Ontology Mapping in Pervasive Computing Environment[1]

C. Y. Kong, C. L. Wang, F. C. M. Lau

Department of Computer Science
The University of Hong Kong
{cykong, clwang, fcmlau}@cs.hku.hk

Abstract. Ontology provides a formal, explicit specification of a shared conceptualization of a domain. It enables knowledge sharing in open and dynamic distributed systems. Using ontology, devices can understand the messages without prior knowledge about the format or content of the messages. It also allows devices not expressly designed to work together to interoperate. In this paper, we propose an online ontology mapping mechanism for facing up to new challenges in ontology mapping in pervasive computing environment. Our proposed design takes similarities of the names, properties and relationships of concepts into consideration during mapping. It outperforms the previous *source-based* and *instance-based* approaches in terms of efficiency as it does not require finding a one-to-one corresponding mapping of concepts between two ontologies. It can also use history records to store the information about the instances instead of storing all the instances which is more space efficient than traditional instance-based ontology mapping.

1 Introduction

Pervasive computing environment is an environment saturated with computing and communication capability, yet so gracefully integrated with users that it becomes a "technology that disappears" [1]. With the prevalence of mobile network technologies, users can move from place to place to get their tasks done. They need computing services in all places where a different knowledge domain may be embodied in each smart space. The ad hoc, spontaneous, and dynamic nature of the pervasive computing environment would require the systems to provide instant knowledge reasoning for efficiently mapping between user queries and domain knowledge.

Ontology has been introduced for bridging the knowledge gaps between different domains [11]. Ontology represents the semantics of different concepts. It provides a formal, explicit specification of a shared conceptualization of a domain that can be communicated between people and heterogeneous and widely spread application systems [2]. It is a formal explicit description of concepts (also called *classes*) in a

[1] This research is partly supported by HKU Large Equipment Grant 01021001 and Hong Kong RGC Grant HKU-7519/03E.

domain of discourse, properties of each concept describing various features and attributes of the concept (also called *slot*) and restrictions on concepts [3]. Using ontology, devices can understand the messages without prior knowledge about the format or the content of the messages. This enables knowledge sharing in open and dynamic distributed systems. It also allows devices and agents not expressly designed to work together to interoperate. Ontology and its data instances are separated, therefore providing a means for intelligent agents to reason about contextual information.

Ontology techniques have been adopted in several pervasive computing projects such as CoBrA [4] and GAIA [5]. Context Broker Architecture (CoBrA) defines a set of OWL ontologies for context information. Agents use these ontologies to acquire, reason about and share context knowledge. GAIA is a middleware to enable active spaces. It also defines a set of ontologies about the active spaces such as entity and context information. Devices in GAIA use these ontologies to communicate with each other. In both cases, ontologies were used in a very restricted way. Ontologies for their applications are defined in the stationary environment. As thus, devices could only communicate using the same ontologies. Researchers in pervasive computing have put their efforts in defining different ontologies used in their projects [4,5]. Ontology mapping has also been widely researched in semantic web [6-10].

We identify four new challenges for ontology mapping in pervasive computing environment: (1) online mapping (when two devices are communicating), (2) efficiency in mapping, (**3**) space limitation of devices, and (4) knowledge propagation to support user mobility.

Existing ontology mapping tools are used to help ontology designers to design new ontologies. They provide a merged ontology or a set of related concepts with some certainty factor associated with each pair. In pervasive computing environment, a device may communicate with many different devices. It is difficult for the ontology designers to merge all the possible ontologies with which the device is going to communicate because it is hard to predict such communications. Even if the ontology designers can merge the possible ontologies, the merged ontology cannot reflect any modifications in the original ontologies unless the ontology designers also change the merged ontology. In order to avoid merging all possible ontologies and to handle the ontology versioning problem, ontology mapping should be done at the time when two devices communicate (called *online ontology mapping*). Efficiency and space limitation are important factors for designing online ontology mapping. It is not possible for memory limited devices such as handheld and wearable devices to store a set of data instances of the ontologies that they use for communication. High user mobility in pervasive computing environment enables sharing of users' communication history. Knowledge sharing of history records can help devices to communicate in different places.

In this paper, we propose an online ontology mapping mechanism to tackle these new challenges in pervasive computing environment. This paper is structured in 5 sections. The next section is about the related research. Section 3 introduces the philosophy of our mapping mechanism. Section 4 gives the evaluation result and Section 5 is the conclusion and presents the future work.

2 Related Research

Ontology mapping has been widely researched in semantic web. Most ontology mapping tools developed are to find a one-to-one corresponding mapping between concepts in two ontologies [6-10]. These mapping tools can be classified into two types: *source-based* and *instance-based*.

Source-based mapping tools compare the similarity of the concepts based on the properties of the concepts and the structure of the ontology defined in the source ontologies. Examples of source based mapping tools are PROMPT [6], Chimaera [7], and ONION [8]. PROMPT and Chimaera merge two source ontologies into a new ontology that includes concepts from both sources. They compare similarity of concept names to generate a match list of concepts. Users decide which concepts should be mapped based on the match list. ONION results in a set of mappings (articulation rules using their terms) between two ontologies. It transforms source ontologies into graphs. The nodes and the edges of the graph are used to match two graphs. Nodes are matched based on their names and a set of user-defined synonyms words. PROMPT, Chimaera and ONION use similarity between concept names for mapping. They work well for ontologies having a specialized terminology like medical ontology where each concept is a disease and each disease has a unique name. Their matching accuracy decreases when mapping ontologies with more general terminologies.

Instance-based ontology mapping tools compare the similarity of the concepts based on the source ontologies and their data instances. Examples of instance-based ontology mapping tools are FCA-Merge [9] and GLUE [10]. FCA-Merge merges two source ontologies into a new ontology. FCA-Merge generates a pruned concept lattice by analyzing the frequencies of usage of concepts. Merging decisions are made based on the pruned concept lattice. FCA-Merge suits best the mapping of text documents**:** it requires a set of common instances for the mapping ontologies. For example, the instances are in the form of documents or homepages. GLUE gives a set of pairs of related concepts with some certainty factor associated with each pair. It analyzes the distributions of the concepts in data instances of the source ontologies and uses joint probability distribution to calculate the similarity between two concepts. GLUE, however, does not consider the structure of the ontologies (i.e., the relationships between concepts) during mapping.

Our proposed design makes use of both the source ontologies and their instances. It takes the similarities of the concept names, properties of the concepts and their relationships into consideration during mapping. It works for many different types of ontologies since it can assign a different weighting to each similarity. Compared with those mapping tools that find a one-to-one corresponding mapping of concepts between two ontologies, our mechanism is more efficient since it is not one-to-one mapping. Compared with traditional instance-based ontology mapping tools, our mechanism is more space efficient since it uses history records to store the information about the instances instead of storing all the instances.

3 Proposed Design

In our proposed design, we assume a proxy exists in every smart place, where a smart space is an environment saturated with users and devices such as a meeting room, a department office or a university campus. A proxy coordinates all the resources and functions of devices in its smart space. It helps devices to allocate resources and functions and to obtain contextual information of the smart space. As mentioned in the pervious section, online ontology mapping needs to be reasonably efficient, which can be achieved by reducing the number of concepts to be mapped between two ontologies. A device submits a request which is an instance I_1 of an ontology O_1. The proxy tries to find a function in the smart place that satisfies the request where the function is described by ontology O_2. Our proposed matching mechanism maps all the concepts used in I_1 to O_2 instead of finding a one-to-one corresponding mapping for all the concepts in O_1 and those in O_2 as is done in traditional semantic web ontology mapping.

In this paper, O_1 denotes the ontology used by the device requesting resources or functions. O_2 denotes the ontology that describes the resources or functions with which the proxy would try to match the request. I_n is an instance of ontology O_n and C_n is a concept in ontology O_n. w_i is a weight. Sim(A,B) is the similarity between item A and item B.

3.1 Extraction of concepts to be compared with concept C_1

To compare two ontologies O_1 and O_2, we have to map a concept C_1 in O_1 to a concept C_2 in O_2. To increase the efficiency of mapping, we filter the highly related concepts and generate a set of possible candidates of C_2 in O_2. For each pair of concepts between C (in O_1) and C' (in O_2), we calculate the similarity of their concept names where N(string) is the number of characters in the string.

$$Sim\left(C\ name,\ C'\ name\right) = \frac{N\left(longest\ substring\right)}{N\left(C\ name\right)+N\left(C'name\right)}$$

For the first k concepts with the highest similarity degree denoted by $C_{1..k}$, we find the possible candidates set to be compared with C by:

$$Possible\ candidate\ set = \left\{ \begin{array}{l} C_{1..k} \cup \\ concepts\ that\ has\ relationship\ with\ C_{1..k} \cup \\ merged\ concepts\ of\ C_{1..k}\ with\ each\ of\ its\ neighbor \cup \\ parent\ (\sup er\ class)\ concepts\ of\ C_{1..k} \cup \\ chidren\ (sub\text{-}class)\ concepts\ of\ C_{1..k} \end{array} \right\}$$

In ontology mapping, a concept in O_1 may be split into two concepts in the same way as a concept "name" being split into two concepts, "first name" and "last name". To handle the splitting problem, our proposed mechanism merges concepts with their neighborhood concepts. Merging concepts C'_1 and C'_2 of the same ontology is done by merging their concept names, attributes and relationships. To resolve naming

conflict of attributes and relationships, attributes and relationships are renamed as C'$_1$.attribute name and C'$_2$.attribute name and C'$_1$.relationship name and C'$_2$.relationship name respectively. Duplicated relationships are removed during merging. A relationship between C'$_1$ and C'$_2$ is converted as attribute with the name of the relationship as the attribute name.

3.2 Comparison between Two Concepts C_1 and C_2

For each candidate in the candidates set generated in the previous section, we compare the similarity between two concepts. In similarity research, similarity is commonly defined as:

$$\frac{P(C_1 \cap C_2)}{P(C_1 \cup C_2)} = \frac{P(C_1, C_2)}{P(C_1, C_2) + P(C_1, \sim C_2) + P(\sim C_1, C_2)} \quad \textbf{(1)}$$

From the project GLUE, $P(C_1, C_2)$ is defined as equation (2) where U_1 and U_2 are the instance set of O_1 and O_2 respectively, N ($U_1^{C1,C2}$) is the number of instances of O_1 that contain concept C_1 and concept C_2, N ($U_2^{C1,C2}$) is the number of instances of O_2 that contain concept C_1 and concept C_2, N (U_1) and N (U_2) are the number of instances of O_1 and the number of instances of O_2 respectively.

$$P(C_1, C_2) = \frac{N(U_1^{C1,C2}) + N(U_2^{C1,C2})}{N(U_1) + N(U_2)} \quad \textbf{(2)}$$

In our proposed matching mechanism, we use these formulae when comparing between concepts C_1 and C_2. To calculate $P(C_1, C_2)$, we should have two instance sets U_1 and U_2. As discussed in the above section, it is not space efficient for the devices and the proxies to store these sets of instances. Instead, we use history records to determine the instance sets U_1 and U_2. The proxy counts the number of concepts appearing in each mapping instance. For example, an ontology O_A contains concepts C_a, C_b, C_c and C_d. A request instance contains concepts C_a and C_b. The number of instances that contain C_a and C_b are incremented by 1, the number of instances that contain C_c and C_d remains unchanged, and the total number of instances of O_A is also incremented by 1. $N(U_1)$ and $N(U_2)$, therefore, are found. In pervasive computing environment, we assume that there exists at least one instance from each ontology. One is the request instance and the other is the resource/function instance. To get $N(U_1^{C1,C2})$, we have to estimate the number of instances of O_1 that contain concept C_1 and concept C_2. The number of instances of O_1 that contain concept C_1 can be found from the history records. We estimate the number of concepts that contain both concept C_1 and concept C_2 by calculating the similarity degree of the properties and relationships of concept C_1 and concept C_2. If the properties and relationships of the concepts are similar, it is likely that the instance of concept C_1 is an instance of concept C_2. For property such as memory size which is a numerical type, it is important that there is mapping to another concept whose property contains also a numerical value; otherwise, it is difficult to satisfy functionality requests like "memory size less than 10k". Weights, therefore, are needed when calculating

property similarity. Besides, using properties and relationships of the concepts, the proposed mechanism also matches instances when comparing two concepts. When a concept of the request instance matches a concept of the resource or function instance, it is likely that two concepts are matched. The proxy uses all the present instances of O_1 and O_2 in its smart space to compare with the request instance.

For each pair of instance sets U_1 and U_2,

1. Partition U_1 into two sets. One set contains concept C_1 (U_1^{C1}) while the other set does not contain concept C_1 ($U_1^{\sim C1}$) based on the history record.
2. Partition U_2 into two sets. One set contains concept C_2 (U_2^{C2}) while the other set does not contain concept C_2 ($U_2^{\sim C2}$) based on the history record.
3. Estimate the similarity between O_1 and O_2 with equation (3) where $N(O_1)$ and $N(O_2)$ are the total numbers of concepts in O_1 and O_2 respectively. For each concept in O_1, find the maximum concept name similarity with O_2 based on result calculated in the previous section. If the maximum concept name similarity is larger than the threshold, increment the total number of similar concepts.

$$Sim(O_1, O_2) = \frac{total\ number\ of\ similar\ concepts}{N(O_1) + N(O_2)} \tag{3}$$

4. Find N ($U_1^{C1,C2}$).

$$Number\ of\ instance\ in\ U_1\ that\ contains\ concept\ C_2 = Equation(3) * N\left(U_2^{C2}\right) \tag{4}$$

For each pair of property/attribute in C_1 (denoted by P_{C1}) and property/attribute in C_2 (denoted by P_{C2}), compute their property similarity using equation (5).

$$\begin{aligned} &Sim(P_{C1}, P_{C2}) \\ &= w_1 * Sim(P_{C1}name, P_{C2}name) + w_2 * Sim(P_{C1}cardinality, P_{C2}cardinality) + \\ &\quad w_3 * Sim(P_{C1}data\ type,\ P_{C2}\ data\ type) + w_4 * Equation(6) \end{aligned} \tag{5}$$

Property instance similarity is calculated by counting the number of instances of C_2 whose property has similar content as the corresponding property of the instances of C_1. The property instance similarity is calculated when the properties are using text description.

$$\begin{aligned} &Sim(content\ of\ instance\ 1,\ content\ of\ instance2) \\ &= \frac{N(longest\ substring)}{N(property\ instance\ of\ O_1) + N(property\ instance\ of\ O_2)} \end{aligned} \tag{6}$$

$$\begin{aligned} &Property\ similarity,\ ps = \prod frequency\ of\ properity\ i\ in\ C_2\ * Equation(5) \\ &for\ i = 1\ to\ number\ of\ property\ in\ C_2 \end{aligned} \tag{7}$$

$$N\left(U_1^{C1,C2}\right) = N\left(U_1^{C1}\right) * ps = N\left(U_1^{C1}\right) * Equation(7)$$

$$N\left(U_1^{C1,\sim C2}\right) = N\left(U_1^{C1}\right) - N\left(U_1^{C1,C2}\right)$$

$$N\left(U_1^{\sim C1,C2}\right) = Equation(4) - N\left(U_1^{C1,C2}\right)$$

5. Similarly, calculate $N(U_2^{C1,C2})$, $N(U_2^{C1,\sim C2})$ and $N(U_2^{\sim C1,C2})$.
6. Compute $P(C_1,C_2)$, $P(C_1,\sim C_2)$ and $P(\sim C_1,C_2)$ using equation (2).
7. Compute the similarity degree using equation (1). This similarity degree is called *instance similarity degree* as we use instances to calculate.
8. For each relationship between C_1 and C_2, compute similarity between relationship R_{C1} and R_{C2}.

$$Sim(R_{C1}, R_{C2}) = w_1 * Sim(R_{C1}name, R_{C2}name) + w_2 * Sim(R_{C1}cardinality, R_{C2}cardinality) + w_3 * Sim(R_{C1}type, R_{C2}type)$$

9. Calculate *Relationship Similarity Degree*.

$$\text{Relationship similarity degree} = \prod \text{similarity of relationship } i \text{ in concept } C_1 \text{ and } C_2 \text{ for } i = 1 \text{ to number of relationships in } C_2$$

10. Calculate the similarity between C_1 and C_2.

$$Sim(C_1, C_2) = w_1 * \text{instance similarity degree} + w_2 * \text{relationship similarity degree}$$

3.3 Comparison between Two Ontologies O_1 and O_2

Below is the methodology to compare two ontologies; OntologyMapping() is our mapping function and NewMapping() is a procedure call that is invoked when mapping is performed from scratch.

```
NewMapping()
{
   Extract the candidate concepts as Section 3.1.
   For each candidate found in section 3.1
      Computer P(C1 ∩ C2) / P(C1 ∪ C2) as Section 3.2
   If the highest similarity degree > threshold,
      Mapping is found.
   Else
      Mapping is failed.
}
```

```
Ontology Mapping (Ontology O1, Ontology O2)
{
   Search history mapping record.
   If O1 and O2 have been mapped,
      If O1 and O2 have the same last modified date
         (version number) as the history record,
         For each concept Ci in the instance,
            If O2 is mapped to Ci in the record,
               Mapping is found.
            Else
               Invoke NewMapping().
      Else
         For each concept Ci in the instance,
            If O2 is mapped to Ci in the record,
               Compute P(C1 ∩ C2) / P(C1 ∪ C2) as Section 3.2 for
               Ci and the mapped concept in O2.
               If similarity degree > threshold,
                  Existing mapping is reused.
               Else
                  Invoke NewMapping().
            Else
                  Invoke NewMapping().
   Else
      Invoke NewMapping().
   Update number of instances and concepts
   encountered.
   For each new mapping found,
      Add <concept in O1, concept in O2, similarity
      degree,instance count> in history record.
}
```

4 Evaluation

This section gives the implementation details and the evaluation of the proposed online ontology mapping mechanism. The matching mechanism has been implemented using Java language. It takes two ontologies, the instances of the two ontologies and the request instance as inputs. A configuration file is used by users to define the weights and thresholds. The concepts presented in the request instance are matched with the second ontology. The first ontology defines the concepts used in the request instance. A matching table with the concepts in the request instance and the concepts in the second ontology are outputted. We used Semantic Web Research Community (SWRC) ontology [12] as the first ontology for our mapping experiment. Based on the SWRC ontology, we manually created the second ontology, the

instances of the two ontologies and the request instances. The accuracy of the mapping is defined as the percentage of the mappings from using our matching mechanism that match the manual mappings.

We have divided the evaluation into two parts. The first experiment is to compare our online ontology mapping mechanism with the source based mapping mechanism. We manually created the second ontology with the concepts and their properties having names similar to the SWRC ontology. Source based mapping mechanism has an accuracy of over 90% while our matching mechanism has an accuracy of about 80%. Our matching mechanism is less accurate due to some of the instances having totally different content, for example, the information about a person. However, our matching mechanism is faster than the source based mapping mechanism. The source based mapping mechanism uses 10 seconds to find a one-to-one corresponding mapping between the concepts in the two ontologies. If we do not take instances into consideration in our matching mechanism (by setting the weights of similarity of instance to zero), our solution needs just 6 seconds to map the request instance with the second ontology, where the request instance contains 6-8 concepts out of 24 concepts in the SWRC ontology. If we take instances into consideration, it uses about 20 seconds to map the request instance with the second ontology.

The second experiment is to compare our online ontology mapping mechanism with the instance-based ontology mapping mechanism. We manually created similar content of the instances of the two source ontologies with most of the names of the concepts and the attributes being different. We found that our matching mechanism is much faster than instance-based ontology mapping with similar accuracy (about 70%). We also created content of the instances that are highly different. Our match mechanism has about 40% accuracy which is two times more accurate than the instance-based ontology mapping mechanism as we have also considered the names of the concepts, attributes and the relationships of the concept.

Our matching mechanism can perform faster than source based ontology mapping, while achieving the same level of accuracy. The accuracy of our matching mechanism for the first experiment can be further improved (>85%) by increasing the weightings of the similarity of the names of the concepts and the attributes. Our matching mechanism is more accurate and more efficient than instance based ontology mapping. Our solution can achieve better space efficiency. Based on the experiments we have done, we use about 2K memory to store 10 instances of a concept. Instance based ontology mapping needs to use at least 30 instances in order to achieve a high accuracy [10]. That means instance-based ontology mapping should use at least 144K for storing 30 instances for each concept in the SWRC ontology. Our matching mechanism does not require storing the instances. We use a history record that stores the total number of instances and concepts that have been mapped, which takes up only a few K's of memory.

5 Conclusion and Future Work

In this paper, we have identified four challenges of ontology mapping in pervasive computing environment. They are online mapping, efficiency, space limitation and

knowledge propagation. We have proposed an online mapping mechanism to respond to these challenges. The proposed mechanism takes similarities of concept names, and properties of concepts and their relationships into consideration during mapping. It attaches a weighting to each similarity to suit different types of ontologies. The mechanism does not require finding a one-to-one corresponding mapping of concepts between two ontologies, which helps increase the efficiency. To be space efficient, it uses history records to store the information about the instances instead of storing all the instances. The history records can be propagated to other places for knowledge propagation. More experiments are required to prove the space efficiency for our matching mechanism.

References

1. M. Satyanarayanan. *Pervasive computing: Vision and Challenges*. IEEE Personal Communications, p.10-17. August, 2001.
2. T. R. Gruber. *A translation approach to portable ontology specifications.* Knowledge Aceuisition 5, p.199-220.
3. N. F. Noy and D. L. McGunness. *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Standard Medical Informatics Technical Report SMI-2001-0880. March, 2001.
4. H. Chen, T. Finin and A. Joshi. *Using OWL in a Pervasive Computing Broker.* In Proceedings of Workshop on Ontologies in Agents Systems, held in conjunction with the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems. July, 2003.
5. M. Roman, C.Hess, R. Cerqueira, A. Ranganathan, RH Campbell, K. Nahrstedt. *Gaia: A Middleware Infrastructure for Active Spaces.* IEEE Pervasive Computing Vol. 1, No. 4, p. 74-83. October – December, 2002.
6. N. F. Noy and M. A. Musen. *PROMPT: Algorithm and tool for automated ontology merging and alignment.* In 17th National Conferences on Artificial Intelligence (AAAI-2000). 2000.
7. D. L. McGuinness, R. Fikes, J. Rice and S. Wilder. *An environment for merging and testing large ontologies*. In A. G. Cohn, F. Giunchiglia and B. Selman, editors, Principles of Knowledge Representation and Reasoning: Proceedings of the 17th International Conference (KR2000). 2000.
8. P. Mitra, G. Wiederhold and M. Kersten. *A graph-oriented model for articulation of ontology interdependencies.* In Proceedings Conferences on Extending Database Technology 2000 (EDBT'2000). 2000.
9. G. Stumme and A. Mädche. *FCA-Merge: Bottom-up merging of ontologies.* In 7th International Conference on Artificial Intelligence (IJCAI'01), p. 225-230. 2001.
10. Doan, J. Madhavan, P. Domingos and A. Halevy. *Learning to map between ontologies on the semantic web.* In 11th International WWW Conference. 2002
11. Jeff Heflin. *Web Ontology Language (OWL) Use Cases and Requirements*. February, 2004. (http://www.w3.org/TR/2004/REC-webont-req-20040210/).
12. Web Research Community (SWRC) http://ontobroker.semanticweb.org/ontologies/swrc-onto-2001-12-11.daml/.