Grounding the Foundations of Ontology Mapping on Neglected Ambitions

Hamid Haidarian Shahri*, James A. Hendler+

* MINDSWAP Research Group
Department of Computer Science
University of Maryland, College Park, USA
hamid@cs.umd.edu

+ Tetherless World Research Constellation
Computer Science Department
Rensselaer Polytechnic Institute, Troy, USA
hendler@cs.umd.edu

Abstract. The problem of ontology mapping has attracted considerable attention over the last few years, as the usage of ontologies is increasing. In this paper, we revisit the fundamental assumptions that drive the mapping process, put the current research in context, and propose new research directions. Based on real-world use cases, we identify two distinct goals for mapping, which are: (i) ontology development and (ii) facilitating interoperability. Most of current research on ontology mapping has been focused on *ontology development* and is rooted in the seminal work of [McG00, Noy00]. For example, the well studied problem of ontology merging is an ontology development task. Note that, *today*, with the increase in the number of information systems that utilize ontologies, *facilitating interoperability* between these systems is becoming more critical. However, mapping has seldom been studied, from this perspective. We show the consequences of focusing on interoperability, with illustrative examples, and provide an in-depth comparison to the information integration problem in databases. As a result, class matching is emphasized, as opposed to the matching of other entities in an ontology. Various class similarity metrics are proposed and their complexity is analyzed. Then, an algorithm, which utilizes these metrics, is designed and evaluated experimentally. To the best of our knowledge, this is the first work that focuses on the neglected goal of facilitating interoperability, in ontology mapping.

1. Introduction

The need for communicating between autonomous and distributed information systems is increasing with the wide usage of the Web. Nowadays, the issue of sharing data across resources and enterprises is no longer a desirable feature, but a necessity. Considerable amount of research on data integration and schema mapping over the past decades have lead to significant improvements in this area [Rah01, Hal96, Hal06, Bat86]. The difficulty of finding correspondences between schemas originates from the fact that the conceptual models, used for data representation, do not capture the semantics of the data with enough precision. For example, it is very difficult to infer that *area* in one schema and *location* in another schema refer to the same real-world entity, as the meaning of attributes in the schema is not encoded explicitly. This problem is referred to as semantic heterogeneity.

Ontologies encode the specification of concepts more accurately, than schemas. The rich set of relationships defined between concepts in ontologies, help in mitigating the semantic heterogeneity problem. Since different ontologies exist and are being used by various organizations, it is necessary to find *correspondences* between these ontologies. The terms: ontology mapping, matching, alignment, integration, and merging, in the research literature, relate to this issue in various ways! In fact, unifying the interpretation of this diverse terminology is quite challenging. Usually, the goals of the task of finding correspondences in ontologies are not explicitly stated. Moreover, there is considerable vagueness on how the task should be performed, as the problem is often stated for some *specific* setting, or a *theoretical* one. Generally, there exists no consensus on what solution to use and under what circumstances, as evident by the variety of the terminology used. Nevertheless, previous research [Kal03, Noy04, McG00] is very valuable for developing the foundations of the ontology mapping problem.

In our opinion, this vagueness can only be resolved by observing the *use cases* of the problem. To the best of our knowledge, this is one of the very first attempts to put the previous research, on ontology mapping, in a unified context. This study revisits the ontology mapping problem in *various settings*, to furnish generality, and at the same time avoids *theoretical assumptions*, by adhering to real-world use cases.

The contributions of the paper are as follows: (i) Different *use cases* of ontology mapping are explored and clarified with real-world motivating examples. (ii) Two separate *goals* of the ontology mapping problem are identified, based on the use cases. They are interoperability and ontology development. (iii) *Interoperability* is highlighted as the major goal in ontology mapping, and the problem is revisited in this context, as opposed to the usual ontology development context. (iv) We use illustrative examples to show the consequences of focusing on interoperability, and also provide an in-depth comparison to the information integration problem in databases. Based on this comparison, *class matching* is emphasized as the main ingredient in ontology mapping. This is different from finding all matching entities, which is the focus of ontology development efforts. (v) Various *metrics* for finding the matching (corresponding) classes are proposed, and a formalization of these metrics is provided. The *time complexity* of computing these metrics is analyzed, which is important for scalability in real-world settings. (vi) Finally, an algorithm that utilizes these class similarity metrics is designed and implemented. Our *empirical results* support the effectiveness and scalability of these metrics, through various experiments.

The paper is organized as follows. In section 2, we identify the goals of ontology mapping with motivating examples. In section 3, the problem of information integration is defined, class matching is emphasized as the main ingredient of the mapping, and various class similarity metrics are proposed. In section 4, an experimental evaluation of these metrics is provided. Section 5 covers the related work, and section 6 is the conclusion.

2. Revisiting Ontology Mapping Goals with Motivating Examples

Currently, there are many ontologies that have been designed by different organizations and communities, and hence there is a need for a mapping between them. There are two quite distinct goals for ontology mapping. These goals are based on the types of use cases that we have identified, and will clarify in this section with motivating examples. Although, there are similarities between the use cases, one can differentiate the subtle requirements that arise from these examples, with careful observation. One possible goal of mapping is *ontology development*, when an ontology is being designed or engineered by an organization. The other possible goal of mapping is *interoperability*, when there are various parties, which are using different ontologies and the parties need a mechanism to be able to communicate and exchange information. ***This distinction has seldom been addressed, in previous research on ontology mapping.*** Clearly, interoperability is of considerable importance, as will be explained in this section, but unfortunately this goal is overlooked, in the research literature.

Ontology Development: Since ontology is an abstraction for representing knowledge and all concepts that fit into the domain of human knowledge are connected together in some fashion, it is very hard to limit an ontology in terms of what it represents. This decision is usually made based on business needs, i.e. the ontology designer decides not to include some concepts, as they seem irrelevant to current organizational demands. Assume that an organization is currently using a *host ontology*, *H*.

Overtime, as business models change and evolve, the ontology *H* also needs to be changed and often extended. Sometimes, the new business models, or some fragments of the changes that are required in the ontology, have already been captured by ontologies that are being used in other organizations. In this case, the required extensions to host ontology *H*, are existent in some other *guest ontology*, *G*. Now, the ontology designer of *H*, needs to: 1) find the correspondences between ontologies *H* and *G*, 2) decide on what concepts, relations, and instances of *G*, need to be added to *H*, based on the correspondences found in the previous step. This use case closely resembles the problem that has been analyzed in the context of merging two ontologies, in the current literature [Noy00, McG00, Stu01, Udr07].

Example 1: Consider two organizations offering various products and using two different ontologies O_1 and O_2 shown in Figure 1(a) and 1(b), respectively. O_1 is shown with ovals, while O_2 is shown with rectangles. The orange color represents the corresponding concepts between O_1 and O_2. In Figure 1, since class *Videos* in ontology O_1 is defined in a very similar context to class *Movies* in ontology O_2, it is conceivable to merge the two ontologies and produce a more comprehensive ontology. In essence, O_1 is being extended with O_2 and the merged ontology is a mix of ovals and rectangles, as shown in Figure 1(c).

Both organizations may need to make changes in their operation, in order to use the merged ontology. Furthermore, merging can be problematic, if the ontologies are defining the classes in different contexts, as merging would easily lead to irresolvable inconsistencies. Assume that *Electronic Equipment* in O_1 also has *Toys* as a subclass. Now the merged ontology would have two *Toys* concepts, one of which is a subclass of *Sale Items* (Figure 1c) and the other is a subclass of *Electronic Equipment* (not shown). Even combining the two *Toys* concepts may not have the desired effect.

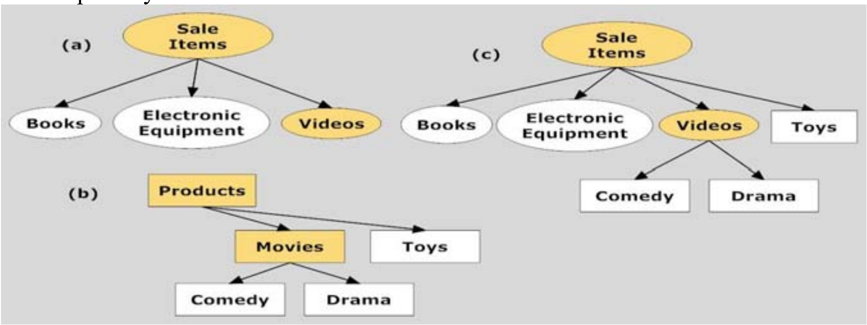


Fig. 1. Two ontologies O_1 and O_2 shown in (a) and (b). The merged result is shown in (c).

Important points arise from the study of this use case, which are as follows:

1. When ontologies are being merged, there is potential for inconsistencies and the ontology designer needs to make complex decisions in various steps of the process. Hence, the merging process can only be s*emi-automated* [Noy03] and no algorithmic solution exists. Moreover, the process must be *interactive*, to allow the designer to verify the changes.
2. The nature of the merging problem is such that the host ontology is usually not only being *extended*, but also needs to evolve, to accommodate the neighboring classes of the corresponding class in the guest ontology. For example, if the class *Movies* did not have a parent class, a simple extension would have sufficed, but now we must accommodate the *Products* class as well.

3. Finding the correspondences between two ontologies is necessary, as illustrated by this use case. This step should be focused on the *classes* that match, in the ontologies. Matching of corresponding properties and instances can only provide auxiliary information for the ultimate task of class matching, as we will explain in Section 3.
4. Merging can not be an *iterative* process and should be performed for a limited number of guests; in the context of developing a new host ontology that meets the new business demands in the organization. Notice that in this use case, we are only dealing with two ontologies (host and guest) and it is not possible to merge *many ontologies* from different parties to provide interoperability between the parties, as will be explained in section 2.2. This observation elegantly leads us to the other goal of ontology mapping, which is interoperability.

Interoperability: Different enterprises use their proprietary and autonomous systems and are often not willing to change their business models and operations. However, they also need to exchange information. In many circumstances, users need to query these distributed and autonomous sources of information, and retrieve data from all of them, as if all the information resides in a unified source.

Let us define this scenario more formally. Various autonomous ontologies, O_1, O_2, O_3, …, O_n, are designed and being used by *n* different organizations, also known as *parties*. Each ontology O_i is designed based on the business model that governs the operations of the organization that it belongs to. Hence, the ontology being used by each party can not be changed or extended. To facilitate interoperability, in this scenario, two steps are required: 1) correspondences between the ontologies of different parties have to be determined, 2) a skeleton *S*, must be developed, to represent these correspondences.

Example 2: Consider two universities in which faculties and departments within the faculties are structured differently, as shown in Figure 2(a) and 2(c). The ontologies O_1 and O_2 for the two universities are represented with ovals. There are six corresponding concepts in O_1 and O_2, namely: *University*, *Science*, *Maths*, *CS*, *Physics*, and *Chemistry*, shown with an orange color. Note that these six concepts appear in different places in O_1 and O_2. These six concepts are used in skeleton *S*, as shown in Figure 2(b), and represented with rectangles.

When creating the skeleton, one of the original ontologies takes the *master* role, while the other takes the *slave* role. In this example O_1 is the master and O_2 is the slave. The shape of the skeleton (i.e. the relationship between the concepts in the skeleton) is determined by the master, which is O_1 in our example. Then, each concept in skeleton *S* is connected to its corresponding concept in the original ontologies O_1 and O_2, with a subclass relation. Figure 2 only shows such connections for the *University* concept, with red dotted arrows, and other such connections are not drawn for more readability. In Example 2, each organization's ontology (i.e. O_1 and O_2) remains intact, unlike Example 1.

The following observations can be made by careful examination of Example 2, and comparing it to Example 1:

1. *Isolation*: Creating the skeleton *S*, to represent ontology mappings is much more flexible than merging, and the autonomous ontologies O_i, are isolated from any further changes. This is very desirable, since autonomous organizations, which are using the ontologies, are usually not willing to change their business practices for the sole purpose of communicating with other organizations. Hence, interoperability must be facilitated by other means.
2. *Class Matching*: Similar to the previous use case, determining correspondences between two ontologies should be focused on the *classes* that match, in the two ontologies. The matching of corresponding properties and instances only provides auxiliary information for the ultimate task of class matching. Section 3 will elaborate on why class matching

should be the focus, and formally defines the required terminology, like class, property and instance.

3. *Tractability*: The creation of skeleton *S*, is more tractable and comprehensible, and leads to fewer inconsistencies than the merging process. Therefore, it can be streamlined and tackled algorithmically.
4. *Scalability*: Note that when creating a global system to facilitate interoperability, if we were to merge the ontologies from many different parties (as explained in section 2.1), we would have to merge all the ontologies and produce one monolithic ontology, which is clearly not scalable, nor feasible.

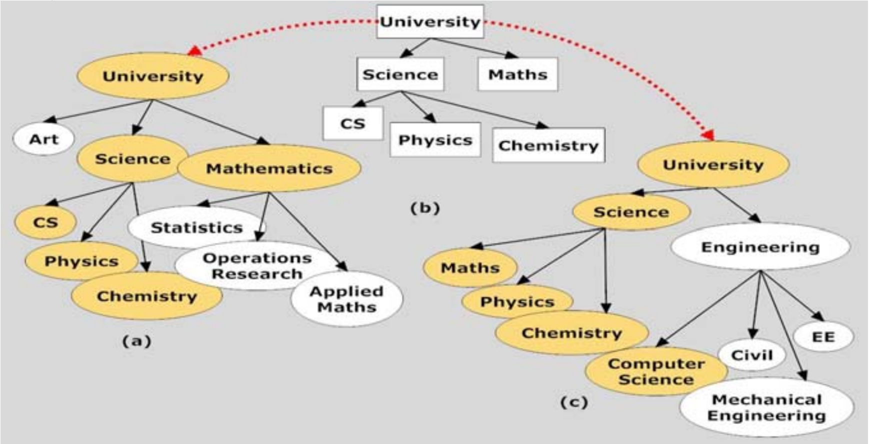


Fig. 2. O_1 and O_2 shown in (a) and (c) are the ontologies of two autonomous organizations. Skeleton *S* connecting the ontologies is shown in (b), in the middle.

The use cases above and the discussion in this section demonstrate that *interoperability* (i.e. facilitating the exchange of information between organizations) is a very important goal in ontology mapping. This goal is quite similar to what the database community is trying to achieve, in the context of information integration research and schema matching [Rah01, Len02, Bat86]. However, current solutions to the ontology mapping problem have not addressed this goal, and are primarily focused on merging ontologies. The merging process is geared towards the *development of an ontology*, which is the other goal identified in this section. In the following sections, we will concentrate on the interoperability goal, and describe its implications on the ontology mapping problem.

3. Class Matching: The Main Ingredient of Ontology Mapping

In the previous section, by observing the use cases, we illustrated that correspondence between ontologies need to be determined. In this section, first, we will show that the process of determining correspondences should be focused on *classes*, and not other *entities* (e.g. properties and instances) in an ontology. Then, a formalization of the class matching process will be provided, and various metrics that determine the similarity of classes will be explored. The required terminology will be defined gradually throughout the discussion.

This section is primarily based on Resource Description Framework (RDF), which is a framework and W3C recommendation for representing information on the Web. RDF is designed to represent information in a flexible way. The generality of RDF facilitates sharing of information between applications and making information accessible to more applications across the entire Internet. Note that the interoperability goal, as identified in the previous section, nicely aligns with RDF design. Moreover, the Web Ontology Language (OWL) is based on RDF. Hence, our discussion applies to OWL as well.

Definition 1 (Resource): All things described by RDF are called resources.

Definition 2 (Triple): Each triple represents a statement of a relationship between the things denoted by the nodes that it links. Each triple has three parts: a subject, an object, and a predicate that denotes a relationship.

The direction of the link is significant; it always points toward the object. An RDF triple is conventionally written in the order subject, predicate, object.

Definition 3 (Property): The predicate is usually known as the property of the triple.

Definition 4 (RDF Graph): An RDF graph is a set of RDF triples. The set of nodes of an RDF graph is the set of subjects and objects of triples in the graph.

The graph can be illustrated by a node and directed-arc diagram, in which each triple is represented as a node-arc-node link (hence the term "graph"). The assertion of an RDF triple says that some relationship, indicated by the predicate, holds between the things denoted by subject and object of the triple. The assertion of an RDF graph amounts to asserting all the triples in it, so the meaning of an RDF graph is the conjunction of the statements corresponding to all the triples it contains.

Definition 5 (Class and Instance): Resources may be divided into groups called classes. The members of a class are known as instances or individuals of the class. Associated with each class is a set, called the extension of the class, which is the set of the instances of the class.

Classes are themselves resources. They are often identified by URI's and may be described using RDF properties. The rdf:type property may be used to state that a resource is an instance of a class. RDF distinguishes between a class and the set of its instances.

If a class C is a subclass of a class C', then all instances of C will also be instances of C'. The rdfs:subClassOf property may be used to state that one class is a subclass of another. The term super-class is used as the inverse of subclass. If a class C' is a super-class of a class C, then all instances of C are also instances of C'.

Definition 6 (Datatype): A datatype consists of a lexical space, a value space and a lexical-to-value mapping. The lexical space of a datatype is a set of Unicode strings. The lexical-to-value mapping of a datatype is a set of pairs whose first element belongs to the lexical space of the datatype, and the second element belongs to the value space of the datatype.

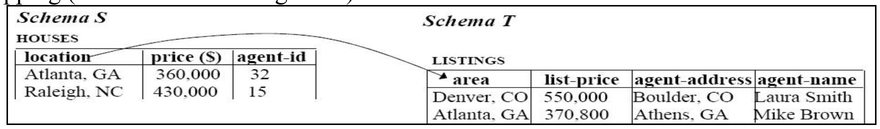
All datatypes are classes. The instances of a class that is a datatype are the members of the value space of the datatype. Each member of the lexical space is paired with (maps to) exactly one member of the value space. Each member of the value space may be paired with any number (including zero) of members of the lexical space (lexical representations for that value).

Definition 7 (Ontology): An ontology is an RDF graph, which is in turn a set of RDF triples.

3.1. A Harmonious Perspective on Information Integration, Interoperability and Class Matching

Information Integration: The problem of combining heterogeneous data sources under a single query interface is commonly known as "data integration" or "information integration" in the database community. Here, the idea is to provide a uniform query interface over a mediated schema. This query is then transformed into specialized queries over the original databases. This process can also be called view based query answering, because we can consider each of the data sources to be a view over the mediated schema. Formally such an approach is called Local As View (LAV), where “Local” refers to the local sources/databases. An alternate model of integration is one where the mediated schema is designed to be a view over the sources. This approach is called Global As View (GAV), where “Global” refers to the global (mediated) schema [Len02].

Figure 3 shows an example of the information integration problem in databases. Here, the goal is to generate a mapping between attributes in various schemas. The schemas usually reside in separate autonomous data sources. Figure 3 illustrates a mapping between schema *S* and schema *T*. The *location* attribute (column) in the *HOUSE* relation (table) is mapped to the *area* attribute in the *LISTINGS* relation. Note that this is a simple, but critical example, and will be used later in this section to demonstrate how ontology mapping should be performed to facilitate interoperability, and how ontology mapping relates to schema mapping (i.e. information integration).



Schema S

HOUSES

location	price ($)	agent-id
Atlanta, GA	360,000	32
Raleigh, NC	430,000	15

Schema T

LISTINGS

area	list-price	agent-address	agent-name
Denver, CO	550,000	Boulder, CO	Laura Smith
Atlanta, GA	370,800	Athens, GA	Mike Brown

Fig. 3. Example of the information integration problem in databases. The goal is to generate a mapping between attributes in various schemas. The schemas usually reside in separate autonomous data sources.

Interoperability: Following the above explanation, it is critical to point out that, the term “integration” is somewhat vague and imprecise, and could be interpreted as some type of “merging” of schemas, however, this is not what actually occurs in databases, as the schemas in each data source are handled autonomously in different organizations, and need to be kept that way. The ultimate goal of information integration is to provide *interoperability* between various systems, which is the exact same goal that we identified in section 2.2. Notice that the term “interoperability” is much clearer, for describing the motivations and objectives of the process. By analogy, in ontology mapping, there is no merging of ontologies involved, when we are trying to achieve interoperability between organizations, which use different ontologies (see section 2.2). Nevertheless, merging is useful in the context of developing new ontologies, as mentioned in section 2.1. The difference is subtle, but deeply rooted in the identification of the goals, and the distinction has seldom been addressed, in previous research on ontology mapping.

Expression of Simple Facts in RDF: Some simple facts indicate a relationship between two things. Such a fact may be represented as an RDF triple in which the predicate names the relationship, and the subject and object denote the two things. Figure 4(a) shows the predicate *authoredBy* which is the relationship between the subject *Book* and the object *Author*, both depicted as ovals. The use of extensible URI-based vocabularies in RDF facilitates the expression of facts about arbitrary subjects; i.e. assertions of named properties about specific named things. A URI can be constructed for any thing that can be named, so RDF facts can be about any such things.

Expression of Simple Facts in the Relational Model: A familiar representation of a fact might be as a tuple (row), in a relation (table) in a relational database. Figure 4(b) shows the *authoredBy* table. The table has two attributes (columns), namely *Book* and *Author*. These attributes correspond to the subject and object of the RDF triple, in Figure 4(a). The name of the table corresponds to the predicate of the RDF triple.

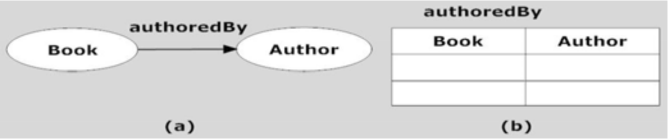


Fig. 4. Correspondence between the RDF and relational models. (a) The predicate authoredBy which is the relationship between the subject Book and the object Author in an RDF triple (b) The table authoredBy which has two attributes, namely Book and Author, in the relational model.

Class Matching: When integrating information between various databases for interoperability, correspondences between the attributes of two relations in two schemas S_i and S_j, where $i \neq j$, are determined, and the attributes are mapped to each other (as shown in Figure 3). Now, in an RDF triple, subject and object correspond to the *attributes* (i.e. columns). Hence, the subjects and objects need to be mapped to each other, in various ontologies. Note that subjects and objects are *classes* in the ontology, and tuples are instances of the classes. Therefore, if the *classes* are mapped accurately, instances can be retrieved correctly across various ontologies. Using the same analogy, in databases, if the attributes (columns) are mapped accurately, tuples can be retrieved correctly across various schemas.

The matching of entities, other than classes, in an ontology (like instances and properties), may be useful, when merging ontologies in ontology development efforts, however, this is not the case, when we are trying to facilitate interoperability. To the best of our knowledge, no previous work, in the ontology mapping literature, has emphasized the importance of matching of "classes", as opposed to other entities (e.g. properties, instances), in the ontology. This is one of the critical implications of focusing on the interoperability goal, in ontology mapping.

3.2. Class Matching Formalization and Categorization of Class Similarity Metrics

In the previous section, we identified class matching as the main ingredient of ontology mapping, to facilitate interoperability. Now, we provide a categorization of various class similarity metrics, and the complexity analysis of computing each metric. Then the role of the reasoner is explained, and our class matching algorithm is presented. Based on previous definitions, nodes in two RDF graphs, which are the classes in two ontologies, need to be matched.

Definition 8 (Map): Let C_1 be the set of classes of ontology O_1 and C_2 be the set of classes of ontology O_2. Map m is a total function $m : C_1 \otimes C_2 \to [0,1]$, where $C_1 \otimes C_2$ is defined as the set of all distinct unordered pairs of sets C_1 and C_2, that is: $C_1 \otimes C_2 = \{\{a,b\} \mid a \in C_1, b \in C_2, a \neq b\}$.

Definition 9 (Class Matching, Threshold, Similarity Value, Similarity Metric): Class matching is the process of determining corresponding classes between ontologies O_1 and O_2, which is specified using a threshold t. Map m assigns a similarity value to each pair of classes. If the similarity value, defined by map m, is greater than t, then the classes match. Similarity value is the sum of the following four similarity metrics: lexical, extensional, extensional closure, and global path, which are computed as follows.

Definition 10 (Lexical Similarity Metric): Lexical similarity metric is a function that assigns a real-valued number in the range of [0, 1], to $s \in C_1$ and $t \in C_2$, based on the closeness of the strings representing the names of s and t.

Theorem 1 (Lexical Similarity Complexity): The complexity of computing the lexical similarity metric for ontologies O_1 and O_2 is $O(|C_1|.|C_2|)$. There is no need for reasoning in the computation of this metric.

Definition 11 (Extensional Similarity Metric): Let $s \in C_1$ and $t \in C_2$ be two classes in the ontologies. The set of individuals which belong to s and t are represented as e(s) and e(t), respectively. Extensional similarity metric for s and t is computed as $| e(s) \cap e(t) | / | e(s) \cup e(t) |$.

Theorem 2 (Extensional Similarity Complexity): The complexity of computing the extensional similarity metric for classes s and t is $O(|e(s)|.|e(t)|)$. The set of individuals e is computed using the reasoner.

Definition 12 (Extensional Closure Similarity Metric): Let $s \in C_1$ and $t \in C_2$ be two classes in the ontologies. x is a subclass of y, is denoted as $x \sqsubseteq y$. Extensional closure of s, denoted as $e_c(s)$, is computed as $e_c(s) = \{ \bigcup_{i \in C_1} e(i) \mid i \sqsubseteq s \}$. Extensional closure similarity metric for s and t is equal to $| e_c(s) \cap e_c(t) | / | e_c(s) \cup e_c(t) |$.

This intuitively means that when comparing two classes, the extensional closure considers not only the individuals that belong to a class s, but also all the individuals that belong to subclasses of class s.

Theorem 3 (Extensional Closure Similarity Complexity): The complexity of computing the extensional closure similarity metric for classes s and t is $O(|e_c(s)|.|e_c(t)|)$. The subclasses of a class and their respective individuals are computed using the reasoner.

Definition 13 (Global Path Similarity Metric): Let $s \in C_1$ and $t \in C_2$ be two classes in the ontologies. Path of s, denoted as p(s), is the path which starts from the root and ends at s, in the RDF graph. Global path similarity metric for s and t is equal to the score assigned to similarity of p(s) and p(t). The score is based on the lexical similarity of classes included in the two paths.

Theorem 4 (Global Path Similarity Complexity): If the length of the path p(s) is denoted as $||p(s)||$ (which is equal to the depth of the class hierarchy in the worst case), then the complexity of computing the global path similarity metric for classes s and t is $O(||p(s)||.||p(t)||)$. The class hierarchy is created from the subclass relationships, using the reasoner.

Reasoning for Class Matching: The role of a reasoner is very central in class matching for ontology mapping, and this is one of the points that distinguish ontology mapping, from schema mapping in databases, as there is no reasoning involved in databases. Standard ontology reasoners provide the following services:

- *Classification*: Computes the subclass relations between every named class to create the complete class hierarchy. The class hierarchy can be used to answer queries such as getting all or only the direct subclasses of a class.
- *Realization*: Finds the most specific classes that an individual belongs to; in other words, computes the direct types for each of the individuals. Realization can only be performed after classification since direct types are defined with respect to a class hierarchy. Using the classification hierarchy, it is also possible to get all the types for that individual.
- *Consistency checking*: Ensures that an ontology does not contain any contradictory facts.
- *Concept satisfiability*: Checks if it is possible for a class to have any instances. If a class is unsatisfiable, then defining an instance of that class will cause the whole ontology to be inconsistent.

Class Matching Algorithm: Based on the previous definitions, we present our class matching algorithm, below:

```
ClassMatching Algorithm
Input: Ontologies $O_1$, $O_2$
Output: Set M of matching class pairs ($c_1$, $c_2$)
 1. for $c_1 \in C_1$, $c_2 \in C_2$
 2.   lexSim ← lexicalSim($c_1$.name, $c_2$.name)
 3.   ($c_1$.ex, $c_2$.ex) ← reasoner.Extensions($c_1$, $c_2$)
 4.   extSim ← extensionalSim($c_1$.ex, $c_2$.ex)
 5.   ($c_1$.all, $c_2$.all) ← reasoner.AllExtensions($c_1$, $c_2$)
 6.   extCSim ← extensionalClosureSim($c_1$.all, $c_2$.all)
 7.   ($c_1$.p, $c_2$.p) ← reasoner.GlobalPath($c_1$, $c_2$)
 8.   gpSim ← globalPathSim($c_1$.p, $c_2$.p)
 9.   if (lexSim+extSim+extCSim+gpSim > threshold) then
10.     M ← M $\cup$ ($c_1$, $c_2$)
11. end for
12. return M
```

The algorithm generally follows from the discussion in section 3.2. Line 2 relates to definition 10. Lines 3-4 compute the extensional similarity, as in definition 11. Lines 5-6 are based on the extensional closure similarity, as in definition 12. Lines 7-8 compute the global path similarity, as in definition 13. Lines 9-10 are based on the idea of similarity value and threshold, introduced in definition 9.

4. Experimental Evaluation

To evaluate the effectiveness of various class similarity metrics, we designed and implemented an ontology mapping module in Swoop. Swoop is an open source tool for browsing and development of ontologies, developed in the MINDSWAP research group [Kal05]. It is a hypermedia-based ontology editor that employs a web-browser metaphor for its design and usage. In our implementation, Pellet was used for reasoning [Sir07]. Pellet is an open source reasoner written in Java. Details of how reasoning is used in the class matching process were provided in section 3.2. The experiments were run on a 1.86 GHz Pentium machine with 512 MB of RAM and a Windows XP operating system.

Our experimental trials included a number of synthetic and real-world ontologies from various domains, to ensure generality. The results reported here are from two real-world ontologies that have been developed separately by different organizations. The Karlsruhe ontology [Kar] is used in the Ontoweb portal. It is a refinement from other ontologies such as (KA)[2]. It defines terms used in a university organization and bibliographic items. The INRIA ontology [Inr] has been designed by Antoine Zimmermann from the BibTeX in OWL ontology and the Bibliographic XML DTD. Its goal is to easily gather a number of RDF items. These items are BibTeX entries found on the web, and are transformed into RDF according to this ontology. The actual hierarchy of this ontology contains classes which are subclasses of several other classes. The ontologies have 24 corresponding classes. Table 1 shows the characteristics of the ontologies with more details.

When comparing the name of classes in two ontologies, various string similarity metrics can be used. We implemented the Jaro-Winkler, Jaccard, Monge-Elkan and Levenstein metrics and the results show that the performance of these metrics varies noticeably, as illustrated in Figure 5. The Jaro-Winkler metric shows a more robust behavior for ontology class matching, based on name. Precisions in the range of below 60 percent are not very useful, as many of the detected matches would be incorrect. By decreasing the threshold for identifying a match, we can increase the recall rate to some extend. However, as the diagram demonstrates, it is not possible to increase the recall to above 80%, by only decreasing the threshold, as this would cause a sharp drop in precision, i.e. introduce many incorrect results.

Table 1. Detailed characteristics of the ontologies.

	University Ontology	Publication Ontology
# Classes	64	48
# Properties	72	58
# Individuals	68	59
Min. Depth of Class Tree	1	1
Max. Depth of Class Tree	5	4
Average Depth of Class Tree	2.4	2.3
Min. Average Branching of Class Tree	1	1
Max. Average Branching of Class Tree	13	17
Average Branching Factor of Class Tree	3.15	3.25

Figure 6 shows the running time required of computing the lexical similarity of classes in both ontologies using various string similarity metrics. Jaro-Winkler, which showed best performance in the Figure 5, takes 172 ms to compute and lies approximately in between the other string similarity metrics, in terms of running time.

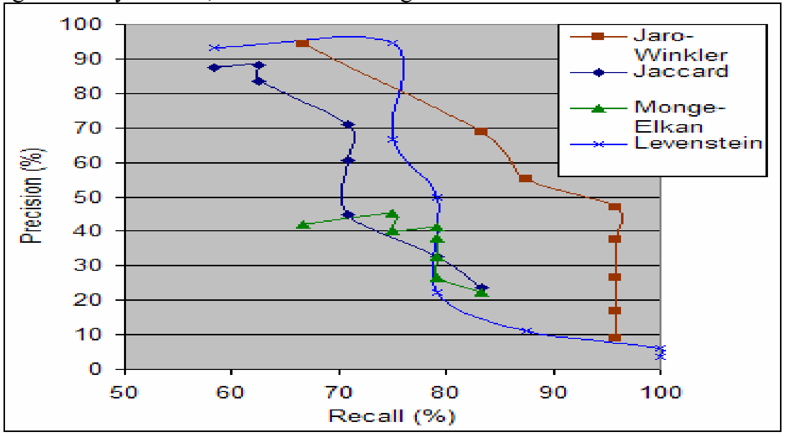


Fig. 5. Performance of various string similarity metrics for matching of classes in ontologies based on name.

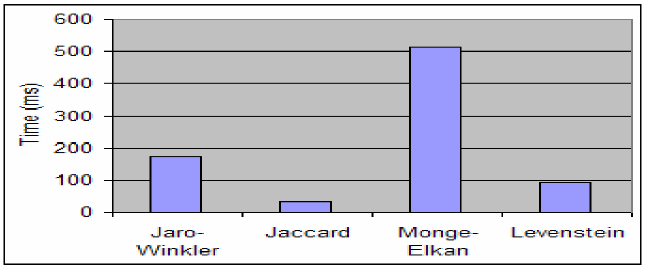


Fig. 6. Running time of computing the lexical similarity of classes using various string similarity metrics.

By using lexical similarity of classes, measured by the Jaro-Winkler similarity metric, 16 of the 24 corresponding classes could be identified (i.e. true positives - TP), as shown in Figure 7(a). Note that decreasing the detection threshold for lexical similarity metric would decrease the precision and increase the number of false positives (FP). To tackle this problem and find more matching classes (without changing the threshold), we also employed other similarity metrics namely, extensional, extensional closure and global path similarity metrics. Our experiments show that utilizing these additional metrics, help in finding more correct matching classes (true positives). At the same time, they do not introduce many false positives, as was the case with decreasing the detection threshold.

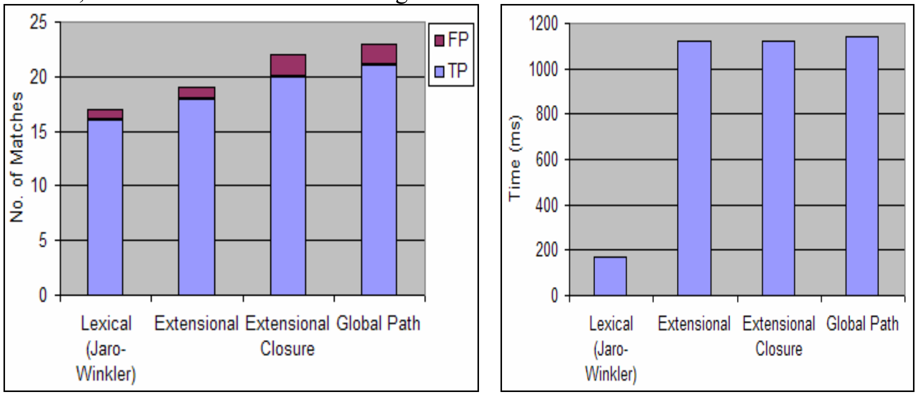


Fig. 7. (a) Detection of more matching classes using additional class similarity metrics, such as extensional, extensional closure, and global path. **(b)** Running time for computing lexical, extensional, extensional closure and global path similarity metrics.

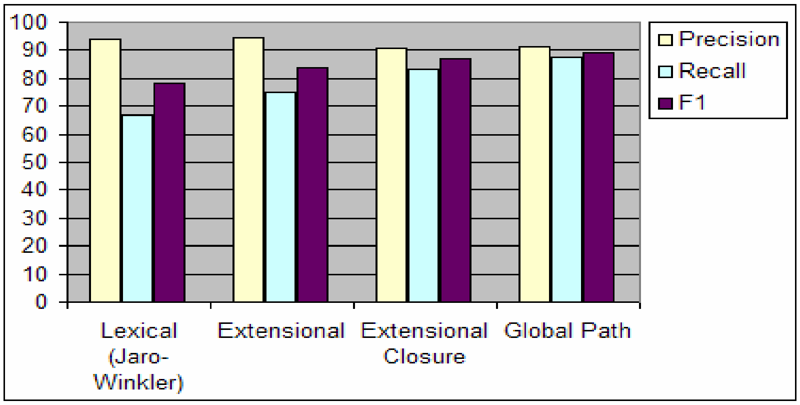


Fig. 8. Using extensional, extensional closure and global path similarity metrics, in addition to lexical, increases the recall and F1 quality measure.

Computing the lexical similarity of classes does not require reasoning. However, to compute the extensional, extensional closure and global path similarity metrics, the reasoner must be used. Basic reasoning services of an ontology were defined in section 3.2. Realization is used for finding the extension (instances) of a class for extensional similarity. Classification creates the class hierarchy, which is used in computing the path from the root of an ontology to any given class, and in fact fixes the global position of the class within a hierarchy. For computing the extensional closure, we need classify the ontology to find the subclasses and also do realization to retrieve the instances of all the subclasses. To perform reasoning, the ontology must be consistent and all classes must satisfiable. Hence, activating the reasoner in fact triggers all the above steps, and accounts for most of the running time. The time required for the rest of the computation, involving comparison of retrieved instances or comparing the classes in a path, is relatively small. As shown in Figure 7(b), the running time for computing the lexical similarity is small, in compare to the other three similarity metrics, which are approximately the same and require reasoning.

Figure 8 shows that by using the extensional, extensional closure and global path similarity metrics, in addition to lexical, the recall rate increases, while precision remains almost the same. This in turn implies that the F1 quality measure is increasing, as shown with the darker purple bar in Figure 8.

5. Related Work

Most of the solutions to ontology mapping produce a merged ontology as the final output [McG00, Noy03, Stu01, Udr07], and all this work is in the context of ontology development. Our work on ontology mapping for interoperability does not merge the ontologies. However, finding the matching classes is necessary in our approach, which is somewhat similar to the matching of various entities, in ontology merging. [Kal03] provides a good survey of various ontology mapping systems. A classification of different ontology matching techniques is provided in [Shv05]. Many systems look at finding lexical matches between ontologies and use dictionaries for this task [Wie99]. Chimaera is one of the early ontology merging tools, which considers structures such as subclass and superclass relations and slot attachments [McG00]. [Cal01] use a description logic based, model theoretic approach to integrating ontologies. [Noy03] provides interactive support for merging ontologies and uses the graph structure of ontologies to provide suggestions. [Stu01] uses the set of shared instances or the set of shared documents annotated with concepts of two ontologies and generates a lattice to relate the concepts of the ontologies using formal concept analysis. [Udr07] uses common instances and reasoning to extract the structure of the ontology, and then merges the two ontology graphs. Graph-based approaches have been utilized for schema matching in [Mil98, Mel02].

6. Conclusions and Future Work

The problem of ontology mapping has two distinct goals, namely ontology development and facilitating interoperability. The state of the art and current research in ontology mapping has been focused on ontology development and is rooted in the seminal work of [McG00, Noy00] in 2000. Clearly, *today*, providing interoperability between autonomous organizations is critical, considering the proliferation of the Web and the number of enterprises that use the Web infrastructure in their information systems and marketing schemes (e.g. Amazon, eBay). Unfortunately, the ontology mapping problem has rarely been studied in the context of facilitating interoperability.

In this paper, we identify the importance of this goal and provide an in-depth comparison to the information integration problem in databases. As a result, we distinguish the merging of ontologies, as an ontology development task [McG00]. Furthermore, class matching is emphasized, as opposed to the matching of other entities in an ontology. Various class similarity metrics, which are lexical, extensional, extensional closure, and global path similarity are proposed, and their time complexity are analyzed. Lexical similarity is measured using Jaro-Winkler, Jaccard, Monge-Elkan, and Levenstein string distance measures. Then, a class matching algorithm that utilizes these metrics is designed and evaluated experimentally. Note that this is one of the first attempts to solve the semantic heterogeneity problem in ontologies, with attention to facilitating interoperability, and much more work remains to be explored. We are currently working on the deployment of this design for interoperability, in real-world information systems that use ontologies.

Acknowledgement: The Swoop ontology editor and Pellet reasoner, which are open source tools, developed in the MINDSWAP Research Group at the University of Maryland, were used for the implementation of our ontology mapping system.

References

[Bat86] Batini, C., Lenzerini, M., Navathe, S.B., “A comparative analysis of methodologies for database schema integration”. *ACM Computing Survey*, Vol. 18(4), pp. 323–364, 1986.

[Cal01] Calvanese, D., De Giacomo, G., Lenzerini, M., “A framework for ontology integration”. *Proc. of the First Semantic Web Working Symposium*, pp. 303-316, 2001.

[Hal06] Halevy, A.Y., Franklin, M.J., Maier, D., “Principles of dataspace systems”. *Proceedings of Twenty-Fifth ACM Symposium on Principles of Database Systems (PODS’06)*, June 26-28, Chicago, Illinois, USA, pp. 1-9, 2006.

[Hal96] Halevy, A.Y., Rajaraman, A., Ordille, J.J., “Querying Heterogeneous Information Sources Using Source Descriptions”. *Proceedings of 22th International Conference on Very Large Data Bases (VLDB'96)*, September 3-6, Mumbai, India, pp. 251-262, 1996.

[Inr] INRIA ontology, http://fr.inrialpes.exmo.rdf.bib.owl.

[Kal03] Kalfoglou, Y., Schorlemmer, M., “Ontology Mapping: The State of the Art”. *Knowledge Engineering Review*, Vol. 18(1), pp. 1-31, 2003.

[Kal05] Kalyanpur, A., Parsia, B., Sirin, E., Cuenca-Grau, B., Hendler, H., “Swoop: A Web Ontology Editing Browser”. *Journal of Web Semantics*, Vol. 4(2), 2005.

[Kar] Karlsruhe ontology, http://www.aifb.uni-karlsruhe.de/ontology.

[Len02] Lenzerini, M., “Data Integration: A Theoretical Perspective”. *Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS’02)*, June 3-5, Madison, Wisconsin, USA, pp. 233-246, 2002.

[McG00] McGuinness, D.L., Fikes, R., Rice, J., Wilder, S., “An Environment for Merging and Testing Large Ontologies”. Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR’00), Breckenridge, Colorado, USA, 2000.

[Mel02] Melnik, S., Garcia-Molina, H., Rahm, E., “Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching”. *ICDE 2002*, pp. 117-128, 2002.

[Mil98] Milo, T., Zohar, S., “Using Schema Matching to Simplify Heterogeneous Data Translation”. *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB’98)*, 1998.

[Noy00] Noy, N.F, Musen, M., “PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment”. *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)*, Austin, TX, USA, July 2000.

[Noy03] Noy, N.F, Musen, M., “The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping”. *International Journal of Human-Computer Studies*, Vol. 59(6), pp. 983-1024, 2003.

[Noy04] Noy, N.F., “Semantic Integration: A Survey of Ontology-Based Approaches”. *SIGMOD Record*, Vol. 33(4), pp. 65-70, 2004.

[Rah01] Rahm, E., Bernstein, P.A., “A survey of approaches to automatic schema matching”. *VLDB Journal*, Vol. 10(4), pp. 334-350, 2001.

[Shv05] Shvaiko, P., Euzenat, J., “A Survey of Schema-based Matching Approaches”. Journal on Data Semantics, 2005.

[Sir07] Sirin E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., Katz, Y., “Pellet: A practical OWL-DL reasoner”. *Journal of Web Semantics* (To Appear), 2007.

[Stu01] Stumme, G., Maedche, A., “FCA-merge: Bottomup merging of ontologies”. *IJCAI 2001*, pp. 225-234, 2001.

[Udr07] Udrea, O., Getoor, L., Miller, R.J., “Leveraging Data and Structure for Ontology Integration”. *Proceedings of the 26th ACM SIGMOD International Conference on Management of Data (SIGMOD’07)*, 2007.

[Wie99] Wiederhold, G., Jannink, J., “Composing Diverse Ontologies”. *Proceedings of the IFIP Working Group on Database, 8th Working Conference on Database Semantics(DS-8)*, Rotorua, New Zealand, 1999.