Abstract

Relational Database Systems (RDBSs) are well-known and widely used in many organizations, however, semantic conflicts between the participating RDBSs must be resolved before data can be exchanged between them. Semantic resolution between the RDBSs is extremely difficult to address mainly because participating RDBSs are designed and built independently. Furthermore, individual RDBSs are likely to evolve over time and the changes must be reconciled dynamically. In this paper, we describe an approach to resolve the semantic conflicts between RDBSs automatically while allowing the individual RDBSs to evolve. Relational Database Ontology (RDBO) is created and used to ensure the semantic descriptions of the individual RDBSs are conformed to a set of vocabularies, structures, and restrictions. We show how a modified reasoning engine is used to validate and infer additional semantic relationships from the existing relationships. We also show how terms defined in different database ontologies are compared to each other semantically using semantic weights and our modified reasoning engine. As a result, RDBSs can interoperate with each other seamlessly and at the correct level of semantics defined in their ontologies.

1 Introduction

In general, each RDBS is designed to store a particular data set and is described by the terms commonly used and understood by its local users. Since the individual RDBSs are designed and built independently, many challenges arise when two or more RDBSs must interoperate with each other. One of the main challenges is semantic conflict, which include [12]: (i) naming conflict, such as homonyms and synonyms; (ii) generalization and specialization conflicts; (iii) atomic and composite conflicts; etc. Over the past two decades, many different approaches have been proposed in the literature for resolving the semantic conflicts between RDBSs. These different approaches, ranging from resolving the semantic conflict at the data level, schema level, and application level, all have two things in common: (i) no deduction of additional semantic relationships from existing ones; (ii) no mention of semantic comparisons between terms defined in different RDBSs. For example, (a) given “Term A is semantically equivalent to Term B” (denotes by $T_A \equiv T_B$) and $T_B \equiv T_C$. Previously we cannot derive $T_A \equiv T_C$ from $T_A \equiv T_B$ and $T_B \equiv T_C$; or (b) given $T_A$ and $T_B$ from two different systems, what can we say about $T_A$ and $T_B$? Is $T_A$ more general, more specific, or equivalent to $T_B$? Without some semantic reference, no such comparisons can be done.

Other approaches such as the unified global schema and multidatabase language approaches also have limitations. The unified global schema approach brought with it many challenges since the individual RDBSs are designed and built independently so it is difficult for them to agree on a single schema. The multidatabase language approach gives users the ability to perform queries across multiple databases but semantic reconcilations are left to the users to resolve manually.

Recently, motivated by the Semantic Web [1], some approaches make use of ontologies to resolve semantic conflicts between participating systems [9, 10, 16]. Weihua and Shixian [19] presented a layered model approach that combined agents and ontologies to address the semantic interoperability problem in large-scale environments. The model has three layers: (i) a syntax layer that deals with syntactic interoperability; (ii) a semantic layer that deals with semantic interoperability; and (iii) an agent layer that deals with information interoperability. Suwanmanee et al. [13] propose a mediator-wrapper approach with OWL ontologies to enable semantic interoperability between relational or object data sources. Each data source is described by an OWL ontology and it is assumed that these ontologies are created manually. Mappings between ontologies are defined manually and only atomic mappings are mentioned. The mediator uses the Racer reasoner engine [5] to check for consis-
ency of the “integrated” ontology and RICE (Racer Interactive Client Environment) to allow users to pose queries over the “integrated” ontology. These approaches can be applied to RDBSs but no standardization on how RDBS semantics are described and no semantic comparisons between terms are mentioned.

Similar to the web, semantic conflicts between RDBSs should be validated and resolved dynamically while allowing participating RDBSs to have full control over their data. In this paper, we describe an approach to resolve the semantic conflicts between RDBSs using database ontologies and a modified reasoning engine. Database ontologies provide the semantic information of the participating RDBSs and mappings between them indicate how data in different RDBSs are related to each other. The modified reasoning engine is used to: (i) compare and rank terms defined in different database ontologies; (ii) validate relationships or mappings between the related terms defined in different database ontologies to ensure that they are semantically and correctly stated; (iii) deduct additional semantic relationships from existing relationships. The modified reasoning engine carries out these operations automatically based on the semantic information provided. This enables RDBSs to interoperate with each other semantically and at the correct levels of granularity regardless of their structures and how their semantic information is stated.

The rest of the paper is organized as follows: Section 2 describes the general requirements for enabling semantic interoperability between RDBSs. Section 3 describes how RDBS semantics are presented while conforming to a standardized structure that supports reasoning. Section 4 describes how semantic mappings are defined between the individual RDBSs, shared domain ontologies, and shared global ontologies. This section also describes a mapping scheme that supports transformations/translations. Mappings with translations are necessary for data instances of concepts that are semantically the same but defined in different formats. Section 5 describes how mappings between ontologies are validated and compared using a modified reasoning engine called Pellet. Finally, Section 6 concludes the paper and provides a brief discussion of the future work.

2 Semantic Interoperability Between RDBSs

In general, semantic interoperability between RDBSs requires: (i) a common set of vocabularies and their semantic relationships and constraints for describing the RDBSs; (ii) standardized database ontologies that describe the semantics of the individual RDBSs; (iii) a set of mappings state the semantic relationships between the database ontologies (hence RDBSs); and (iv) a reasoning engine that validates, compares, and deduce semantic relationships between database ontologies. T-Box and A-Box reasoning (reasoning about concept definitions and their instances respectively) ensure inferences between input ontologies can be made automatically. Figure 1 shows our framework for enabling semantic interoperability between RDBSs. The following sections describe our framework in details.

3 Representing RDBS Semantics

Since there are many different types of DataBase Management Systems (DBMSs), a common language for representing the RDBS semantics is necessary. It is also important that the common language chosen is rich and expressive enough to accommodate the dynamic and heterogeneous nature of the individual RDBSs. OWL, Web Ontology Language [18], is designed for such purposes so we choose OWL as the common language for representing the RDBS semantics. Furthermore, a common way of describing the RDBSs in OWL is also needed because without it, different RDBSs are likely to describe themselves differently so it would be difficult to relate, validate, and compare the semantic definitions of terms defined and used in different RDBSs. Trinh et al. [14] have described such a common model in OWL called the Relational Database Ontology (OWL-RDBO). OWL-RDBO is a set of common vocabularies and their semantic relationships and constraints for describing RDBSs. OWL-RDBO preserves the underlying structural constraints of the RDBSs and guarantees user applications work with data instances that conformed to a set of vocabularies and structures. Using the common vocabularies defined in OWL-RDBO, database ontologies are created for the individual participating RDBSs (see Figure 1). Database ontologies can be created manually, however, even for the domain experts, this task can be error-prone and time-consuming. To ease the burden on users, we have developed a tool for generating and publishing database ontologies automatically from the metadata of the RDBSs while maintaining their structural constraints [15]. The generated database ontologies are instances of the OWL-RDBO and are independent of the underlying RDBSs they described. Database ontologies are necessary for a number of reasons: (i) they provide an explicit and common semantic description of the underlying RDBSs that both human and computers can understand; (ii) there are many existing domain ontologies available that database ontologies can map to and facilitate the data exchange with other existing systems; and (iii) there are many existing reasoning engines available that can be used to validate the semantic mappings between the database ontologies and to infer additional semantic relationships from existing ones. For example, the following OWL mapping states that the term Employee, a rdbo:Table object, is semantically equivalent to the term Staff in another database ontology with the namespace URI of cs:
Figure 1. A framework for semantic interoperability between the RDBSs.

If `cs:Staff` is not of type `rdbo:Table` then `Employee` and `cs:Staff` are not equivalent because they are not the same type. A database schema can be considered as an ontology for a mini-world but it lacks formal semantics and reasoning supports. For example, `umcs:Employee ≡ cs:Staff` and `cs:Staff ≡ cs:Employee` then by transitivity, `umcs:Employee ≡ cs:Employee`. Using the database ontologies and a reasoning engine, such conclusions can be extracted automatically from the knowledgebase of the reasoning engine. We will describe how this is done in Section 5.

4 Semantic Mappings Between RDBSs

Database ontologies and semantic mappings between them are the key to success for semantic interoperability between RDBSs. Semantic mappings between the database ontologies are stated independent of the underlying RDBS’s logical and physical structures so changes can be made to the underlying RDBS’s structures without effecting the semantic relationships between the database ontologies. Semantic mappings between the database ontologies can be defined in three ways [17]:

- between the individual database ontologies.
- between the individual database ontologies and shared domain ontologies.
- between the individual database ontologies, shared domain ontologies, and shared global ontology.

The first approach is flexible but the main drawback is there will be a large number of mappings between the database ontologies (i.e., each term in one database ontology must explicitly map to all other equivalent terms in other database ontologies) [6, 8]. Compared to the first approach, the second approach is better in the sense that shared domain ontologies are used so the number of mappings between the database ontologies is reduced. A shared domain ontology is an ontology that describes a set of common terms, properties, and their relationships used in a domain. Shared ontologies are created by their domain experts. The third approach reduces the number of mappings between the database ontologies even more since both shared domain ontologies and a global ontology are used. Similar to a shared domain ontology, a shared global ontology contains a set of common terms, properties, and their relationships agreed and shared by all participants. In our approach, mappings can be stated between the database ontologies, shared domain ontologies, and shared global ontologies (see Figure 1). Shared global ontology is created and agreed by experts in the community.

Semantic mappings between the ontologies can be created manually or semi-automatically as described by Doan et al. [3]. Given any two terms $T_i$ and $T_j$ in any two database ontologies, there are four types of semantic relationships between $T_i$ and $T_j$:

1. $T_i$ is equivalent to $T_j$ (denoted by $T_i \equiv T_j$)
2. $T_i$ is not equivalent to $T_j$ (denoted by $T_i \neq T_j$)
3. $T_i$ is a generalization of $T_j$ (denoted by $T_i \supseteq T_j$)
4. $T_i$ is a specialization of $T_j$ (denoted by $T_i \subseteq T_j$)

The first two relationships are obvious. Generalization and specialization describe the hierarchy structures in RDBSs. For example, if Employee IS A Person then
Figure 2. Three approaches of how ontologies can be used for enabling semantic interoperability between RDBSs (adopted from [17]): (a) mappings between the individual ontologies; (b) mappings between the individual ontologies and shared domain ontologies; (c) mappings between the individual ontologies, shared domain ontologies, and shared global ontologies.

Person is more general than Employee (denoted by Person ⊒ Employee).

Semantic mappings must be defined at both the conceptual and data levels and OWL supports mappings at both levels. Semantic mapping at the conceptual level maps one concept to another. A concept is a term or a phrase that is used to represent an object that exists in the real world, that has certain properties, and that is distinguishable from other objects. Similarly, semantic mapping at the data level maps an instance of one concept to an instance of another concept. The following describes how semantic mappings are defined at both levels.

4.1 Semantic Mappings At The Conceptual Level

Semantic mappings at the conceptual level state the semantic relationships between terms defined in different database ontologies. For example, the mapping:

\[ \text{umcs:lastName} \equiv \text{cs:lastName} \]

states the semantic equivalent between umcs:lastName and cs:lastName, yet, if the formats of umcs:lastName and cs:startDate are in short date format (i.e., MM/DD/YYYY) and long date format (i.e., MM DD, YYYY), then we either need to convert them to a common format or convert one format to the other before comparing their instances. This leads us to **semantic mappings with translations**, which is described next.

4.2 Semantic Mappings With Translations

Semantic mappings with translations state the semantic relationships at the conceptual level with one or more translational operations on the data instances of the participating term(s). Similar to mappings at the conceptual level, mappings with translations are defined in the form:

\[ \langle \text{uri:term}_i \rangle \equiv \text{uri:ℑ}(<\text{uri:term}_j>) \]

where \( \text{uri:ℑ} \) is an aggregate function that applies to the instances of its argument and can be provided as a Web Service. For example, to address the previous date mapping problem, we can define a mapping with translation as follows:

\[ \text{umcs:hiredDate} \equiv \text{ws:ℑ}_{\text{Long2ShortFormat}}(\text{cs:startDate}) \]

where \( \text{ℑ}_{\text{Long2ShortFormat}} \) is an aggregate function provided by the namespace \( \text{ws:} \) that takes a long date format and converts it to a short date format. With the translation provided, instances of the two dates are now in the same format and can be compared to each other. Semantic mappings
with translations do not state the equivalent relationships between the data instances. Instead, they state the equivalent formats of data instances that belongs to equivalent concepts. This ensures the same format is used when comparing data instances of equivalent concepts. Alternatively, if mappings are stated directly between the data instances, they may also include relationships that are not used in the same context. For example, short format instances for \textit{cs:birthdate} and \textit{cs:startDate} are both dates but they are not used in the same context so they should not be compared to each other.

Semantic mappings with translations can also be used to address composite/atomic mapping problems. For example, we can now define a mapping that merges \texttt{firstName} and \texttt{lastName} together and maps the result to \texttt{fullName}:

\[
\text{fullName} \equiv \text{ws:3_merge(name, ",", firstName, lastName)}
\]

where \texttt{ws:3_merge} merges instances of \texttt{firstName} and \texttt{lastName} together with the token "," separation. Likewise, we can also define a translational mapping that splits a term into multiple terms and maps one of the terms to another. For example, we can define a mapping that splits \texttt{fullName} into the two terms \texttt{first} and \texttt{last} and maps the term \texttt{last} to the term \texttt{cs:lastName}:

\[
\text{ws:3_project(last, ws:3_split(fullName, ",", firstName, lastName))} \equiv \text{lastName}
\]

where \texttt{ws:3_split} takes \texttt{fullName} and splits it into two terms based on the "," token and \texttt{ws:3_project} projects out only instances of the term \texttt{last}.

5 Semantic Mapping Validation and Ranking

Semantic mappings between the database ontologies state the semantic relationships between the RDBSs they described. However, before they are used, it is necessary to validate the mappings to ensure they are semantically correct. For example, if the terms \textit{cs:Employee} and \textit{cs:Staff} are disjoint from each other, then we cannot have another term that have both of these terms as generalization. It is also necessary to deduct additional semantic relationships from existing relationships between the database ontologies. This reduces the number of redundant and unnecessary mappings users need to define. For example, given the following mappings:

\[
\text{umcs:Employee} \equiv \text{cs:Staff} \quad \text{cs:Staff} \equiv \text{cs:Faculty}
\]

then, by the \textit{transitivity property}, it is true that:

\[
\text{umcs:Employee} \equiv \text{cs:Faculty}
\]

Previously, we could not deduct such relationships and they all have to be defined explicitly. Using a reasoning engine, we can derive such relationships automatically from the knowledgebase and there is no need for users to define such mappings. Alternatively, without using a reasoning engine, such mappings must be stated explicitly. Similarly, other properties such as \textit{functional property}, \textit{inverse functional property}, \textit{symmetric property}, etc. are all supported and can be used to derive additional relationships from existing ones. It is also necessary to be able to compare the semantics of terms defined in different database ontologies. For example, given the three ontology hierarchies in Figure 3 and the mappings defined between \textit{umcs:Faculty} and \textit{Professor} and \textit{Employee} and \textit{cs:Staff}, can we define an equivalent mapping between \textit{umcs:TeachingStaff} and \textit{cs:Lecturer}? Before we can do so we first need to compare the semantics of these two terms based on the semantics provided. We will show how this is addressed later using semantic weights.

Manual validation, deduction, and comparison of mappings between the database ontologies is extremely difficult if not impossible. We have modified a reasoning engine called \textit{Pellet} (version 1.3) to automate the validations, deductions, and comparisons of semantic mappings between ontologies. \textit{Pellet} supports both reasoning over the terminologies (TBox) and their instances (ABox) (see Figure 1). TBox and ABox reasoning ensure concept definitions and their instances are consistent. \textit{Pellet} generates a semantic knowledgebase automatically from the input ontologies and their semantic mappings. We then query the knowledgebase for relationships that are stated explicitly and those derived from the existing relationships. If \textit{Pellet} produces "clashes" for a relationship, then the relationship does not entail or follow from the semantic information in the knowledgebase. Alternatively, if \textit{Pellet} does not produce "clashes", then the relationship is consistent with the semantic information provided.

Comparing between terms defined in different database ontologies requires a quantitative measurement of the semantic relationships between terms. This measurement indicates how far or close two terms are to each other semantically. We refer to this the quantitative measurement as \textit{semantic weight}. Table 1 shows the semantic relationship types and their corresponding semantic weights. A weight of 1.0 means the two terms are semantically the same (i.e., they have the same intensional meaning). Alternatively, a weight of 0 means the two terms are not equivalent to each other. Finally, a weight of $1 - 0.1 \left\lceil \log(h) \right\rceil$ means one term is more generalized than the other (where $h$ is the height of the ontology tree).
In general, the semantic weight of any two terms \( \text{term}_i \) and \( \text{term}_j \) is calculated as follows:

\[
\text{weight}(\text{term}_i, \text{term}_j) = (1 - 0.1 \lfloor \log(h) \rfloor)^k
\]

where \( k \) is the height differences in the same generalization/specialization hierarchy path between \( \text{term}_i \) and \( \text{term}_j \). If the two terms do not share the same generalization/specialization path, then their semantic weight is 0.0 since they do not participate in a generalization/specialization relationship. The value \( 1 - 0.1 \lfloor \log(h) \rfloor \) is chosen for the generalization relationship because \((1 - 0.1 \lfloor \log(h) \rfloor)^k\) shows a slow linear decrease as \( k \) increases. This implies the more specialized a term is, the further away it is from its generalization. Castano et al. [2] used 0.8 as the weight for generalization/specialization relationships but this value is not a good value because \( (0.8)^k \) converges very quickly to 0. For example, since the height of the middle ontology tree in Figure 3 is 4, the generalization weight of any two terms is 0.9 (i.e., \( 1 - 0.1 \lfloor \log(4) \rfloor \)). Thus, weight(Person,Employee) is 0.9 since Person is a direct generalization of Employee. Similarly, weight(Person,FullProfessor) is 0.729 because:

\[
\begin{align*}
\text{weight}(\text{Person}, \text{Employee}) &= 0.9 \\
\text{weight}(\text{Employee}, \text{Professor}) &= 0.9 \\
\text{weight}(\text{Professor}, \text{FullProfessor}) &= 0.9 \\
\text{and} \\
\text{weight}(\text{Person}, \text{FullProfessor}) &= 0.9 \times 0.9 \times 0.9 = 0.729
\end{align*}
\]

To evaluate whether or not we can define a mapping between umcs:TeachingStaff and cs:Lecturer mentioned in the previous example, we first evaluate and assign semantic weights to the existing semantic relationships that relate umcs:TeachingStaff and cs:Lecturer. From Figure 3, starting with umcs:TeachingStaff, we derive the following semantic weights:

\[
\begin{align*}
\text{weight}(\text{umcs:Faculty}, \text{umcs:TeachingStaff}) &= 0.9 \\
\text{weight}(\text{umcs:Faculty}, \text{Professor}) &= 1.0 \\
\text{weight}(\text{Employee}, \text{Professor}) &= 0.9 \\
\text{and} \\
\text{weight}(\text{Employee}, \text{umcs:TeachingStaff}) &= 0.9 \times 1.0 \times 0.9
\end{align*}
\]

Similarly, starting with cs:Lecturer, we derive the following semantic weights:

\[
\begin{align*}
\text{weight}(\text{cs:Staff}, \text{cs:Lecturer}) &= 0.9 \\
\text{weight}(\text{cs:Staff}, \text{Employee}) &= 1.0 \\
\text{and} \\
\text{weight}(\text{Employee}, \text{cs:Lecturer}) &= 0.9 \times 1.0 = 0.9
\end{align*}
\]

Since Employee is the common generalization in the same hierarchy path for both umcs:TeachingStaff and cs:Lecturer, we compare their semantic weights with respect to Employee. According to the semantic weights calculated, weight(\text{Employee}, \text{umcs:TeachingStaff}) is 0.81 and weight(\text{Employee}, \text{cs:Lecturer}) is 0.9 thus we can conclude that umcs:TeachingStaff and cs:Lecturer do not have the same semantic granularity so an equivalent mapping should not be defined between them. The semantic weight function added to Pellet is described in Algorithm 5.1.

The weights calculated by the three update functions are accumulative. Initially, weights are set to a value of 1 and when the conditions are satisfied, Pellet updates the weights.
Algorithm 5.1 (Semantic Weight Calculation)

Input:
- domain ontology DO
- ontology O1
- ontology O2
- a list of mapping statements stmts between O1, O2, and DO

Output: a list of semantic weights added the input stmts

1. build knowledgebase KB from DO, O1, and O2
2. check KB for consistency
3. if not consistent, exit
4. for each statement st in stmts do
5. check satisfiability of st
6. unfold Ti and Ts to use only primitive T’s
7. if Ti is a subclass of Tj then
8. update_subclass_weight(Ti, Tj)
9. if Ti is equivalent to Tj then
10. update_equivalent_weight(Ti, Tj)
11. if Ti is disjoint from Tj then
12. update_disjoint_weight(Ti, Tj)
13. end for

according to their semantic weights.

6 Conclusion and Future Work

RDBSs are designed and built independently. Database ontologies provide standardized and semantic descriptions of the underlying RDBSs. Semantic mappings between the database ontologies and/or shared ontologies enable semantic interoperability between the RDBSs described by the database ontologies. Mappings between ontologies can be defined manually or semi-automatically and validation is necessary to ensure their correctness. Semantic comparison of concepts defined in different ontologies is also necessary because mappings with different granularity are possible and we want to be able to detect such such mappings. Manually validate and compare mappings is not the solution.

In this paper, we modified the Pellet reasoning engine and showed how it can be used to validate and compare mapping between ontologies automatically. The result ensures RDBSs interoperate with each other at the correct level of semantics regardless of how the individual RDBSs are constructed. Using a reasoning engine, such as Pellet, also has other advantages such as: (i) additional mappings can be deduced from existing mappings thus reducing the number of unnecessary and redundant mappings users need to define; (ii) when new mappings are added, the new knowledge is added automatically to the knowledgebase by the reasoning engine there is no additional work required.

In the future, we would like to investigate the following: (i) apply the semantic weight concept to the three reasoning engines RACER [4], KAON2 [7], and OWLJessKB [11] and compare their results with that produced by Pellet. Regardless of their implementation, the results produced should be based on the semantics; (ii) semantic mappings between ontologies are the key to enable semantic interoperability between systems. Mapping management between ontologies must be managed dynamically since evolutions of the individual RDBSs must be supported. We plan to associate contracts with mappings and use a shared memory model to manage semantic mappings between ontologies. This allows mappings to be added or removed dynamically.

References


