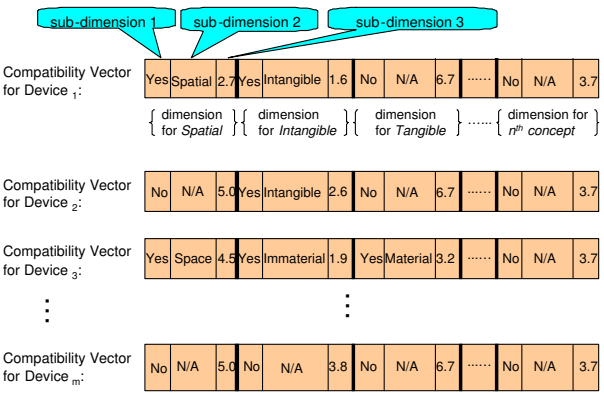


Fig. 3. Compatibility Vectors

the ontology distance between the original ontology and the center one: $D = \sum_{i=1}^{n} (w_i \cdot d_i)$, where d_i is the distance between a pair of concepts, and n is the number of concepts in the center ontology.

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 one. 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.

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 domains, each corresponding to one concept in the center ontology. Therefore, all compatibility vectors have identical dimension, 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; and 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.

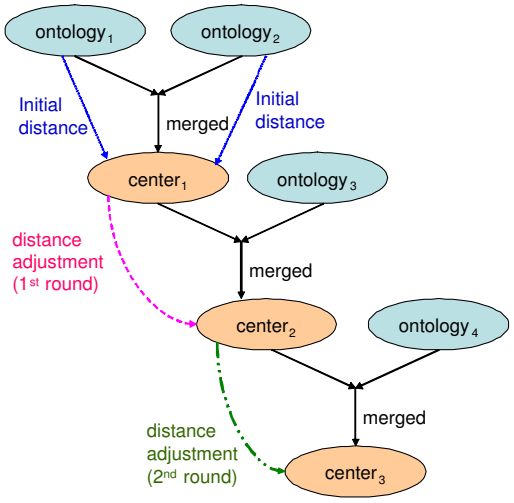


Fig. 4. Dynamic Adjustment of Compatibility Vectors

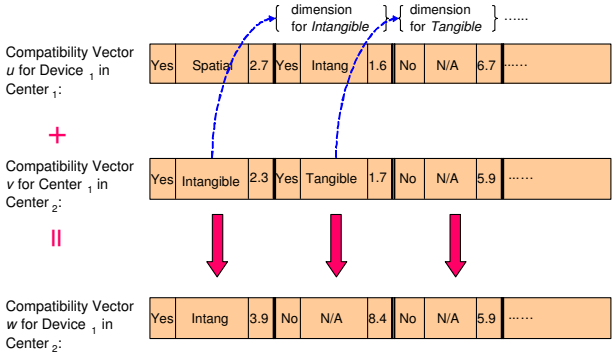


Fig. 5. Example of New Vector Generation

For the first concept ("Spatial") in the center ontology, $device_1$ knows it as "Spatial" and has a concept distance of 2.7; $device_3$ also understands this concept, but with a different name ("Space") and a bigger concept distance of 4.5; neither $device_2$ nor $device_m$ recognizes concept "Spatial", therefore, they have the same concept distance (5.0).

C. Dynamic Adjustment of Compatibility Vectors

As mentioned before, when there is only one mobile device, 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 devices into this MSC, the compatibilities for existing devices might be changed because newly joined devices could contain more accurate and/or complete information regarding the ontology in the same domain.

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 compatibility vectors of $center_1$. Upon the joining of $ontology_3$ and the generation of $center_2$, the distance from $center_1$ to $center_2$ is calculated

and then integrated into the compatibility vectors in $center_2$ for $ontology_1$ and $ontology_2$. This is accomplished by vector integration.

For example, we have compatibility vectors in both $center_1$ and $center_2$. Now we want to upgrade 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 stay 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 mobile device 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 $device_i$ in $center_1$

Output:

- compatibility vector *w* for $device_i$ 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(nlogn)*. There are *n* dimensions in each vector, therefore, we need *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 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*.

D. Ontology Understanding via Compatibility Vectors

The center ontology maintains the compatibility vectors for all original devices; in addition, the vectors themselves contain such information as whether a device understands a specific concept or not, what is the concept name in the original ontology, and so on. Therefore, if two devices 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 a mobile device in the mutual understanding of ontological concepts.

E. Mobile Device(s) Selection through Compatibility Vectors

When a mobile device from outside this MSC needs to ask for service(s), it would like to choose the device(s) understanding its ontology best. The device first compares its own ontology with the center ontology, and then searches in the compatibility vectors to find all those devices understanding the concept of its interest. If there is more than one device competing to provide this service, the request will be sent to those with good compatibilities, that is, devices with concept and/or ontology distance below a certain threshold. Such a threshold could be either specified by the service-consuming device, or otherwise determined by the center ontology. Because the compatibility vectors are stored and maintained by the center ontology, the service-rendering devices have no way to modify or manipulate the vectors. In this sense, the selection of service device(s) is objective and with no bias.

F. Features of Compatibility Vectors

1) Correctness of Compatibility Vectors - A Precise Approach: In this section we prove that our approach obtains a correct compatibility for each mobile device.

To record and maintain the proper compatibility of each device inside a MSC, 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 device and its associated ontology join the MSC, instead of communicating with each existing device, it only talks with the center ontology. Therefore, if we can prove that the newly merged ontology is a correct new center ontology, the correctness of compatibility vectors is guaranteed.

First, we point out that according to the merging algorithm in this paper, 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 IV, 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 IV, "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 IV.

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 we 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 we will get the same result.

2) Complexity of Compatibility Vectors - An Efficient Approach: The time complexity of establishing a MSC, along with the achievement of mutual understanding of ontological concepts, is in the order of $O(mn^2)$. 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 IV. In addition, in order to dynamically update the compatibility vectors, extra time will be spent. According to the previous analysis, $O(nlogn)$ is needed for updating one device, so the time for extra work for all services is $O(mnlogn)$. Therefore, the total complexity becomes $O(m(n^2 + nlogn))$, which is in the same order of $O(mn^2)$.

For device selection, the time complexity is $O(n^2)$. We only need to compare the ontology from the service-consuming device with the center ontology.

VI. EXPERIMENTAL RESULTS

A. Test ontologies

A collection of sixteen ontologies for the domain of "Building" were constructed by graduate students in computer

TABLE I
STATISTIC OF TEST ONTOLOGIES

	Average of Original Ontologies	Merged Ontology
Max Depth	7	8
# of Total Nodes	14	64
# of Inner Nodes	6	42
# of Total Nodes	5	47
# of Inner Nodes	23	182

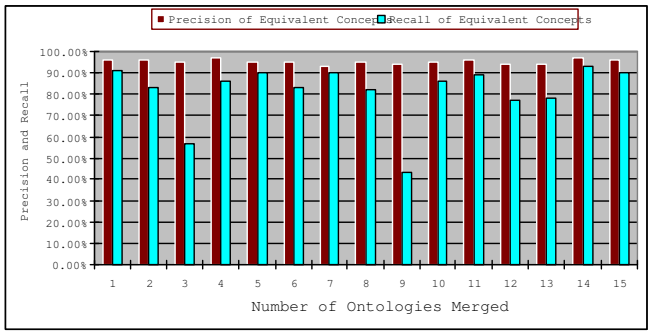


Fig. 6. Precision and Recall Result in Ontology Merging

science at the University of South Carolina and used for evaluating the performance of our merging algorithm, and the utilities from compatibility vectors as well. Table I lists the summarized characteristics of these test ontologies. They have between 10 and 15 concepts with 19 to 38 properties.

B. Experiments on Merging Algorithm Itself

To decide whether a correctly merged ontology is obtained, we asked two ontology experts to carry out a manual mapping and we compared their results with ours. Both precision and recall measurements are applied in the evaluation. The evaluation result is shown in Figure 6, reflecting a promising result. Please refer to [1] for more details.

C. Experiments on Compatibility Vectors

1) Correctness of Compatibility Vectors: We simulated a MSC out of 16 test ontologies. Based on ontology distances calculated (see Figure 7), we sorted the original ontologies with regard to their distances to the center. We then asked two experts to rank the qualities of these ontologies manually; the result is the same as the one from our system.

2) Efficiency of Compatibility Vectors: A set of experiments have been conducted. We first fixed one original ontology as the service-consuming one, and simulated a MSC out of the remaining 5, 10, and 15 ontologies as three experiment settings; then for each MSC setting we did the following in two groups. In the first group the service-consuming 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 8. It is clear that, after adopting our compatibility vectors,

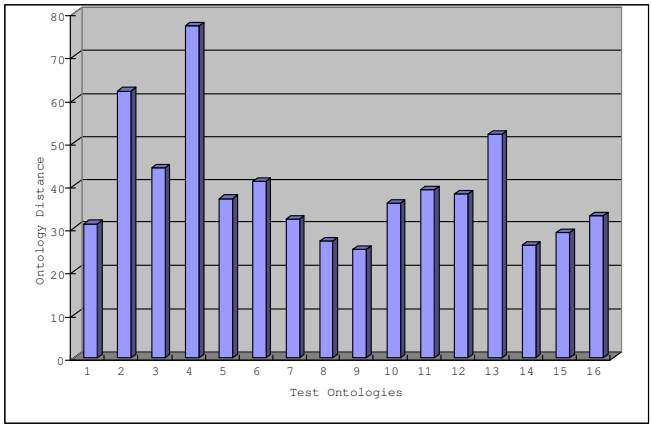


Fig. 7. Ontology Distance Calculated

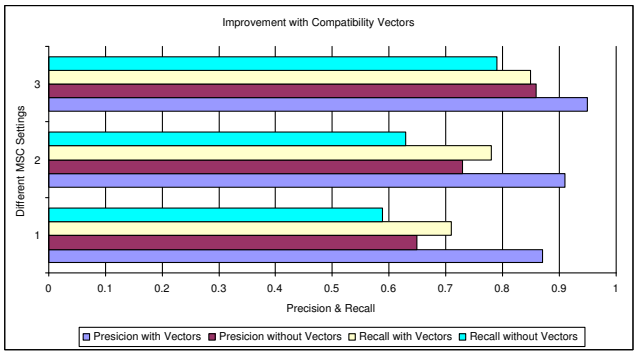


Fig. 8. Improvement with Compatibility Vectors

both precision and recall measurements have been improved. Therefore, in cases where sufficient resources are not available and only a certain number of mobile devices can be chosen for interaction, our approach increases the efficiency by choosing suitable mobile device(s).

VII. CONCLUSION

Mobile computing has become increasingly important with the proliferation of mobile devices. To extend the usually limited capabilities of typical mobile devices, it is essential to integrate mobile services from different providers. As the first step of this process, mobile devices must understand each other's service descriptions. Although ontologies can aid in this understanding, they will likely have heterogeneous semantics if designed independently, as they typically are. We present an automated approach carried out at the schema level to reconcile ontologies as a basis for mobile service integration and invocation. In addition, we introduce compatibility vectors as a method to evaluate and maintain ontology qualities, thereby handling the problem of how to choose ontologies with good compatibilities. We not only prove theoretically that our approach is both precise and efficient, but also show promising results experimentally.

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. To analyze and solve this problem is a potential research direction. Other

future work includes (1) how to maintain compatibility vectors when existing mobile devices modify their corresponding ontologies, and (2) what kind of mechanism is suitable if we simultaneously consider qualities of both ontologies and services.

REFERENCES

[1] J. Huang, R. Zavala Gutiérrez, B. Mendoza, and M. N. Huhns. *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] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. *Learning to match ontologies on the Semantic Web*, in: *The VLDB Journal*, Vol. 12. Springer-Verlag 303–319, 2003.

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

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

[5] N. F. Noy, M. A. Musen. *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] S. Melnik, H. Garcia-Molina, and E. Rahm. 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.

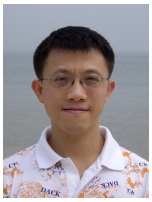
[7] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. *S-Match: an algorithm and an implementation of semantic matching*, in: *Proceedings of the 1st European Semantic Web Symposium*, Vol. 3053. Springer-Verlag 61–75, 2004.

[8] JWNL, "Java WordNet Library - JWNL 1.3", http://sourceforge.net/projects/jwordnet/, 2005.

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

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

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



Jingshan Huang is a Ph.D. candidate in Computer Science and Engineering Department at the University of South Carolina. Mr. Huang is a member of IEEE, AAAI, SIAM, and a review board member of Journal of Open Research on Information Systems (JORIS). He has served as a program committee member for several international conferences and is a technical paper reviewer for many journals and conferences. Mr. Huang's research interests include ontology matching/aligning, ontology quality, semantic integration, Web services, and service-oriented computing. He can be reached at huang27@sc.edu.

Jiangbo Dang earned his Ph.D. degree in computer science from Computer Science and Engineering Department at the University of South Carolina. Dr. Dang's research interests include distributed artificial intelligence and multiagent systems, service-oriented computing, business process and workflow management, and knowledge discovery, data mining, and machine learning. He can be reached at dangj@engr.sc.edu.

Michael N. Huhns is a professor of Computer Science and Engineering Department at the University of South Carolina, where he also directs the Center for Information Technology. Dr. Huhns is a Fellow of the IEEE and a member of Sigma Xi, Tau Beta Pi, Eta Kappa Nu, ACM, Upsilon Pi Epsilon, and AAAI. He is the author of over 200 technical papers and four books in machine intelligence, including the recently coauthored (with Prof. Munindar P. Singh of NCSU) textbook *Service-Oriented Computing: Semantics, Processes, Agents* for John Wiley Publishing Co. Dr. Huhns is on the editorial boards of seven journals and is a founder and board member for the International Foundation for Cooperative Information Systems and the International Foundation for Multiagent Systems. Hi sresearch interests include multiagent systems, service-orinted computing, and semantic Web services. He can be reached at huhns@sc.edu.

Yongzhen Shao is pursuing a Master's degree in the Institute of E-Business of Software School at Fudan University, China. Mr. Shao's research interests include E-Business, Web services, and ontology management. He can be reached via email: shaoyz@fudan.edu.cn.