# Discovering Executable Semantic Mappings Between Ontologies

Han Qin, Dejing Dou, and Paea LePendu

Computer and Information Science
University of Oregon
Eugene, OR 97403, USA
{qinhan, dou, paea}@cs.uoregon.edu

**Abstract.** Creating *executable* semantic mappings is an important task for ontology-based information integration. Although it is argued that mapping tools may require interaction from humans (domain experts) for best accuracy, in general, automatic ontology mapping is an AI-Complete problem. Finding matchings (correspondences) between the concepts of two ontologies is the first step towards solving this problem but matchings are normally not directly executable for data exchange or query translation. This paper presents an systematic approach to combining ontology matching, object reconciliation and multi-relational data mining to find the executable mapping rules in a highly automatic manner. Our approach starts from an iterative process to search the matchings and do object reconciliation for the ontologies with data instances. Then the result of this iterative process is used for mining *frequent queries*. Finally the semantic mapping rules can be generated from the frequent queries. The results show our approach is highly automatic without losing much accuracy compared with human-specified mappings.

## 1 Introduction

The emergence of the Semantic Web has emphasized the need for systems that are able to query, integrate and exploit data from multiple, disparate sources which may use different ontologies, but are about the same domain. Research involving the Semantic Web is experiencing huge gains in standardization in that OWL becomes the W3C standard for ontological definitions in web documents. However, it is extremely unreasonable to expect that ontologies used for similar domains will be few in number [5]. As the amount of data collected in the fields of Biology and Medicine grows at an amazing rate, it has become increasingly important to model and integrate the data with ontologies that are biologically meaningful and that facilitate its computational analysis. Hence, efforts such as the *Gene Ontology* (*GO*) [16] in Biology and the *Unified Medical Language System* (*UMLS*) [19] in Medicine are being developed and have become fundamental to researchers working in those domains. However, different labs or organizations may still use different ontologies to describe their data.

Some ontology-based information integration systems have been developed to process queries and exchange data from data resources with different ontologies.

R. Meersman and Z. Tari et al. (Eds.): OTM 2007, Part I, LNCS 4803, pp. 832–849, 2007.
© Springer-Verlag Berlin Heidelberg 2007

A survey can be found in [32]. For example, the InfoSleuth [4] project provides an agent and ontology based infrastructure for information gathering and analysis in distributed environments such as the Web. In OBSERVER [23] the information sources are described by domain ontologies, and user queries are rewritten by using the inter-ontology matchings. In our own previous work, we used OntoEngine (an inference engine) and our expressive Web-PDDL ontology language to translate Semantic Web documents and answer queries [13] between ontologies. We enhanced these tools with the ability to handle relational databases in addition to Semantic Web documents in OntoGrate [11,12].

Ontology matching and mapping is the key step to enable ontology-based information integration. The goal of ontology matching is to generate correspondences between the concepts from different but related ontologies. In most cases, formal mapping rules with clear semantics need to be generated for information integration systems. The problem of finding matchings and mappings between information resources has been extensively studied by different research communities. To be clear, we distinguish between *matching* (correspondence) and *mapping*. Matching pairs related concepts. One matching example is "both property *phone* and property *work_at* in one ontology are matched to property *workphone* in another." However, mapping not only pairs concepts but also formally defines the relationships between them. For example, "for all Office which has a *phone* number, all People who *work_at* that office must have *workphone* as the same number" is a mapping between property *phone*, *work_at* and *workphone*.

Research in discovering and representing semantic mappings is still in very preliminary stages. Indeed, the ideal choice of the mapping language, one carefully balancing expressivity with scalability, is an open question. Current mapping languages, such as Datalog, F-Logic, DLR, KIF or LOOM, are more or less subsets of first order logic [31]. In this paper, we mainly use general first order logic syntax to represent mapping rules, such as the above mapping we mentioned:

$$\forall x, y, z, People(x) \wedge Office(y) \wedge String(z) \\ \wedge work\_at(x, y) \wedge phone(y, z) \rightarrow workphone(x, z)$$

Our OntoEngine takes these kinds of rules in Web-PDDL[1] for information integration. The rules also can be represented as Datalog, SWRL [1] or other logic languages which some tools can process. Therefore, we call them *executable* mappings. It is similar as that Clio [24,17] calls *operational* mappings for database schemas.

In general, automatic ontology mapping is an AI-Complete problem. Many challenges remain. For example, although it is argued that mapping tools may require interaction from domain experts for best accuracy [29], it is not clear what kind of interaction between the system and domain experts should be supported. In this paper, we propose to combine ontology matching, object

[1] Web-PDDL is also a subset of first order logic. We call mapping rules in Web-PDDL syntax bridging axioms in our previous work [13,11,12].

reconciliation and multi-relational data mining to find the executable mapping rules as automatically as possible. Our approach starts from an iterative process to search the matchings and do object reconciliation for the ontologies with data instances. Then the result of this iterative process could be used for mining *frequent queries*. Finally the semantic mapping rules can be generated from the frequent queries. The results show our approach is highly automatic without losing much accuracy compared with human-specified mappings.

The rest of the paper is organized as follows. We first give some background and related work in Section 2. Then we demonstrate our general framework for ontology mapping in Section 3. We elaborate our iterative approach for ontology matching and object reconciliation in Section 4. In Section 5, we show our extension to a well-known multi-relational data mining tool (i.e., FARMER [25]) for supporting mapping rules discovery. We show some promising results by case studies with real ontologies in Section 6. We conclude the paper and discuss the future directions in Section 7.

## 2 Related Work

In this section we first give background and more detail of some existing schema or ontology matching and mapping systems. We also introduce an object reconciliation system and several Multi-Relational Data Mining systems.

Not surprisingly, the database community was one of the first to invest considerable effort in developing systems that match different database schemas (see [29] for a survey). Most schema matching systems, such as LSD [8], CUPID [20], iMap [7] and COMA [14], focus on retrieving correspondences between attributes using a variety of similarity or correlation heuristics. Similarly, research in knowledge engineering and the Semantic Web has resulted in tools for ontology matching that are absolutely critical in ontology-based information integration. These tools also show promising applications in the database arena (see [26] for a survey). Chimaera [22], Protégé [27], GLUE [9] and MAFRA [21] are some examples of such systems. GLUE [9] employs machine learning and exploits data instances to find matchings between concepts. It uses domain knowledge and domain-independent constraints to increase matching accuracy. Chimaera [22] provides a ontology editor to allow user to merge ontologies. It suggests potential matchings based on the names of properties but needs users to verify them. The disadvantage of this system is that it leaves what to do entirely to users. Protégé [27] gives initial ontology alignments by plugging in one of existing matching algorithms and focuses on guiding users to refine alignments. This system updates its suggestions based on the input of user and gives new alignments to user. MAFRA [21] is an interactive, incremental and dynamic framework, which builds a semantic bridging ontology for distributed ontologies. These semantic bridges specify how to translate entities from source ontology to target ontology. BMO [18] can generate block matchings using a hierarchical bipartition algorithm. This system builds a virtual document for each ontology and compares each pair of concepts with the information in the virtual

document. This algorithm is efficient but it does not figure out how two blocks are matched, however these matching blocks still can be very helpful for finding mappings.

Clio [24,17] is a schema mapping system that can generate operational (i.e., executable) mappings in different formats such as SQL and XQuery for database integration. This system uses semantic information to find matchings first and allows the user to modify them or add new matchings. Then it produces mapping rules based on matchings. It is semi-automatic since it needs human interactions. The research by An, Borgida and Mylopoulos [3,2] provides very interesting methods to construct complex mapping rules between relational tables or XML data and ontologies when given an initial set of correspondences between the concepts in the schemas and ontologies. They offer the mapping formalisms to capture the semantics of XML or relational schemas by constructing the semantic trees from them. Their generated rules will be useful to domain experts for further refinement, as well as to applications. Our approach is to combine ontology matching, object reconciliation and multi-relational data mining to find executable mapping rules in a highly automatic manner.

The object reconciliation problem is studied for determining whether two different objects of data sets refer to the same real-world entity. Xing Dong et al [10] propose to reconcile the object references in three steps. The first step is constructing the dependency graph, which describes the relationships between object pairs. One object pair (i.e., a reconciliation decision) needs to be decided as reconciled or not. The next step is an iteratively re-computing process, which computes the similarity scores of reconciliation decisions. Since the similarity score of one reconciliation decision can both affect and be affected by the similarity score of its neighbors, the algorithm sets a fixed point to stop the process. Finally, transitive closure is computed for the final reconciliation results.

Multi-relational mining techniques are already applied in many research areas such as subgraph mining. FOIL [28] is a first-order learning system which can generate Horn rules for target predicates by examining both positive and negative examples. WARMER [6] is designed for frequent pattern mining in relational databases. This system is built on the foundations of Inductive Logic Programming (ILP) and does not need negative examples. FARMER [25] is also a frequent pattern mining system, it takes object identity as the starting point and introduces several optimizations, and thus it has better performance than WARMER. In this paper, we will borrow some ideas from FARMER for generating ontology mapping rules. We will give more detail of our algorithm in Section 5.

## 3 Framework

In this section, we first introduce our general framework for mapping discovery, then we illustrate our idea with some simple examples. Given a source ontology and a target ontology which model the same domain, ontology matching can

find that some of their concepts (e.g., classes and properties in OWL ontologies) are matched to each other. Object reconciliation can find that some instances from both ontologies represent the same real world entities. Our final goal is to generate executable mapping rules based on that information.

### 3.1 System Architecture

Figure 1 shows the architecture of our system. There are five main components.

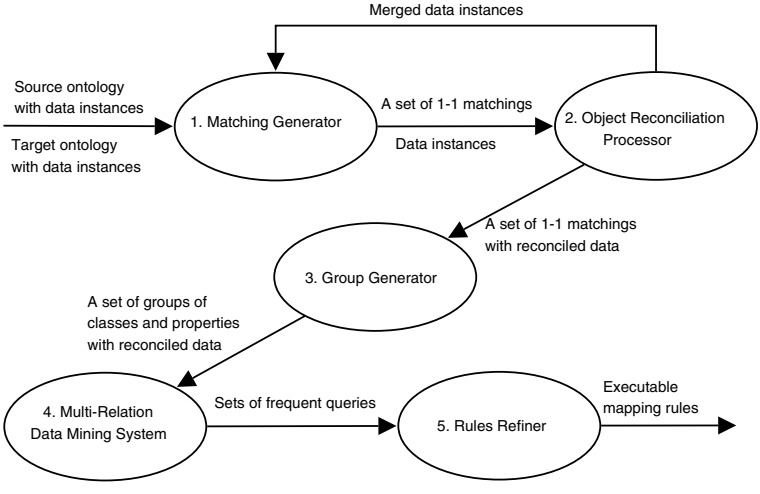


**Fig. 1.** System Architecture

1. Matching Generator (MG): It takes the source and target ontologies with their data instances as input. Different matching system could be plugged in this component. The output of MG is 1-1 matching pairs between classes and properties. If no new matchings can be found, MG will pass this information to the next component, Object Reconciliation Processor. It also passes the data instances.
2. Object Reconciliation Processor (ORP): This component is designed to reconcile instances which refer to the same real world entities. If there are new matchings from Matching Generator, ORP will try to reconcile more instances. Otherwise it passes the 1-1 matchings and the reconciled data to the next component, Group Generator.
3. Group Generator (GG): The matchings are not always 1-1 matchings. GG combines close related 1-1 property matching pairs, class matching pairs and their instances together as a group. For every group, GG generates a set of input data for multi relation data mining system.
4. Multi-Relation Data Mining System (MRDMS): We borrow some ideas of FARMER system to mine *frequent queries*. This system requires certain input format and it can find interesting frequent queries. Since every group generated by GG is comparably small, the search space is not a concern.

5. Rules Refiner (RR): Generating executable mapping rules from frequent queries is a natural extension. However, it is not suitable to set a fixed threshold for support and confidence. The threshold can be different for different cases. RR will filter out the rules which are distinctly incorrect and keep the rest.

### 3.2 Approach Overview with a Simple Example

We give the overview of our approach with a simple mapping example based on the People Ontology[2] from UMD and the Person & Employee Ontology[3] from CMU. And we also use these two ontologies to illustrate our system in the following sections.

The first step of our approach is to find corresponding classes and properties. Currently we use a string matching algorithm [30] to get 1-1 matchings. For example, two properties $name(Person, String)$ (from the People Ontology) and $name\_person(Person, String)$ (from the Person& Employee Ontology) have similar names and are considered matched. Before finding (mining) the semantic mapping of these two concepts, there is one problem that must be solved. Instances from different ontologies may represent the same object but have different names. For example, both of the instance "person001" described by the People Ontology and the instance "p001" described by the Person& Employee Ontology may actually refer the same person (e.g., `Han Qin`). Therefore we have to reconcile them by renaming "p001" to "person001" or vice-versa. This is actually an object reconciliation problem. Giving the matchings we can use some mature object reconciliation algorithm to reconcile the instances. The reconciliation result can help find new matchings. Therefore this is an iterative process between ontology matching and object reconciliation.

In the next step, we generate matching groups of classes and properties. Then we generate the input for the data multi-relational data mining (MRDM) system. Finally MRDM system will mine the *frequent queries* based on the input data. A query is a logical expression of the form $? - P_1, ..., P_n$, which contains an atom key used for counting. For one query if the number of answers of atom key exceeds the threshold, we call this query as frequent query. For this example, one frequent query could be:

*?- @UMD:Person(x), @UMD:name(x,y), @CMU:name_person(x,y)*

where *Person* is the atom key and we use "*@UMD:*" and "*@CMU:*" to represent prefixes of the People and Person& Employee ontologies respectively. Since we are interested in the frequent queries related to *name* and *name_person*, we will derive one rule from this frequent query:

$\forall x, y$ *@UMD:Person*$(x) \wedge$*@UMD:name*$(x, y) \rightarrow$ *@CMU:Person*$(x) \wedge$ *@CMU:name_person*$(x, y)$

[2] http://www.cs.umd.edu/projects/plus/DAML/onts/personal1.0.daml
[3] http://www.daml.ri.cmu.edu/ont/homework/atlas-cmu.daml

## 4 Matching and Object Reconciliation

In this section, we introduce an iterative process between the Matching Generator and Object Reconciliation Processor. We still use some examples based on the People and Person&Employee ontologies to help demonstrate how this process works.

### 4.1 Basic Name Matching

We begin from finding class matching pairs and property matching pairs based on their names. We make use of "Iterative SubString Matching Algorithm" [30] to calculate the similarity of names of each pair. We basically examine every pair of classes and properties from both ontologies. For example, if $N$ is the number of classes in the source ontology and $M$ is the number of classes in the target ontology, there are $N * M$ potential class matching pairs. For each pair we can get a similarity score and we also set a threshold for name similarity to get class matching pairs. Similarly we find some property matching pairs. Other existing matching approaches (e.g., synonym-based approaches by using Wordnet[15]) can also be used in this step to help find more matchings.

### 4.2 Datatype Property Matching

There are some property matching pairs which can strengthen our confidence about potential class matching pairs. For example, one kind of OWL properties is datatype properties and the range is a data type, such as string and number. Datatype property matching pairs can support our system to match a class pair with higher confidence. We can use the types of property arguments to find them. Given the datatype property pair $p(X, Y)$ and $q(U, V)$, where $X$ and $U$ are classes and $Y$ and $V$ are data types, if there is a class matching pair $X \leftrightsquigarrow U$ and $Y$ is the same data type as $V$, we consider $p(X, Y) \leftrightsquigarrow q(U, V)$ as one potential datatype property matching related to the class matching pair $X \leftrightsquigarrow U$. Not only the name similarity of $p$ and $q$ is needed to make more confidence of $X \leftrightsquigarrow U$, but also the data value similarity of $p$ and $q$. One potential datatype property matching will be verified by data value similarity which will be further discussed in Section 4.3. An example is that $@UMD{:}name(Person, String) \leftrightsquigarrow @CMU{:}name\_person(Person, String)$ is one potential datatype property matching based on class matching pair $@UMD{:}Person \leftrightsquigarrow @CMU{:}Person$. The higher the similarity of datatype property matching pairs are, the more confidence this class matching pair has. Support of class matching pair is calculated according to the following equation:

$$Support(X \leftrightsquigarrow U) = \Sigma DatatypePropertyPairSimilarity \quad (1)$$

Datatype property pair similarity is the sum of name similarity and data similarity, which will be further discussed in Section 4.3.

Another problem we should cope with is that two classes may have totally unrelated names, but they represent the same concept. One clue to handle this case is actually from datatype property matchings. If several property matching pairs indicate that class $X$ and class $U$ should be a pair, we can assume $X \leftrightsquigarrow U$ is one class matching pair and add those property matching pairs as its datatype property matchings.

### 4.3 Data Similarity of Datatype Property Pairs

Having a similar name does not necessarily mean that two properties definitely represent the similar concept. Therefore, calculating data similarity of property matching pairs is necessary. Note that data similarity is based on the assumption that the data instances of two ontologies overlap at a relatively high level, otherwise we can not benefit from data level examination. The data similarity can help verify both property matching pairs and those class matching pairs that they are related to. For an existing property matching pair $p(X,Y) \leftrightsquigarrow q(U,V)$, we can calculate the data similarity of $p$ and $q$ by using the following formula:

$$DataSimilarity(p(X,Y) \leftrightsquigarrow q(U,V)) = \frac{2 * |same\_pairs(Y,V)|}{|p(X,Y)| + |q(U,V)|} \quad (2)$$

where $|same\_pairs(Y,V)|$ is the number of the instance pairs which have the same integer value or very similar string value, and $|p(X,Y)|$ and $|q(U,V)|$ stand for the number of instances of property $p(X,Y)$ and $q(U,V)$.

Then we can calculate the total similarity of each class matching pair. For one class matching pair $X \leftrightsquigarrow U$, the similarity is the sum of name similarity and its support. If both clues show that this matching pair is not correct, we remove it from the class matching pair list.

### 4.4 Object Reconciliation

After we get the selected class matching pairs, we adopt an object reconciliation algorithm developed by Dong, Halevy and Madhavan in [10] to reconcile the instances of classes from two ontologies. The original algorithm considers the relationship of matching pairs and determines whether two data objects in different databases represent the same real-world entity. We successfully use the idea in our ontology mapping examples.

We do not want to repeat the detail algorithm since it can be found in the paper [10]. In a summary, for all the possible instance pairs, we can draw a similarity graph and calculate the similarity between them. The formula we use is a simple aggregation of the similarity of datatype matching pairs. For example, Figure 2 shows one positive example and one negative example for People and Person&Employee ontology mapping. Node (person001, p001) has a high similarity and is considered as reconciled. Node (person002, p003) has a comparably low similarity and is considered as not reconciled.

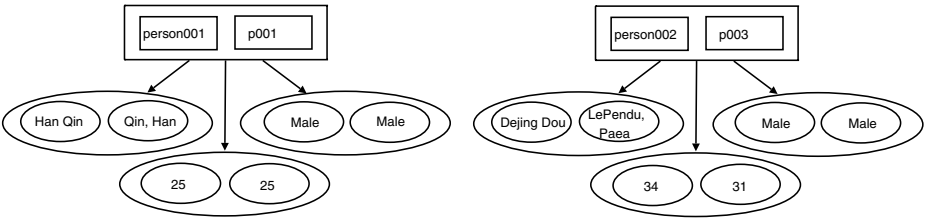


**Fig. 2.** Positive and negative object reconciliation examples

### 4.5 Object Property Matching

In OWL, object properties represent the relationships between two classes, such as the property $alumnus(Organization, Person)$ from the UMD People ontology. Similar to datatype property matching pairs, the object property matching pairs may have similar property name or not. However, it is harder to find object property matchings because their data instances can not help before object reconciliation process is performed. Therefore, only after we reconcile some data instances we can calculate the data similarity of object property matching pairs. Note that this kind of matching pairs have two ways to match: given $p(X, Y) \leftrightsquigarrow q(U, V)$, the first matching is $X \leftrightsquigarrow U, Y \leftrightsquigarrow V$ while the other is $X \leftrightsquigarrow V, Y \leftrightsquigarrow U$. When we calculate the data similarity of object property matchings, whether it is cross matched should be labeled. Similar to data property matching pairs, we give the following formula:

$$DataSimilarity(p(X,Y) \leftrightsquigarrow q(U,V)) = \frac{|same\_pairs(X,U)| + |same\_pairs(Y,V)|}{|p(X,Y)| + |q(U,V)|} \tag{3}$$

After Object Reconciliation Processor executes in each iteration, the Matching Generator tries to create new object property matching pairs based on the object reconciliation results. Figure 3 shows an example of this iterative process: after the first time Object Reconciliation Processor executed based on *@UMD:Person* $\leftrightsquigarrow$ *@CMU:Person*, the system found that "person001" is the same entity as "p001". Given this result, Matching Generator will find a new object property matching pair *@UMD:alumnus* $(Organization, Person)$ $\leftrightsquigarrow$ *@CMU:has_employes*$(Organization, Person)$, since there exists two class matching pairs *@UMD:Organization* $\leftrightsquigarrow$ *@CMU:Organization* and *@UMD:Pe− rson* $\leftrightsquigarrow$ *@CMU:Person* and the data similarity of this property matching pair also shows these two properties should be matched. This new property matching pair will be returned to the system to suggest reconcile more data instances related to *@UMD:Organization* $\leftrightsquigarrow$ *@CMU:Organization*, such as "uo_cs" and "CS_dept". This reconciliation result may help Matching Generator to find more object property matching pairs related to *@UMD:Organization* and *@CMU:Organization*. The process will be end if no new object property matchings can be found. At the end we have the complete graph of class matching pairs and property matching pairs as shown in Figure 4.

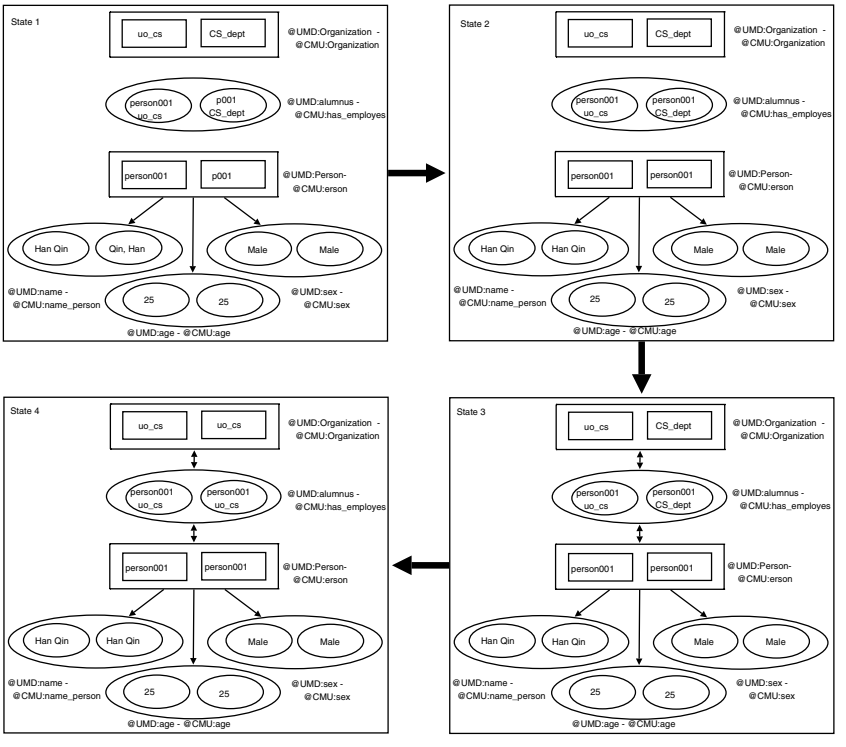


**Fig. 3. The Iterative Process Of the People and Person&Employee Ontology Mapping Example:** After Matching Generator creates new matchings in each iteration, Object Reconciliation Processor can reconcile more data instances and then help MG to find more new matchings. This process will terminate when MG can not find any new matchings.

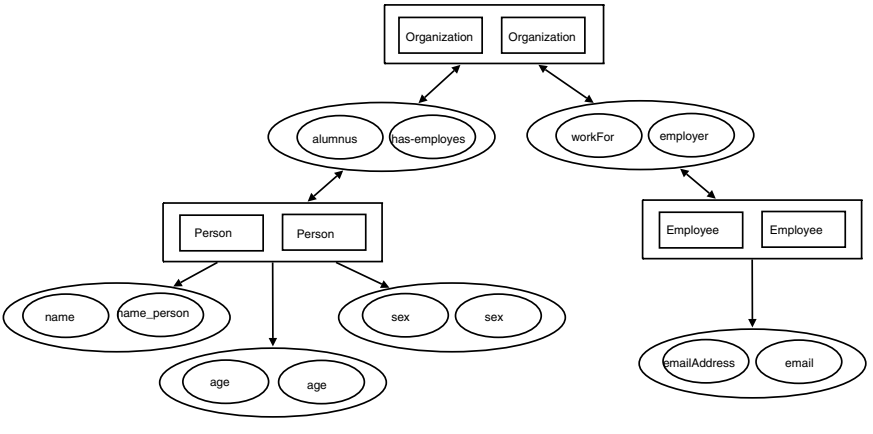


**Fig. 4.** The Graph Of Class Matching Pairs and Property Matching Pairs

## 5 Semantic Mapping Rule Mining

Based on the matching and object reconciliation results, we first generate group matchings and then use data mining techniques to discover final mapping rules.

### 5.1 Matching Groups

In this step, we tackle the problem of reducing the search space of the MRDM systems by generating groups based on 1-1 matchings. The simplest case is to put some of them together as a group based on related classes or properties, such as that *@UMD:name(Person, String)* ⇜⇝ *@CMU:name_person(Person, String)*, *@UMD:Person* ⇜⇝ *@CMU:Person* compose one group. The basic grouping rule we have used is:

**Basic Grouping Rule:** *For one property matching pair @source:p(X,Y)* ⇜⇝ *@target:q(U,V), if there are class matching pair @source:X* ⇜⇝ *@target:U and @source:Y* ⇜⇝ *@target:V, we generate one group which includes all these three matchings.*

Note that if *@source:Y* ⇜⇝ *@target:V* is a pair of string or number, we do not have to really put them in. This rule can cover most of the 1-1 property matching pairs, especially datatype property matching pairs.

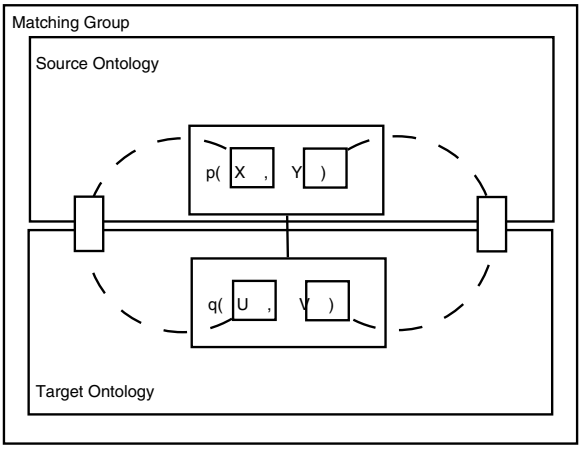


**Fig. 5.** Matcha ming Group Based On One Property Matching Pair

A complex case is that when we check *@source:p(X,Y)* ⇜⇝ *@target:q(U,V)* we only know that one class matching pair (i.e., *@source:X* ⇜⇝ *@target:U* or *@source:Y* ⇜⇝ *@target:V*) exists. Suppose that *@source:X* ⇜⇝ *@target:U* exists, then obviously this group is not complete yet. Therefore, we have to include other properties in this group to make it complete. The basic idea is: assume we can find *@source:Z* ⇜⇝ *@target:V* and in the source ontology there is one property *@source:r(Y,Z)*, then we can make the guess and include *@source:r(Y,Z)* in

this group. Another possible solution is try to find *@target:t(W,V)* if *@source:Y* $\leftrightsquigarrow$ *@target:W* exists. To sum up, we have to find some connection between *@source:Y* $\leftrightsquigarrow$ *@target:V*. One example in Figure 6 is *@UMD:workphone (Person, String)* $\leftrightsquigarrow$ *@CMU:phone(Office, String)*. In the source ontology (i.e., CMU), we can find one related property *@CMU:office(Employee, Office)*, which connects *@UMD:Person* with *@CMU:Office* since *@CMU:Employee* is a subclass of *@CMU:Person* and *@UMD:Person* $\leftrightsquigarrow$ *@CMU:Person* is a class matching pair. Thus this group contains three properties.

For more general cases, we should find connections between both *@source:X* $\leftrightsquigarrow$ *@target:U* and *@source:Y* $\leftrightsquigarrow$ *@target:V*. Figure 5 shows what a complete group is. The dashed line refers to zero or several predicates, super-sub class relationships or class matching pairs. Based on this, we can draw the general group rule:

**General Grouping Rule:** *For one property matching pair, such as @source: p(X,Y)* $\leftrightsquigarrow$ *@target:q(U,V), if we do not have class matching pair @source:X* $\leftrightsquigarrow$ *@target:U (or @source:Y* $\leftrightsquigarrow$ *@target:V), we can search among the properties and class matching pairs to find a connection path from @source:X to @target:U (or from @source:Y to @target:V). In the path there must exist one class matching pair that connects the source and target ontologies.*

Discovering the path is the key step for finding group matchings. And the class matching pair that connects the source and target ontologies is the key class matching pair. To find the path from *@source:X* to *@target:U*, if we can find the key class matching pair *@source:A* $\leftrightsquigarrow$ *@target:B*, the path contains the path from *@source:X* to *@source:A*, *@source:A* $\leftrightsquigarrow$ *@target:B* and the path from *@target:B* to *@target:U*. And we propose Algorithm 1 to find the key class matching pair:

**Algorithm 1.** Searching Key Class Matching Pair

**Input:** class X from source ontology, class U from target ontology, properties of both ontologies except p(X, Y) and q(U, V), class matching pair set, super-sub class relationship in two ontologies.

**Output:** the key class matching pair

Initialize source class set with X
Initialize target class set with U
**while** There does not exist class matching pair A $\leftrightsquigarrow$ B, where A belongs to source class set and B belongs to target class set. **do**
  For all t(H, K), t(K, H), superclass(H, K) and subclass (H, K) where H is in source class set, add K into source class set.
  For all r(N, M), r(M, N), superclass(M, N) and subclass (M, N) where M is in target class set, add N into target class set.
  If no new classes is added, return No_pair.
**end while**
Return A $\leftrightsquigarrow$ B

### 5.2 Generating Mapping Rules

The first step of this part is to discover frequent queries. The algorithm should take a set of predicates and data instances as input, build the search space and finally output a set of queries with high support. We consider classes and properties of ontology as unary or binary predicates. FARMER [25] system can be used for this goal. However, FARMER requires users to specify the input/output type of each argument of the predicates. To make the whole process as automatically as possible we borrow some ideas of FARMER but create a new algorithm (see Algorithm 2) instead in our implementation.

**Algorithm 2.** Generating frequent queries

**Input:** A matching group G. Data instances of all the predicates in G.
**Output:** Frequent queries with their support.

```
Create the first query with the key class matching pair of G.
while Not all predicates of the source ontology is added to the first query. do
    Suppose the type of last argument of the first query is T. Then find predicate P
    (in the source ontology) which has first argument type T. Add P to the end of the
    first query.
end while
Calculate the support of the first query.
Create the second query as a copy of the first query.
while Not all predicates of the target ontology is added to the second query. do
    Suppose the type of last argument of the second query is V. Then find predicate
    Q (in the target ontology) which has first argument type V. Add Q to the end of
    the second query.
end while
Calculate the support of the second query.
Return two queries with their support.
```

The next step of rule generation is a natural extension from frequent queries. The Rule Refiner can generate mapping rules based on frequent queries and class matching pairs. For example, the output of matching group `G` is:

```
? - Person(V0N0),name(V0N0,V1N0) support: 100
? - Person(V0N0),name(V0N0,V1N0),name_person(V0N0,V1N0) support: 95
```

With class matching pair $@UMD{:}Person \leftrightsquigarrow @CMU{:}Person$, a rule like $\forall x, y\ @UMD{:}Person(x) \wedge @UMD{:}name(x, y) \rightarrow @CMU{:}Person(x) \wedge @CMU{:}name\_person(x, y)$ can be generated.

This process is similar to a typical multi-relational data mining process. The main difference is that the information of each matching group helps to reduce the search space of query searching. We consider rules with extremely low support and confidence as distinctly incorrect. Thus we set a very low threshold and keep the rest rules.

# 6 Case Study

## 6.1 People vs. Person and Employee Ontology

The first case we test is the complete UMD People Ontology and CMU Person & Employee Ontology mapping example. The partial examples we give in previous sections are from this case. Figure 6 shows part of two ontologies and some human labeled matchings. Dotted lines refer to property matchings and dashed lines refer to class matchings.

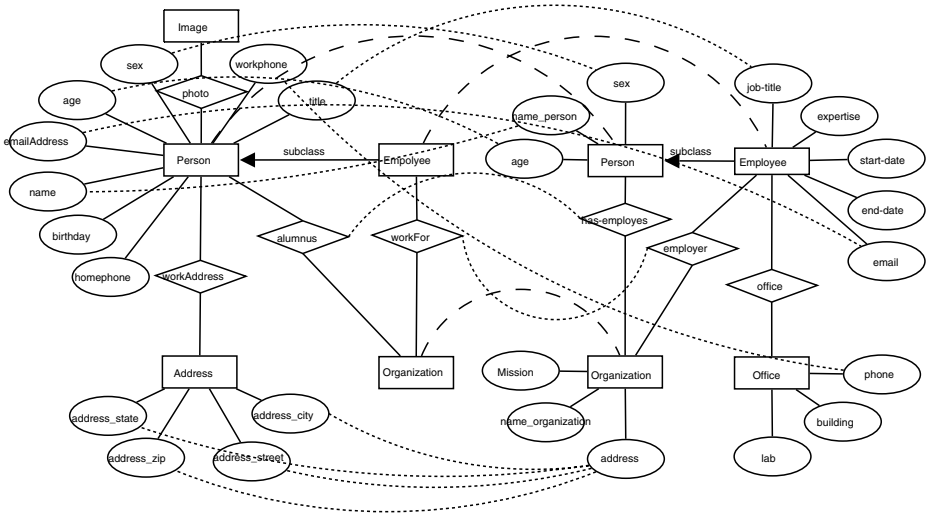


**Fig. 6.** UMD and CMU person ontology

The Matching Generator and Object Reconciliation Processor gives three class matching pairs and 21 property matching pairs. For some matching pairs, we cannot find any complete group. Group Generator actually outputs 17 matching groups. Finally, our system generates 8 rules. With human observation, we can get 9 rules manually. Therefore, the performance of our system in this case is satisfying as 8 out of 9. We can represent the mapping rules in first order logic syntax, such as: $\forall x, y\ Employee(x) \wedge worksfor(x, y) \rightarrow Employee(x) \wedge employer(x, y)$. Also we can represent these rules in our Web-PDDL or different potential standard mapping languages. For example, we can represent rules with Datalog:

```
emailAddress(A, B) :- Person(A), email(A,B).
```

Or we can represent rules with SWRL as following:

```
<ruleml:imp>
 <ruleml:_rlab ruleml:href="#Rule2"/>
```

```
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="&UMD;Person" />
      <ruleml:var>x</ruleml:var>
    </swrlx:classAtom>
    <swrlx:individualPropertyAtom  swrlx:property="&UMD;name">
      <ruleml:var>x</ruleml:var>
      <ruleml:var>y</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom  swrlx:property="&CMU;name_person">
      <ruleml:var>x</ruleml:var>
      <ruleml:var>y</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

The reason we missed one rule is: our system can find the matchings which contain *@UMD:address_state*, *@UMD:address_city*, *@UMD:address_zip*, *@UMD:address_steet* and *@CMU:address*. However, data mining system cannot introduce functions, therefore we do not derive any rule for these matchings successfully. This missing rule needs a concatenation function to combine *@UMD:address_state*, *@UMD:address_city*, *@UMD:address_zip*, *@UMD: address_steet* to *@CMU:address*.

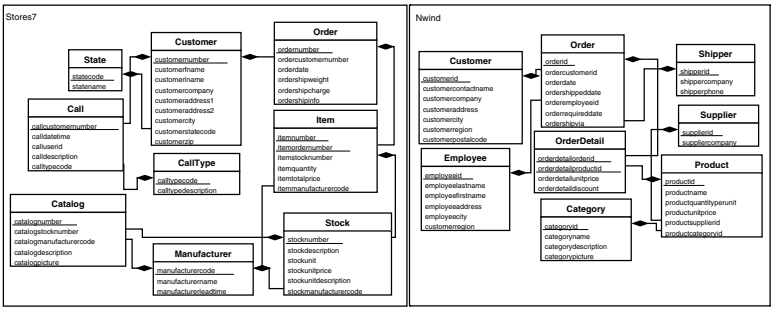


**Fig. 7.** Stores7 and Northwind schema

### 6.2 Online Sale Databases

Our approach can also be applied to databases. Our previous research [11,12] demonstrates a way to convert database schemas to ontologies. In this case, we consider two database schemas: Stores7 from IBM Informix4[4] and Northwind from Microsoft. Both of them are from online sales domain and have related concepts, such as Customer, Order and etc. Figure 7 shows the schemas of two

[4] http://www.ibm.com/software/data/informix/

databases. We have used these two databases to test our OntoGrate system but we used human-specified mappings in [11,12].

Humans can find 18 rules for these two database schemas. And our system discovers 4 class matching pairs and 201 property matching pairs. Then finally 16 rules are obtained by our system. Two rules of our output are incorrect. The reason of incorrectness is also that the MRDM system does not introduce functions, same as the address matching group problem addressed in section 6.1.

## 7 Conclusion and Future Work

We present a highly automatic approach which combines ontology matching, object reconciliation and multi-relational data mining to discover the executable mapping rules between given source and target ontologies from same domain. Our main novel contributions are:

1. We propose an iterative process: basic matchings can be used to guide object reconciliation and the result of object reconciliation can guide to find new matchings. This process also help verify existing matchings.

2. We propose a way to combine matching pairs to form matching groups, which is used to generate queries and mapping rules.

3. We use a data mining approach to find frequent queries and then convert them to mapping rules. We use group matchings to reduce the search space.

Our approach relies on both data instances and ontologies, and thus a constraint is that we must need ontologies with related data instances. Our system cannot guarantee 100% accurately generated rules. If users need 100% perfect results, human effort is surely needed. There are still a lot of interesting problems we cannot solve yet. The first problem we are going to solve is to discover mappings with new functions, such as $\forall x, y, z Person(x) \wedge city\_address(x, y) \wedge street\_address(x, z) \rightarrow address(x, concatenate(y, z))$. Then we will consider how to automatically evaluate an ontology mapping system without human-specified results and how to manage the mapping rules in the scenario that ontologies or database schemas keep changing.

## References

1. Semantic Web Rule Language, `http://www.w3.org/Submission/SWRL/`
2. An, Y., Borgida, A., Mylopoulos, J.: Constructing complex semantic mappings between XML data and ontologies. In: International Semantic Web Conference, pp. 6–20 (2005)
3. An, Y., Borgida, A., Mylopoulos, J.: Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. In: OTM Conferences (2), ODBASE, pp. 1152–1169 (2005)
4. Bayardo, R.J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A., Woelk, D.: InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In: SIGMOD 1997. Proceedings of the 1997 ACM SIGMOD international conference on Management of data, pp. 195–206. ACM Press, New York (1997)

5. Bruijn, J.D., Polleres, A.: Towards an Ontology Mapping Specification Language for the Semantic Web. Technical report, Digital Enterprise Research Institute (June 2004)
6. Dehaspe, L., Toivonen, H.: Discovery of frequent datalog patterns. Data Min. Knowl. Discov. 3(1), 7–36 (1999)
7. Dhamankar, R., Lee, Y., Doan, A., Halevy, A.Y., Domingos, P.: iMAP: Discovering Complex Mappings between Database Schemas. In: Proceedings of the ACM Conference on Management of Data, pp. 383–394. ACM Press, New York (2004)
8. Doan, A., Domingos, P., Halevy, A.Y.: Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In: Proceedings of the ACM Conference on Management of Data, ACM Press, New York (2001)
9. Doan, A., Madhavan, J., Domingos, P., Halevy, A.Y.: Learning to Map Between Ontologies on the Semantic Web. In: WWW. International World Wide Web Conferences, pp. 662–673 (2002)
10. Dong, X., Halevy, A., Madhavan, J.: Reference reconciliation in complex information spaces. In: SIGMOD 2005. Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 85–96. ACM Press, New York (2005)
11. Dou, D., LePendu, P.: Ontology-based Integration for Relational Databases. In: SAC 2006. Proceedings of the 2006 ACM symposium on Applied computing, pp. 461–466. ACM Press, New York (2006)
12. Dou, D., LePendu, P., Kim, S., Qi, P.: Integrating Databases into the Semantic Web through an Ontology-based Framework. In: SWDB 2006. Proceedings of the third International Workshop on Semantic Web and Databases, p. 54 (2006)
13. Dou, D., McDermott, D.V., Qi, P.: Ontology Translation on the Semantic Web. Journal of Data Semantics 2, 35–57 (2005)
14. Dragut, E., Lawrence, R.: Composing mappings between schemas using a reference ontology. In: ODBASE. Proceedings of International Conference on Ontologies, Databases and Application of Semantics (2004)
15. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
16. T. Gene Ontology Consortium. Creating the Gene Ontology Resource: Design and Implementation. Genome Research, 11(8), 1425–1433 (2001)
17. Haas, L.M., Hernandez, M.A., Ho, H., Popa, L., Roth, M.: Clio Grows Up: From Research Prototype to Industrial Tool. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 805–810. ACM Press, New York (2005)
18. Hu, W., Qu, Y.: Block matching for ontologies. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 5–9. Springer, Heidelberg (2006)
19. Lindberg, D., Humphries, B., McCray, A.: The Unified Medical Language System. Methods of Information in Medicine 32(4), 281–291 (1993)
20. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic Schema Matching with Cupid. In: Very Large Data Bases (VLDB) Conference, pp. 49–58 (2001)
21. Maedche, A., Motik, B., Silva, N., Volz, R.: MAFRA - A MApping FRAmework for Distributed Ontologies, pp. 235–250 (2002)
22. McGuinness, D.L., Fikes, R., Rice, J., Wilder, S.: The Chimaera Ontology Environment. In: Proceedings of the National Conference on Artificial Intelligence, pp. 1123–1124 (2000)

23. Mena, E., Kashyap, V., Sheth, A.P., Illarramendi, A.: Observer: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In: CoopIS, pp. 14–25 (1996)
24. Miller, R.J., Hernández, M.A., Haas, L.M., Yan, L.-L., Ho, C.T.H., Fagin, R., Popa, L.: The clio project: Managing heterogeneity. SIGMOD Record 30(1), 78–83 (2001)
25. Nijssen, S., Kok, J.N.: Efficient frequent query discovery in farmer. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) PKDD 2003. LNCS (LNAI), vol. 2838, pp. 350–362. Springer, Heidelberg (2003)
26. Noy, N.F.: Semantic Integration: A Survey Of Ontology-Based Approaches. SIGMOD Record 33(4), 65–70 (2004)
27. Noy, N.F., Musen, M.A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: Proceedings of the National Conference on Artificial Intelligence, pp. 450–455 (2000)
28. Quinlan, J.R., Cameron-Jones, R.M.: Foil: A midterm report. In: Brazdil, P.B. (ed.) ECML 1993. LNCS, vol. 667, pp. 3–20. Springer, Heidelberg (1993)
29. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB J. 10(4), 334–350 (2001)
30. Stoilos, G., Stamou, G.B., Kollias, S.D.: A string metric for ontology alignment. In: International Semantic Web Conference, pp. 624–637 (2005)
31. Stuckenschmidt, H., Uschold, M.: Representation of semantic mappings. Semantic Interoperability and Integration (2005)
32. Wache, H., Vogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hubner, S.: Ontology-based integration of information: A survey of existing approaches. In: IJCAI 2001. Workshop: Ontologies and Information Sharing, pp. 108–117 (2001)