Foundations of Data Interoperability on the Web: A Web Science Perspective

Hamid Haidarian Shahri
Department of Computer Science, University of Maryland, College Park, MD, USA
hamid@cs.umd.edu

ABSTRACT

In this paper, when we use the term ontology, we are primarily referring to linked data in the form of RDF(S). The problem of ontology mapping has attracted considerable attention over the last few years, as the deployment of ontologies is increasing with the advent of the Web of Data. We identify two sharply distinct goals for ontology mapping, based on real-world use cases. These goals are: (i) ontology development, and (ii) facilitating interoperability. We systematically analyze the goals, side-by-side, and contrast them for the first time. Our analysis demonstrates the implications of the goals on ontology mapping and mapping representation. Many studies on ontology mapping have focused on *ontology merging*. Ontology merging is an *ontology development* task (goal i). With the increase in the number of web-based information systems that utilize ontologies, the need for facilitating interoperability between these systems is becoming more visible (goal ii).

We show the consequences of focusing on interoperability with illustrative examples and provide an in-depth comparison to the information integration problem in databases. The consequences include: (i) an emphasis on class matching, as a critical part of facilitating interoperability, and (ii) an emphasis on the representation of correspondences, since the merging of ontologies is not suitable for interoperability. For class matching, various class similarity metrics are formalized and an algorithm which utilizes these metrics is designed. For representation, we present a novel W3C-compliant representation, named skeleton. An *algorithm* for creating the skeleton, for interoperability between ontologies, is also developed. Finally, we experimentally evaluate the effectiveness of the class similarity metrics on real-world ontologies.

Categories and Subject Descriptors
I.2.4 [Knowledge Representation Formalisms and Methods]: Semantic networks; D.2.12 [Interoperability]: Data mapping
General Terms
Algorithms, Experimentation, Standardization
Keywords
Data Interoperability, Linked Data, Web Science

1. INTRODUCTION

The need for communication between autonomous and distributed information systems is increasing with the wide usage of the Web. Nowadays, data sharing 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 last three decades have lead to improvements in this area. 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, in databases, it is very difficult to infer that *area* in one schema and *location* in another schema, refer to the same real-world entity. This is due to the fact that the semantics of attributes in the schema are not encoded explicitly, and the problem is referred to as semantic heterogeneity. Ontologies encode the specification of concepts more accurately, than relational schemas. The rich set of relationships defined between concepts in ontologies help in alleviating the semantic heterogeneity problem. However, since different ontologies exist and are being used by various autonomous organizations and user communities, it is necessary to find correspondences between the ontologies to facilitate interoperability.

As outlined in [Biz09], one of the major research challenges of linked data is to address the issue of schema mapping and data fusion, i.e. to retrieve data from different distributed sources of information and present it to the user. This requires a mapping of terms from different ontologies (vocabularies). In Section 3.2, we illustrate this problem with a precise use case, in which there are two independent universities. We demonstrate how users can query different ontologies (distributed and autonomous sources of information) and retrieve data from all of them, across organizational boundaries.

In this paper, when we use the term ontology, we are primarily referring to linked data in the form of RDF(S). As for the task of "finding correspondences between ontologies," we clarify how the task should be performed and how the correspondences should be represented, in different applications. We also clarify the relationship between "ontology merging" and "information integration." For example: (1) Ontologies should not be merged for *facilitating interoperability*. (2) Ontology merging should be used in the context of *ontology development*. And, the ontology merging process should be semi-automated (i.e. should involve a human in the loop). (3) The *owl:equivalentClass* construct, for the merging of concepts in two ontologies, is not generally applicable for facilitating interoperability.

This study revisits the ontology mapping problem in various settings to furnish generality, and at the same time adheres to real-world use cases. We systematically analyze the problem, by putting the problem in context and identifying the quite distinct goals of ontology mapping. The principal contributions of the paper are as follows:

- We provide a coherent and overarching definition of ontology mapping.
- Two goals for the ontology mapping problem are identified and clarified with real-world motivating examples. These goals are: (i) ontology development, and (ii) facilitating interoperability. Then, we provide a sharp distinction between the goals.
- We clarify the relationship between ontology merging and interoperability, and show that they should not be used interchangeably.
- Different implications of the goals are collectively analyzed. The implications include: representation, inconsistency, automation, class matching, and the relationship with the Semantic Web. These implications are important, since they

determine how ontology mapping should be performed and represented.

- Interoperability (goal ii) is highlighted as one of the major goals of the ontology mapping problem. Then, ontology mapping is revisited in the interoperability context, i.e. independent of the ontology merging context.
- The tight coupling between the interoperability goal and the Semantic Web vision is illustrated.
- Facilitating interoperability between ontologies is rigorously compared with information integration in databases. Based on this comparison, class matching is emphasized as a critical part of facilitating interoperability (goal ii).
- Various class similarity metrics for finding the matching (corresponding) classes are formalized. Then, a class matching algorithm, which utilizes these metrics, is designed.
- We present a novel W3C-compliant representation, named skeleton, to encode the correspondences between ontologies and facilitate interoperability between them. An algorithm for creating the skeleton is developed.
- We experimentally evaluate the effectiveness of the class similarity metrics on real-world ontologies.

2. ONTOLOGY MAPPING DEFINITION

In this section, we explicitly define the ontology mapping problem to avoid misinterpretations. The "ontology mapping" procedure for two separate and autonomous ontologies, O_1 and O_2, consists of the following steps:

- Step 1: *Finding* corresponding entities in ontologies O_1 and O_2.
- Step 2: *Representing* the found correspondences, and *using* it to achieve some goal.

For Step 1, the main ontology entities that can be considered, when finding correspondences between ontologies O_1 and O_2, are: classes (concepts), individuals (instances), and properties (relations). For Step 2, for using the found correspondences, the correspondences need to be represented in a suitable form.

Note that the goals of ontology mapping determine what candidates to consider, when we are finding the correspondences. The goal also determines how to represent the correspondences. This definition of ontology mapping is coherent and overarching, such that it encompasses the different goals of the problem.

3. GOALS OF ONTOLOGY MAPPING

Currently, there are various ontologies that are being used in different organizations. Often, they have been designed by different communities. Hence, there is a need for a mapping between these ontologies. Based on the definition of the ontology mapping problem (refer to Section 2), in order to lay out the foundations of the problem, we start with the goals of ontology mapping. We identify two quite distinct goals for ontology mapping, based on real-world use cases, and illustrate them with motivating examples.

3.1. Ontology Mapping for Ontology Development

Ontology is an abstraction for representing conceptual knowledge. All concepts are covered by the domain of human knowledge and these concepts are connected together in some fashion. Hence, it is very hard to limit an ontology in terms of what it should represent. This decision is usually made, based on the business needs of an organization, i.e. the ontology designer decides not to include some concepts, as they seem irrelevant to current organizational demands. Ontology design (also known as ontology development/engineering) is a complex and subjective issue, similar to database design, and requires a human in the loop.

Assume that an organization is currently using an ontology, *C*. Over time, as organizational models change, business processes evolve and are extended. Therefore the ontology *C*, which models the current business processes, also needs to be changed and often extended. Sometimes, the new business models (or some fragments of the changes) that are required in the current ontology have already been captured by ontologies that are being used in other organizations. In this case, the required extensions to the current ontology *C* are present in some other existing ontology, *E*. Now, the ontology designer of *C* needs to:

- Step 1: Find the correspondences between ontologies *C* and *E*
- Step 2: Decide on what concepts, instances and relations of the existing ontology *E*, need to be added to the current ontology *C*, based on the changes in the business model and the correspondences found in the previous step.

Note that the existing ontology *E* remains the same, while current ontology *C* will change (and be replaced in fact). This use case closely resembles the problem that has been analyzed in the context of merging/integrating two ontologies in the literature [McG00, Noy00, Stu01].

Motivating Example 1: Consider two organizations, Org_1 and Org_2, offering various products, and using two different ontologies, O_1 and O_2, shown in Figure 1. O_1 is shown with white rectangles, while O_2 is shown with grey rectangles. Some classes (i.e. concepts), namely *Sale Items* and *Videos*, in O_1 have corresponding classes (*Products* and *Movies*) in O_2. In Figure 1, since class *Videos* in ontology O_1 is defined in a 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 white and grey rectangles, as shown in Figure 1.

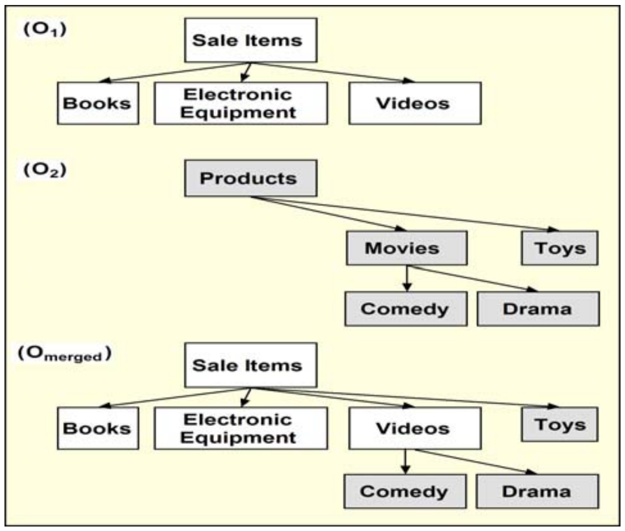

Fig. 1. Two ontologies O_1 and O_2, and the merged (integrated) ontology O_{merged}.

Note that in our scenario, the business model in Org_1, which was using ontology O_1, has changed. For example, Org_1 gradually needs to develop a more comprehensive ontology for its ecommerce operations. Therefore Org_1 will now be using the merged ontology O_{merged}, which is the result of extending O_1 with some existing ontology O_2. This is while Org_2 will keep using O_2 without any changes to its business model.

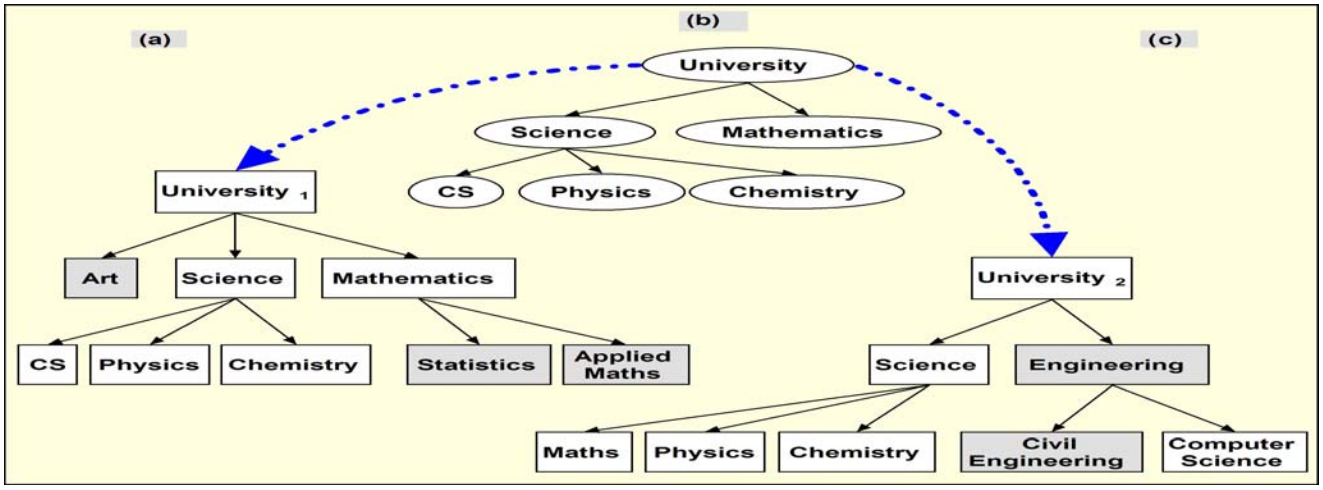

Fig. 2. Ontologies O_1 and O_2, which belong to two different autonomous organizations, are shown in (a) and (c). Skeleton *S*, connecting the ontologies, is shown in (b), in the middle. The concepts in ontology O_1 (shown in the figure) are the organizational units within *University*$_1$. The instances in ontology O_1 (not shown in the figure) are the courses that are offered by the organizational units within *University*$_1$. Each concept in skeleton *S* is connected to its corresponding concepts in the original ontologies O_1 and O_2, with a subclass relationship.

3.2. Ontology Mapping for Interoperability

Different enterprises use their own proprietary systems and are usually not willing to change their business models and operations. However, they also need to exchange information with other enterprises. Hence, interoperability between enterprises needs to be facilitated across organizational boundaries. That is, in many circumstances, users need to query different ontologies (distributed and autonomous sources of information) and retrieve data from all of them, as if all the information is residing in a unified source.

Let us define this scenario more formally. Two different ontologies, O_1 and O_2 are designed separately and are being used by two autonomous organizations, also known as parties. Each ontology 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, as we did for the merging use case in Section 3.1. To facilitate interoperability between the organizations in this scenario, two steps are required:

- Step 1: Finding the correspondences between ontologies O_1 and O_2
- Step 2: Representing the correspondences in a suitable structure, which we call skeleton *S*.

The skeleton is described using the motivating example, next.

Motivating Example 2: Consider two universities in which faculties, and departments within the faculties, are organized differently. Ontologies O_1 and O_2 are shown in Figure 2(a) and 2(c), respectively. Ontologies O_1 and O_2 represent the organizational hierarchy of *University*$_1$ and *University*$_2$, and are depicted with rectangles. There are six corresponding concepts in O_1 and O_2, namely: *University*, *Science*, *Maths*, *CS*, *Physics*, and *Chemistry*, shown with a white color. These six concepts appear in different places in O_1 and O_2. The skeleton *S* consists of these six concepts, as shown in Figure 2(b), and represented with ovals.

When creating a skeleton, first, we need to know the shape (i.e. class hierarchy) of the skeleton. The shape of the skeleton governs the relationship between the concepts in the skeleton. The shape of the skeleton is determined by the ontology of one of the parties (i.e. O_1 or O_2). In Figure 2, the shape of the skeleton is the same as ontology O_1. Each concept in skeleton *S* is connected to its corresponding concepts in the original ontologies O_1 and O_2, with a subclass relationship.

Note that Figure 2 shows such connections for the *University* concept, only. The *University* concept in the skeleton is connected to concepts *University*$_1$ and *University*$_2$ in ontologies O_1 and O_2, with blue dotted arrows. Other such connections are not shown in the figure for more readability. In Example 2, the ontology of each organization (i.e. O_1 and O_2) remains intact, unlike Example 1. There is no change in the business models (structures) of the universities, at all. However, both universities (parties) can be queried using the skeleton, which provides interoperability between them, as described next.

3.3. Contrasting the Goals at a Glance

In Sections 3.3 and 3.4, we clarify that interoperability may not always be facilitated by ontology merging. This clarification is a side effect of distinguishing the goals of ontology mapping. In principle, the two use cases in Sections 3.1 and 3.2 are very different. In Figure 2, the concepts in ontologies O_1 and O_2 are the organizational units within *University*$_1$ and *University*$_2$. Each concept contains various instances. The instances are the courses that are offered by an organizational unit (concept). For example, the *Computer Science* department (concept) in O_2 contains the courses (instances) that are offered in that department.

Here, we describe interoperability explicitly. In the interoperability use case, we would like to query for all courses related to computer science, and retrieve the results from both universities (i.e. across organizational boundaries). In Figure 2, with the skeleton, we can query for *CS* courses in ontology O_1, and using query expansion, we move to the corresponding concept in the skeleton (which is *CS*), and then also retrieve the relevant courses from the *Computer Science* concept in ontology O_2. Therefore, the query would return the results, as if all data resides in a unified source. In essence, the skeleton increases the recall of queries by enabling users to retrieve results from distributed sources, under a unified framework.

In Figure 2, let us assume (incorrectly) that we want to merge the ontologies (O_1 and O_2) to facilitate interoperability. Consider that course *abc* is offered in the *CS* department in O_1, while a different course, named *xyz*, is offered in the *Computer*

Science department in O_2. Merging of these two departments by stating that the two concepts (*CS* and *Computer Science*) are equal (similar to but not exactly like Figure 1, for the ontology development goal) would imply that instances of one concept are also a member of the other concept. In this example, after merging the *CS* and *Computer Science* concepts, the ontology reasoner would infer that course *abc* is a member of both *CS* department in $University_1$ and *Computer Science* department in $University_2$, and is offered by both departments. Also, course *xyz* is offered in both departments, which is obviously not correct.

Similarly, in the OWL language (W3C Recommendation), using the *owl:equivalentClass* construct for the merging of two concepts (*CS* and *Computer Science*), instead of creating a skeleton, for the purpose of interoperability is not acceptable for the same reason. In other words, stating that $Class_1$ and $Class_2$ are equivalent classes using *owl:equivalentClass*, implies that every instance of $Class_1$ is also a member of $Class_2$. This is a very strong statement, and not generally applicable for facilitating interoperability between two systems.

Additionally, in Figure 2, when facilitating interoperability between parties, the parties are autonomous, and the data in the ontologies are often separate. While the parties need a mechanism for querying, we can not change the ontologies (business models) of either party, as we did in Figure 1 for $Organization_1$ by merging. In Figure 2, ontologies O_1 and O_2, and skeleton *S* are isolated and being administered independently in different *namespaces* (refer to OWL terminology). The ontology of each party does not change at all, and the skeleton is created separately, to connect the existing parties.

Up to now, we have distinguished the two goals (i.e. ontology development vs. interoperability), and also clarified that ontology merging is for ontology development and may not always be used for facilitating interoperability. Notice that merging was originally proposed for ontology development (refer to our explanation of [McG00, Noy00] in Section 8).

3.4. Implications of Context on Ontology Mapping

Traditionally, ontologies have been used for creating intelligent/expert systems. Those systems were often deployed, by a limited number of experts, in constrained domains. The design of suitable ontologies in such systems required tools and expertise. As a result, the ontology development goal (Section 3.1) was at the focus of attention, e.g. [McG00, Noy00]. Nowadays, with the advent of the Semantic Web, the need for interoperability (Section 3.2) between systems/ontologies is becoming more visible.

In order to explore the requirements of the interoperability goal, we carefully probe the above use cases. This will illustrates how ontology mapping should be performed, to achieve interoperability. In this section, we study the ontology mapping goals, across different *dimensions*. The dimensions serve as a guideline for the ontology mapping task, and they influence the design of tools and algorithms for this task.

A. Representation: Obviously, the goals of ontology mapping should propel the solution forward. Based on the definition of ontology mapping provided in Section 2, Step 1 (i.e. finding correspondences) is similar for achieving either goal of ontology mapping. In Step 2, for representing the found correspondences between two ontologies, we need a suitable representation. The question of how to represent the correspondences can be studied more concretely, in light of the distinction that we made about the goals of mapping. For developing and merging ontologies, the merged ontology is in fact the representation for the found correspondences (refer to O_{merged} in Figure 1). This is similar to the work of [McG00, Noy00].

As described in Section 3.3, merging of ontologies does not create a suitable representation for facilitating interoperability between two systems, in the general case. For interoperability, we present a novel W3C-compliant representation, named skeleton, to encode the correspondences between ontologies and facilitate interoperability between them, as shown in Figure 2. In Section 6, we provide an algorithm for creating the skeleton and also examine why the skeleton is a suitable representation. In Figure 2, there is no merging involved, and O_1, O_2 and *S* reside in different namespaces.

B. Inconsistency: When merging ontologies for ontology development (Section 3.1), various inconsistencies can arise and the task involves complex decision making, since there may be various ways to avoid the inconsistencies. For example, in Figure 1, consider the class *Videos* in ontology O_1 and class *Movies* in ontology O_2, and instances of both these classes, which are movies, categorized using genres from the YahooMovies website. Consider that in ontology O_1, there is a cardinality restriction for instances of *Videos*, such that each instance of *Videos* has exactly one genre from the YahooMovies website. However, in ontology O_2, there is a cardinality restriction for instances of *Movies*, such that each instance of *Movies* has exactly two genres from the YahooMovies website. Now, if we merge the classes for *Videos* and *Movies* (as we did in Figure 1 for O_{merged}), it is not obvious how to handle this cardinality inconsistency. There are various options and the ontology designer has to make these decisions at design time, when developing the new ontology. Handling these complex issues is an integral part of the ontology development process, as outlined in the first goal.

For another example of inconsistency in Figure 1, assume that in addition to the *Toys* class which is a subclass of *Products* in O_2, the *Electronic Equipment* class in O_1 also has a *Toys* class as subclass. Now, the resulting merged ontology would have two *Toys* concepts, one of which is a subclass of *Sale Items* (shown in O_{merged} in Figure 1) and the other is a subclass of *Electronic Equipment* (not shown in O_{merged}). Even combining the two *Toys* concepts may not have the desired effect, since other conflicts could arise, similar to the cardinality problem, as mentioned previously. Additionally, the nature of the merging problem is such that the current ontology is not only being extended, but also needs to evolve, to accommodate the neighboring classes of the corresponding class in the existing ontology. For example in Figure 1, if the class *Movies* did not have a parent class, a simple extension would have sufficed, but now that it has a *Products* class as its parent, we must accommodate the *Products* class as well, when merging *Movies* into O_1.

Considering our small example in Figure 1 and the various inconsistencies that could arise from merging, it is obvious that ontology merging is usually not a *scalable* process and should be performed in the context of developing a new ontology, to meet the new business demands of an organization. It is certainly not suitable for creating a global system to facilitate interoperability between parties. On the other hand, using a skeleton for interoperability does not create such inconsistencies, since the ontologies of the organizations are kept separately in different namespaces, as shown in Figure 2.

C. Automation: For both goals of ontology mapping, Step 1 (i.e. finding correspondences), should have a human user in the loop (unless the results are approximate). Notice that the issue of representing the correspondences is dealt with separately, in Step 2, after the correspondences are given/determined.

As for Step 2 (creating a suitable representation for the correspondences), when merging ontologies for ontology development, there is a potential for inconsistencies to arise. Ontology design (similar to database design) involves subjective decisions. Hence, the merging process can only be "semi-automated" at best. Ontology merging algorithms should have a human user in the loop, as in the PROMPT Suite [Noy00]. Moreover, the process should be interactive, to allow the changes to the ontology to be verified at each step, by the human ontology designer. The designer should be familiar with ontologies and domain knowledge modeling.

On the other hand, when creating the skeleton representation for facilitating interoperability, as long as the correct set of corresponding concepts between the parties is given, as input, the skeleton can be created using a fully automated algorithm (refer to Section 6), since the process does not create inconsistencies.

D. Class Matching: In Step 1 of ontology mapping, when finding the correspondences between two ontologies, various entities in the ontology (e.g. classes, individuals, and properties) could be considered. For ontology merging, all entities are important for correspondences. In Section 4, we compare the information integration problem in databases to the interoperability goal in ontology mapping. The comparison shows that finding corresponding *classes*, is a critical part of ontology mapping for facilitating interoperability. However, this does not imply that matching of individuals is not important. In fact, matching of corresponding individuals provide auxiliary information for the ultimate task of class matching. Note that the skeleton representation (Figure 2) is actually geared towards capturing class correspondence, as well.

E. The Relationship with the Semantic Web: By carefully examining the design goals of the Semantic Web, we can conclude that there is a close relationship between the ontology mapping problem for interoperability and the Semantic Web vision. The design goals of the Semantic Web include [Hef04]: (1) Using shared ontologies, (2) Supporting ontology evolution, (3) Ontology interoperability, (4) Inconsistency detection across ontologies, (5) Balance of scalability and expressivity in creating ontologies, (6) Ease of use, (7) Compatibility with other standards, and (8) Supporting internationalization.

We will use the item numbers to refer to the eight goals, above. By focusing on the ontology mapping problem with an emphasis on interoperability (Section 3.2), as opposed to the ontology merging emphasis (Section 3.1), we also address the core design goals of the Semantic Web. Note that in Figure 2, allowing various autonomous (isolated) organizations/parties to create and adopt their own ontologies as a community, is in agreement with **goal 2** (supporting ontology evolution). Facilitating interoperability between these isolated parties using the skeleton is in agreement with **goal 3** (ontology interoperability). In Section 5, when presenting the class similarity metrics, we demonstrate how **goal 1** (using shared ontologies) supports the class matching process for interoperability. Altogether, this neatly ties the ontology mapping problem for interoperability to the first three design goals of the Semantic Web. The idea of having distributed modular ontologies (adopted by different communities), and providing links between these ontologies to support interoperability, is inline and tightly coupled with the spirit of the Semantic Web.

F. Discussion: Based on Section 3, it is clear that ontology development (as in ontology merging) and interoperability (i.e. information integration) are two separate goals of ontology mapping. We illustrated that ontologies should not be merged for facilitating interoperability. The comparison of the goals is vital, since it clarifies how ontology mapping should be performed and represented, in different applications.

The above use cases and analysis demonstrate that interoperability is an important goal of ontology mapping. By interoperability, we mean that users should be able to query distributed information sources, as if the data resides in a unified source. In Section 4, we focus on this goal; that is, independent of ontology merging. We treat the interoperability goal more thoroughly, by comparing it with the information integration problem in databases. The interoperability goal is quite similar to what the database community is trying to achieve in the context of information integration research and schema matching.

4. CLASS MATCHING: A CRITICAL PART OF FACILITATING INTEROPERABILITY

Based on the definition of ontology mapping (in Section 2), in Step 1, the correspondences between ontologies need to be determined, and the question is what candidates to consider, when finding correspondences between ontologies. Now that we have clarified the goals of ontology mapping (in Section 3), we are ready to tackle the question of what candidates to consider. For the ontology development goal and merging of two ontologies, all entities of an ontology (i.e. classes, individuals, and properties) are usually considered, so that the two ontologies can merged. In this section, we show that for the interoperability goal, finding the corresponding classes (i.e. class matching) is very critical.

Considering the similarity of the ontology mapping problem for facilitating interoperability and the information integration problem in databases, we analyze and compare them, in this section. The comparison illustrates that class matching is critical, when mapping between ontologies for facilitating interoperability. Note that since OWL is based on RDF, our discussion also applies to OWL. Additionally, we are not mapping between ontologies and databases, i.e. there is no transformation involved between the two models. We are only comparing the models to illustrate what should happen, when facilitating interoperability between ontologies. The problem of facilitating interoperability between databases has been studied for three decades in the context of information integration in the database community. Hence, our comparison provides insight and sheds light on the relatively newer problem of interoperability between ontologies.

4.1. Information Integration in Databases

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. A thorough theoretical study of the problem is presented in [Len02]. We base our discussion on [Doa01, Len02], while other interpretations may exist. The main idea is to provide a uniform query interface over a mediated schema. The 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.

Figure 3 shows an example of the information integration problem in databases. Here, the goal is to generate a mapping between columns (*Town* and *City*) in different local schemas (*S* and *T*), by mapping them to some global schema. The local schemas usually reside in separate autonomous data sources (*DataSource1* and *DataSource2*). Figure 3 illustrates one example mapping between schema *S* and schema *T*. More details about the information integration process in databases can be found in [Doa01, Len02]. This is a simple and important example that will be used later in this section to demonstrate how ontology mapping should be performed to facilitate interoperability, and how ontology mapping for interoperability relates to schema mapping (i.e. information integration).

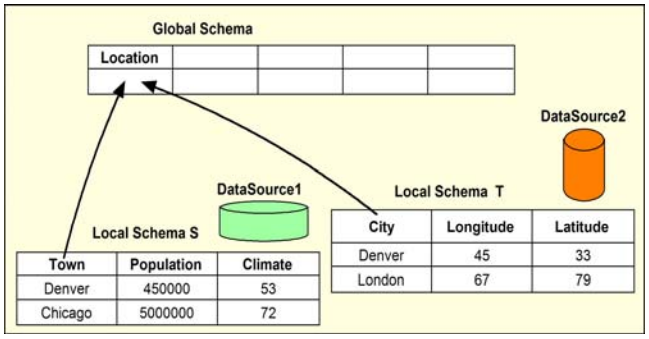

Fig. 3. Example of the information integration problem in databases. The goal is to generate a mapping between columns (*Town* and *City*) in different local schemas (*S* and *T*), by mapping them to some global schema. The local schemas usually reside in separate autonomous data sources (*DataSource1* and *DataSource2*).

4.2. Interoperability

Following the description of information integration in databases above, it is important to point out that, the term "integration" might be vague to some extent, since it might unintentionally be interpreted as some type of "merging" of schemas. This is not what actually occurs in databases, as the schemas in each local data source are handled autonomously and need to be kept separately. The local data sources are often administered by different organizations and are not merged (integrated). In fact organizations are not willing to change their business models and everyday operations.

The ultimate goal of information integration is to provide interoperability between various systems, which is the exact same goal that we identified in Section 3.2. The term "interoperability" is clearer for describing the motivations and objectives of the process. By analogy, there is no merging of ontologies involved in ontology mapping, when we are trying to achieve interoperability between organizations, which use different ontologies (refer to Section 3.2).

4.3. Expression of Simple Facts in the RDF Model

Simple facts in the RDF model indicate a relationship between two things. Such a fact may be represented as an RDF triple in which the predicate (i.e. property) names the relationship, and the subject and object denote the two things. Figure 4(a) shows the predicate *hasAuthor*, which is the relationship between the subject and the object. The subject is an instance of class *Book*, while the object is an instance of class *Author*. The classes are depicted as ovals. For example, *The Art of Computer Programming* (which is a book) hasAuthor *Donald Knuth* (who is an author).

In contrast with the relational database model (Section 4.4), 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. The use of Uniform Resource Identifiers (URIs) in the RDF model provides a very powerful mechanism for facilitating interoperability on the Semantic Web. Consider that the success and scalability of the current WWW infrastructure is a vivid illustration of the tremendous potential of the idea of using links or URIs (which seems like a simple idea at first glance).

4.4. Expression of Simple Facts in the Relational Model

A familiar representation of a fact in the relational model in databases is a row in a table. The terms row and table are also known as tuple and relation, respectively. A table has a number of columns (also known as attributes). Figure 4(b) shows the *hasAuthor* table. The table has two columns, namely *Book* and *Author*. A row for example indicates that, *The Art of Computer Programming* (which is a book) hasAuthor *Donald Knuth* (who is an author).

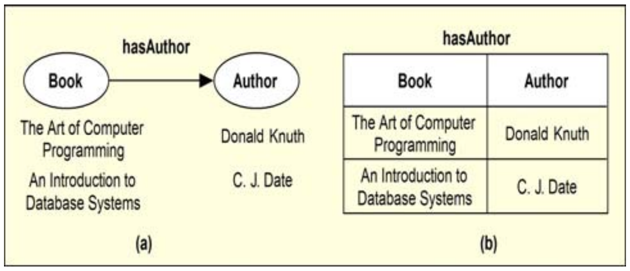
hasAuthor
Book
Author
The Art of Computer Programming
An Introduction to Database Systems
Donald Knuth
C. J. Date
(a)
hasAuthor
Book | Author
The Art of Computer Programming | Donald Knuth
An Introduction to Database Systems | C. J. Date
(b)

Fig. 4. Correspondence between the RDF and relational models. (a) The predicate *hasAuthor*, which is the relationship between the instances of class *Book* and the instances of class *Author*, in the RDF model. (b) The table *hasAuthor*, which has two columns, namely *Book* and *Author*, in the relational model.

4.5. The Analogy between the RDF and Relational Models

By comparing the explanations of the RDF and relational models above, and by observing Figure 3, we can infer that the classes *Book* and *Author* (in RDF) correspond to the columns *Book* and *Author* (in relational). The correspondence between classes and columns is an important one, and it will be used in Section 4.6. The correspondence is also depicted in Figure 5. The other substantial correspondence is between instances of a class in RDF, with column values in relational.

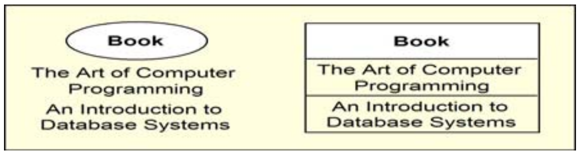
Book
The Art of Computer Programming
An Introduction to Database Systems
Book
The Art of Computer Programming
An Introduction to Database Systems

Fig. 5. The correspondence between classes (in the RDF model) and columns (in the relational model). There is also a correspondence between instances of a class (in RDF) and column values (in relational).

In the above discussion, the description of the relational model was constrained, such that a table only contained two columns. Now, we consider the general case, where a table contains more than two columns. In Figure 6(b), the table in the relational model has three columns, namely *Book*, *Author* and *Publisher*. Then, the RDF model would also have three corresponding classes, as shown in Figure 6(a). The correspondence between classes and columns still holds. It is essential to realize that the name of the table in the relational model is arbitrary. We used TableName in Figure 6(b). Additionally, two predicates, namely *hasAuthor* and *hasPublisher*, are now used in the RDF model (Figure 6(a)). The name of the two predicates in RDF is arbitrary and could be anything.

Notice that in Figure 4, we do not infer a correspondence between the name of the table (*hasAuthor* in relational) and the name of the predicate (*hasAuthor* in RDF), as this correspondence has no real substance. The comparison in Figure 6 actually eliminates the role of the table name in the relational model. Therefore, the substantial correspondence is between classes and instances in RDF, with columns and column values in relational, respectively. There is no correspondence for predicate (i.e. property) names in the relational model. This is due to the fact that in the RDF model, relations are encoded explicitly, in contrast with the relational model.

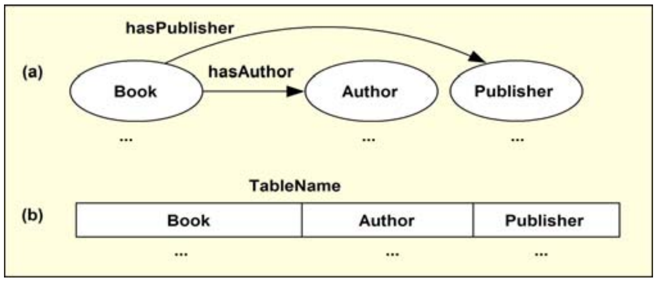

Fig. 6. A more general correspondence between the RDF and relational models. (a) The three classes in RDF are *Book*, *Author*, and *Publisher*. The name of the two predicates (*hasAuthor*, *hasPublisher*) in RDF is arbitrary and could be anything. (b) A table in the relational model, which has three columns, namely *Book*, *Author*, and *Publisher*. The name of the table (*TableName*) is arbitrary.

4.6. Class Matching: Why

Section 4.1 showed that in databases, the final output of the information integration process is a mapping between *columns* in different local schemas (refer to Figure 3). In Section 4.5, we illustrated that *columns* in the relational model correspond to *classes* in the RDF model (refer to Figure 4 and 5). Therefore, it is clear that to facilitate interoperability between ontologies, the *classes* in the ontologies need to be mapped to each other.

In RDF, the data is the instances of classes. The ultimate objective of interoperability is to *query* and correctly retrieve these data instances, across various ontologies. The data resides in the classes in the ontologies. Notice that as long as the correct mapping between the classes in the ontologies exists, users can query and correctly retrieve the data instances, across various ontologies. For example, in Figure 2, users would like to retrieve course instances from both *CS* class in *University*$_1$ and *Computer Science* class in *University*$_2$. By analogy, in the relational model, the data is the values in the columns. A correct mapping between the columns in the schemas enables users to correctly retrieve the data (column values) across various schemas.

Let us return to the question (raised in the ontology mapping definition in Section 2) that what candidates to consider, when finding correspondences between ontologies. Our detailed comparison in this section clarifies that a critical part of ontology mapping for facilitating interoperability is the matching of classes. The class matching objective directly facilitates interoperability. This is one of the critical implications of focusing on the context of interoperability in ontology mapping, as mentioned in Section 3.4.D. The class matching objective does not imply that other entities are not used for class matching. Matching of other entities is usually helpful for the matching of classes, as described in Section 5. Basically, the question of how to find the matching classes is a separate issue, addressed in the next section.

5. CLASS SIMILARITY METRICS AND CLASS MATCHING ALGORITHM

In the previous section, we identified that class matching is a critical part of ontology mapping for facilitating interoperability. In this section, we provide an algorithm for class matching (i.e. finding corresponding classes). The algorithm utilizes various class similarity metrics. Each of the class similarity metrics is formally defined. Additionally, the relationship between the matching of classes and the matching of instances is clarified.

Definition 1 (Mapping for interoperability): 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 $m : C_1 \otimes C_2 \rightarrow [0,1]$, where $C_1 \otimes C_2$ is defined as the set of all distinct unordered pairs of elements of sets C_1 and C_2, that is: $C_1 \otimes C_2 = \{(a,b) \mid a \in C_1, b \in C_2\}$.

Definition 2 (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 threshold t, then the classes match. We compute the similarity value by summing the result of the following four similarity metrics: lexical, extensional, extensional closure, and global path, which are defined, later. In order to compute the similarity value more effectively, a weighted sum of these metrics could also be used, or the result of each metric could be normalized.

As mentioned in Section 3.4.C, in regard to automation, the merging of ontologies for the ontology development goal can not be automated and needs a human user in the loop. On the other hand, ontology mapping for the interoperability goal consists of two steps (Section 3.2): finding the class correspondences (Step 1), and representing the class correspondences using a skeleton (Step 2). For Step 1, human judgment may be required for setting the threshold. For Step 2, Section 6 provides a fully automated algorithm for creating the skeleton.

While class matching can only be semi-automated, skeleton creation, which happens after a set of correspondences are given, can be fully automated. Section 5 provides the class matching algorithm, and Section 6 provides the skeleton creation algorithm. In principle, ontologies can cover any domain of knowledge, and the *nature of data instances* is extremely diverse in different applications. Hence, it is difficult to provide general guidelines on how to set the threshold for all ontologies/applications. Essentially, the threshold needs to be determined experimentally for each application and dataset.

Definition 3 (Lexical Similarity Metric): Let $s \in C_1$ and $t \in C_2$ be two classes in the ontologies. The lexical similarity metric is a function that assigns a real-valued number in the range of [0, 1] to the pair {s, t}, based on the closeness of the strings representing the names of s and t.

Definition 4 (Extensional Similarity Metric): Let $s \in C_1$ and $t \in C_2$ be two classes in the ontologies. The set of individuals which are direct members of s and t are represented as e(s) and e(t), respectively. The extensional similarity metric for s and t is computed as $|e(s) \cap e(t)| / |e(s) \cup e(t)|$. This is similar to computing the Jaccard similarity coefficient of two sets.

Definition 5 (Extensional Closure Similarity Metric): Let $s \in C_1$ and $t \in C_2$ be two classes in the ontologies. If class x is a subclass of class y, it is denoted as $x \sqsubseteq y$. The extensional closure of s, denoted as $e_c(s)$, is computed as $e_c(s) = \{\bigcup_{i \in C_1} e(i) \mid i \sqsubseteq s\}$. The extensional closure similarity metric for s and t is equal to $|e_c(s) \cap e_c(t)| / |e_c(s) \cup e_c(t)|$.

Instance Matching

Based on Definitions 4 and 5, instances within two classes need to be matched, i.e. duplicate instances should be identified. This task is necessary for both the ontology development goal and the interoperability goal. When merging ontologies for ontology development, duplicate instances in corresponding classes need to be detected and eliminated, so that the classes in the merged ontology would only contain unique instances. When performing class matching for facilitating interoperability, the instances of classes are not merged. Nevertheless, duplicate instances may need to be detected, in order to compute the intersection of instances of two classes correctly, when computing the extensional and extensional closure similarity metrics.

There are various approaches that can be used for instance matching. One simple approach is based on the assumption that if two instances in different ontologies are the same, they also use the same URI, which would help in identifying the instances uniquely. This assumption is usually not applicable in practical settings, since different organizations use different naming standards and URIs. Another approach, which we also use in our experiments, is to identify duplicate instances using approximate string matching techniques for the name of instances, similar to the lexical similarity metric used for the name of classes (Definition 3).

In a more complicated approach, the domain knowledge about instances and the facts stated in a knowledge base may also be used. As mentioned in Section 3.4.E, some of the design goals of the Semantic Web (SW) are quite beneficial for instance matching on the SW. Remember that design goal 1 was using shared ontologies. An example of a shared ontology is FOAF. If *T. B. Lee* and *Tim Berners-Lee* are two instances and both have the same *foaf:mbox* property value (i.e. email address), which is an *owl:InverseFunctionalProperty*, the ontology reasoner can infer that the instances are duplicates. In other words, ontology inference helps in identifying duplicate instances that may not be detected using approximate string matching. Notice that in this example, the use of shared ontologies (e.g. foaf) helps in instance matching, which is a part of the ontology mapping process. This approach, for instance matching on the Web, has important applications for facilitating interoperability in the Semantic Web vision.

Definition 6 (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 that starts from the root of an ontology and ends at s. The global path similarity metric for s and t is equal to the score assigned to the similarity of p(s) and p(t). The score is based on the lexical similarity of the classes that appear in the two paths.

Class Matching Algorithm

For Step 1 in Section 3.2, we now present our class matching algorithm, below. The Class-Matching algorithm exploits the class similarity metrics introduced in this section. Line 2 relates to Definition 3. Lines 3-5 compute the extensional similarity using the ontology reasoner, as in Definition 4. Lines 6-8 are based on the extensional closure similarity, as in Definition 5. Lines 9-11 compute the global path similarity, as in Definition 6. Line 12 computes the similarity value for classes and compares it to the threshold, as in Definition 2.

```
Class-Matching Algorithm
Input:
O1, O2: Original ontologies
C1: Set of classes in O1
C2: Set of classes in O2
Output:
M: Set of matching class
     pairs (c1, c2), s.t. c1∈C1, c2∈C2
1. for all c1∈C1, c2∈C2
2.    lexSim← lexicalSim(c1.name, c2.name)
3.    c1.ex← reasoner.Extensions(c1)
4.    c2.ex← reasoner.Extensions(c2)
5.    extSim← extensionalSim(c1.ex, c2.ex)
6.    c1.all← reasoner.AllExtensions(c1)
7.    c2.all← reasoner.AllExtensions(c2)
8.    extCSim← extensionalClosureSim(c1.all, c2.all)
9.    c1.path← reasoner.GlobalPath(c1)
10.   c2.path← reasoner.GlobalPath(c2)
11.   gpSim← globalPathSim(c1.path, c2.path)
12.   if ( lexSim+extSim+extCSim+gpSim > threshold)
        then M ← M ∪ (c1, c2)
13.end for
14.return M
```

6. SKELETON REPRESENTATION

The issue of representation is an important implication of focusing on the context of interoperability in ontology mapping, as mentioned in Section 3.4.A. Based on the definition of ontology mapping (in Section 2), in Step 2, the found correspondences between the ontologies need to be represented. Now, the question is how to represent them in a suitable format. For the ontology development goal, when integrating ontologies, the outcome of the process is one merged ontology. This merged ontology actually represents the correspondences between the ontologies. For the interoperability goal, we should not merge the ontologies. Instead, we provide a novel W3C-compliant representation, named *skeleton*, to encode the class correspondences between the ontologies, as shown in Figure 2. In this section, we provide an algorithm for creating the skeleton (refer to Step 2 in Section 3.2).

Sections 3.2 and 3.3 described how the skeleton facilitates interoperability between organizations. We also described the query expansion mechanism in ontologies for query answering. The skeleton represents the class correspondences between ontologies of organizations. These correspondences are essential for searching and query answering in the ontology reasoning process. The skeleton is a suitable representation, as it allows query answering over various ontologies (organizations). The skeleton increases the recall of queries by enabling users to retrieve results from distributed sources. Our design for the skeleton representation is compliant with the W3C recommendations. In other words, when using the skeleton, query answering and query expansion can be performed, using standard tools, without any ontology reasoner adjustments, and extra implementation effort. That is to say, all these issues are handled by the ontology reasoner, in a standard fashion.

To describe the *Skeleton-Creation* algorithm, below, we use the motivating example, depicted in Figure 2. The *Class-Matching* algorithm, introduced in Section 6, is a prerequisite for the *Skeleton-Creation* algorithm. The output of the *Class-Matching* algorithm is the set of matching class pairs (*M*) of the two ontologies. This set (*M*) is the input of the *Skeleton-Creation* algorithm. In lines 1-5, for each pair in set *M*, a class node is created in the skeleton. The name of the class in ontology O_1 is assigned to the class in the skeleton. The class in the skeleton is connected to the classes in the pair. In Figure 2, the classes in the skeleton are *University*, *Science*, *Maths*, *CS*, *Physics*, and *Chemistry*, which are connected to their corresponding classes in O_1 and O_2. Figure 2 shows such connections for the *University* concept, only, i.e. the *University* concept in the skeleton is connected to concepts $University_1$ and $University_2$ in ontologies O_1 and O_2, with blue dotted arrows. In line 6, the ontology reasoner infers the class hierarchy of ontology O_1. In line 7, this hierarchy is used for connecting the class nodes in the skeleton, to each other.

```
Skeleton-Creation Algorithm
Input:
O₁, O₂: Original ontologies
C₁: Set of classes in O₁
C₂: Set of classes in O₂
M: Set of matching class
      pairs (c₁, c₂), s.t. c₁∈C₁, c₂∈C₂
Output:
S: Skeleton
1.  for each pair (c₁, c₂) in M
2.    Create a class node s∈S
3.    s.name ← c₁.name
4.    Connect s to c₁ and c₂, using
         subclass relation
5.  end for
6.  H₁ ← reasoner.ClassHierarchy(O₁)
7.  Create the same class hierarchy
       as H₁, between all classes s∈S
8.  Return S
```

7. EMPIRICAL EVALUATIONS

In order to evaluate the effectiveness of the class similarity metrics, formalized in Section 5, we implemented these similarity metrics. In our implementation, Pellet was used for ontology reasoning. Pellet is an open source reasoner written in Java. The results of our experimental trials, reported here, are from two real-world ontologies. The ontologies were developed separately by different organizations. This dataset is selected from the datasets that are provided for the Ontology Alignment Evaluation Initiative.

When computing the lexical similarity metric for comparing the name of classes in two ontologies, various string similarity measures can be used. The results show that the performance of these measures varies considerably, as illustrated in Figure 7. The Jaro-Winkler measure shows a more robust behavior for finding corresponding classes in ontologies, based on class name. By decreasing the threshold in the class matching algorithm for each string similarity measure, the recall increases, the precision decreases, and various precision-recall performance levels are achieved, as shown by the plots on the curves, in Figure 7. Usually, there is a precision-recall tradeoff, and precisions below the 60 percent level are not very useful, since many of the detected matches would then be incorrect.

In Figure 7, 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 percent, by only decreasing the threshold, since this would cause a sharp drop in precision, i.e. introduce many incorrect results.

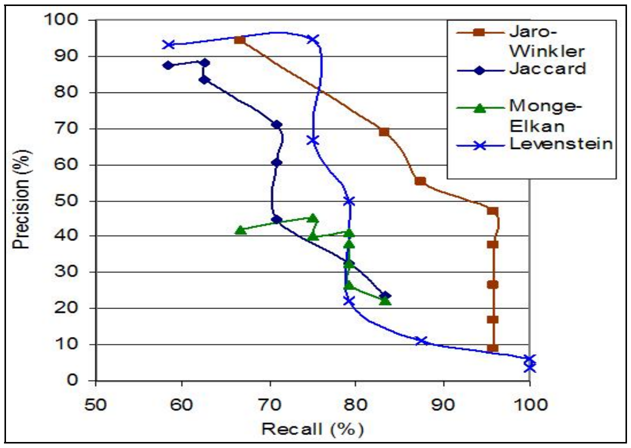

Fig. 7. Performance of various string similarity measures for finding corresponding classes in ontologies, based on name.

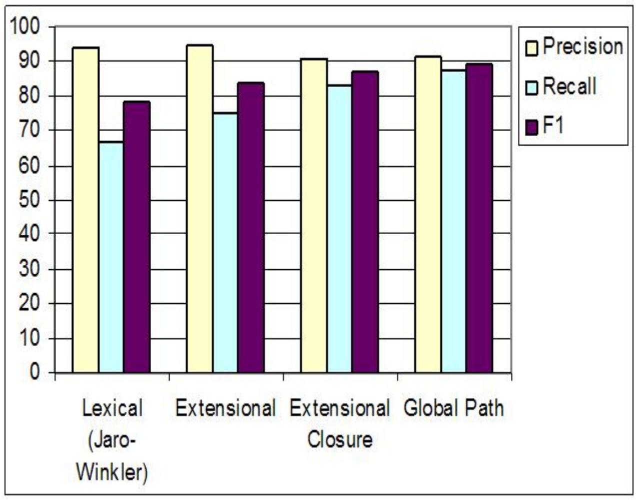

Fig. 8. Using extensional, extensional closure and global path similarity metrics, in addition to lexical, increases the recall and F1 quality measure.

The results in Figure 8 are cumulative from left to right, and each similarity metric is added to the previous ones. The precision and recall bars for Lexical (Jaro-Winkler) in Figure 8, are showing the same precision and recall values, as the first point on the Jaro-Winkler curve in Figure 7. Also, for all the experiments in Figure 8, we use the same threshold, as the first point on the Jaro-Winkler curve in Figure 7. Hence, in Figure 8, the threshold does not change, and there is no precision-recall curve (unlike Figure 7). Figure 8 shows that using the extensional, extensional closure, and global path similarity metrics, in addition to lexical, improves the recall. At the same time, in Figure 8, the precision remains almost the same.

Note that the recall can also be improved by decreasing the threshold in Figure 7 - however that considerably reduces the precision of results (as shown in Figure 7). In Figure 8, by using additional similarity metrics, we can improve the recall, without decreasing the threshold and losing precision. Therefore, we effectively overcome the precision-recall tradeoff, as evident by the increase in the F1 quality measure. This demonstrates that using the additional class similarity metrics helps in finding more corresponding classes and achieving better results.

8. RELATED WORK

[McG00] tackles ontology merging for government intelligence, where ontologies are developed by various teams. The teams are responsible for the development, design and maintenance of ontologies. These ontologies need to be integrated into other large application ontologies. [McG00] creates a tool called Chimaera for the above tasks. [Noy00] developed the PROMPT tool for merging ontologies. The goal was to reuse existing ontologies to help in ontology development efforts. Merging of ontologies causes inconsistencies and the user is interactively prompted with suggestions to remedy these inconsistencies. Both [McG00, Noy00] provide concrete ontology merging scenarios. Note that these works are in the context of ontology development (refer to Section 3.1). In both papers, there is no mention of interoperability or information integration (refer to Section 3.3).

[Stu01] uses the set of shared instances or the set of shared documents that are annotated with the concepts of two ontologies. Then, a lattice is generated to merge and relate the classes of the ontologies, using formal concept analysis. Various systems have studied finding lexical matches between ontology entities, or use dictionaries, WordNet and other resources for the matching process. Doan utilizes machine learning for ontology matching [Doa03]. It exploits a Naïve Bayes (NB) classifier to detect the similarity of classes in two ontologies. Tabulator is a nice generic data browser and editor for the Semantic Web. It provides a way to browse RDF data on the web, using outline and table modes [Ber06].

[van08] provides an interesting overview of the semantic interoperability problem and reviews some of the core issues involved. van Harmelen states that with the rapid growth of the Internet and the Web, more principled mechanisms to facilitate semantic interoperability (i.e. facilitate querying of data) across organizational boundaries have become necessary. He emphasizes that despite many years of work on the semantic interoperability problem, this old problem is still *open* and has acquired a new urgency, now that physical and syntactic interoperability barriers have largely been removed. Note that the skeleton addresses the semantic interoperability problem, i.e. the skeleton enables users to query the data across organizational boundaries. [Ala08] illustrates some of the advantages of Semantic Web technologies through pragmatic examples. [Cor10] presents an approach for achieving RDF data integration using SPARQL query rewriting. They use graph rewriting rules to create a new graph pattern that is expressed in the form of the target ontology, but maintains the intended semantics of the source ontology.

Some ontology mapping solutions produce a merged ontology as the final output, e.g. [McG00, Noy00, Stu01]. That line of work is generally in the context of ontology development. In our approach, when mapping between ontologies for interoperability, there is no merging involved. Our approach to facilitating interoperability is different from ontology merging solutions, since merging inherently follows a different goal (refer to Sections 3.3 and 3.4). In our approach, for facilitating interoperability between organizations, we find the class correspondences between the ontologies and then create a skeleton to represent these correspondences. To our knowledge, this is the first work that provides the required *algorithms* for this approach, with attention to the requirements of the Semantic Web, in a fashion that is compliant with the W3C recommendations.

9. CONCLUSION

In this paper, we explicitly defined the ontology mapping problem and systematically examined its components. We investigated the mapping problem across various dimensions and carefully distinguished its goals. Based on the goals, we addressed the questions of what candidates to consider for finding the correspondences, and how to represent the correspondences. We also illustrated the tight coupling between the interoperability goal and the Semantic Web vision. Furthermore, we rigorously compared the interoperability goal for ontology mapping with the information integration problem in databases.

ACKNOWLEDGEMENT: We would like to thank (in alphabetical order) Ron Alford, Philip Bernstein, Tim Finin, Hugh Glaser, Jennifer Golbeck, Christian Halaschek-Wiener, Jim Hendler, Vladimir Kolovski, Don Perlis and Taowei Wang, who contributed to this work in different ways. This material is based upon work supported in part by NSF Grant IIS0803739, AFOSR Grant FA95500910144 and ONR Grant N000140910328.

REFERENCES

[Ala08] Alani, H., Chandler, P., Hall, W., O'Hara, K., Shadbolt, N., Szomszor, M., "Building a Pragmatic Semantic Web," *IEEE Intelligent Systems*, 23(3), pp. 61-68, 2008.

[Ber06] Berners-Lee, T., et al., "Tabulator: Exploring and Analyzing Linked Data on the Semantic Web," *Proc. of Semantic Web User Interaction Workshop*, ISWC, 2006.

[Biz09] Bizer, C., Heath, T., Berners-Lee, T., "Linked Data - The Story So Far," *Int. J. Semantic Web Inf. Syst.*, 5(3), pp. 1-22, 2009.

[Cor10] Correndo, G., Salvadores, M., Millard, I., Glaser, H., Shadbolt, N., "SPARQL Query Rewriting for Implementing Data Integration over Linked Data," *Proc. of 1st Int. Workshop on Data Semantics (DataSem 2010)*, Switzerland, 2010.

[Doa01] Doan, A., Domingos, P., Halevy, A., "Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach," *SIGMOD*, 2001.

[Doa03] Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., Halevy, A., "Learning to Match Ontologies on the Semantic Web," *Very Large Databases Journal (VLDB Journal)*, Vol. 12, No. 4, 2003.

[Hef04] Heflin, J., "OWL Web Ontology Language Use Cases and Requirements," *W3C Recommendation*, February, 2004.

[Len02] Lenzerini, M., "Data Integration: A Theoretical Perspective," *PODS*, 2002.

[McG00] McGuinness, D., Fikes, R., Rice, J., Wilder, S., "An Environment for Merging and Testing Large Ontologies," *KR*, 2000.

[Noy00] Noy, N., Musen, M., "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment," *AAAI*, 2000.

[Stu01] Stumme, G., Maedche, A., "FCA-merge: Bottomup Merging of Ontologies," *IJCAI*, 2001.

[van08] van Harmelen, F., "Semantic Web Technologies as the Foundation for the Information Infrastructure," In van Ooster, P. (Ed.), *Creating Spatial Information Infrastructures*, Wiley, 2008.