RDB2ONT: A Tool for Generating OWL Ontologies From Relational Database Systems

Quang Trinh, Ken Barker, Reda Alhajj
Advanced Database Systems and Applications (ADSA) Laboratory
University of Calgary,
Calgary, Alberta, Canada
{qtrinh, barker, alhajj}@cpsc.ucalgary.ca

Abstract

This paper describes a framework that uses the Semantic Web infrastructure to address semantic interoperability between relational database systems in large-scale environments and at multiple levels of granularities. Given a relational database system, we describe a formal algorithm to use the relational database Rs meta-data and structural constraints to construct its OWL ontology while preserving the structural constraints of the underlying relational database system. The generated ontology is described using and conforming to a set of vocabularies defined in an ontology that describes relational database systems on the web. Using this set of vocabularies guarantee that applications on the web can work with data instances that conformed to a set of known vocabularies and structures. Finally, we describe our prototype and how semantic conflicts are resolved between multiple relational database systems using the generated ontologies.

1 Introduction

Motivated by making the SW a reality, a large number of domain ontologies and collections of tools aimed at supporting the SW have been proposed and created over the last five years. For example, Protégé [9] is an ontology editor that allows users to create and customize ontologies; SWOOP [2] is a tool that uses a reasoner to detect errors in ontologies and provide detailed explanations of why the errors occurred; GLUE [3] is a system that uses multiple machine learning techniques to find mappings between ontologies; etc. However, even with the help of these tools, constructing ontologies describing the underlying participating systems manually is error-prone and time-consuming even for the domain experts. The problem is more obvious when there are a large number of participating systems and/or the underlying system is large and changes frequently. This is the problem for relational database systems and many legacy systems. Thus, a tool for generating and publishing ontologies dynamically from relational database systems is needed before database systems can dynamically integrate data semantically.

In this paper, we describe a tool called the Relational DataBase-to-ONTology (RDB2ONT) tool that generates and publishes OWL ontologies dynamically from the meta-data and structural constraints of relational database systems while preserving their structural constraints. We also describe how the generated OWL ontologies are used to resolve some common semantic conflicts between database systems. The contributions of this work are as follows: (i) the meta-data of the underlying database systems can be expressed at multiple levels of granularities in the generated ontologies thereby enabling database systems to interact with each other and/or other participating systems semantically and automatically at multiple levels of granularities; (ii) the generated database ontologies provide a standardized and meaningful way to describe the underlying relational database systems and can be used to “bridge” the semantic gaps between the underlying relational database systems and/or other participating systems; and more importantly, (iii) the generated ontologies can also be used to “bridge” the semantic gaps between the underlying relational database systems and existing and well-known domain ontologies thereby enabling the underlying relational database systems to seamlessly interoperate with a much larger number of systems and applications.

The rest of the paper is organized as follows: Section 2 describes a set of OWL vocabularies, their semantic relationships, and restrictions for describing relational database systems on the SW. We called this the OWL Relational DataBase Ontology (OWL-RDBO). This is necessary for two reasons: (i) it guarantees applications on the SW can work with data instances that conform to a set of known
vocabularies and structures; (ii) it enables type-validations of the data instances. For example, any OWL-capable applications can conclude that company:Employee and rdbo:Employee (where company and rdbo are namespaces) are two different things since they have two different namespaces (i.e., one is a concept representing an employee definition in a company and the other is a concept representing an employee relation in a relational database. Section 3 provides a general framework for semantic interoperability between relational database systems on the web. Section 4 describes the RDB2ONT tool and a formal algorithm for constructing OWL ontologies from the relational database’s meta-data and structural constraints while preserving their underlying structural constraints. Preserving the underlying structural constraints is important because it preserves the integrity of the data in the underlying database systems. This section also discusses and gives examples of how one-to-one, one-to-many, and many-to-many relationships between relations in relational database systems are described in OWL. Section 5 describes the two mapping schemes for addressing the domain and composite/atomic conflicts between ontologies. Section 6 briefly describes our prototype and Section 7 discusses the future work and concludes the paper.

2 Relational Database Ontology (OWL-RDBO)

The OWL-RDBO describes relational database systems in OWL in a standardized way that can be semantically interpreted by any OWL-capable applications and reasoners. The OWL-RDBO has a prefix of rdbo and a physical namespace of http://white.cpsc.ucalgary.ca/ontologies/rdbo/rdbo.owl#. We only briefly describe the OWL-RDBO here as it appears elsewhere [13] but in brief the OWL-RDBO contains:

(i) vocabularies for describing relational database systems on the web such as: rdbo:DatabaseName, rdbo:Schema, rdbo:Relation, rdbo:Table (semantically equivalent to rdbo:Relation), rdbo:Attribute, rdbo:PrimaryKeyAttribute, rdbo:ForeignKeyAttribute, etc.;

(ii) semantic relationships between the vocabularies defined in (i) such as: rdbo:hasNamespace (each relational database has a namespace, which uniquely identify a database), rdbo:hasRelations (the list of relations in a relational database), rdbo:hasAttributes (the list of attributes in each relation), rdbo:primaryKey (the list of attributes that made up the primary key of a relation), rdbo:hasType (each attribute has a type), rdbo:isNullable (indicates whether or not an attribute is nullable), etc.; and

(iii) restrictions on the vocabularies and their semantic relationships such as: each database has exactly one namespace, each relation has zero or more attributes, each attribute has exactly one type, etc.

3 A Framework for Semantic Interoperability Between Relational Database Systems

Before discussing the details of our algorithm and methodology, a framework is presented in Figure 1 to base the discussion. Figure 1 shows a framework for enabling the semantic interoperability between relational database systems in large-scale environments. There are two databases (Databasej and Databasek) and each is described by a database ontology (OWL Database Ontologyj and OWL Database Ontologyk). These two database ontologies are generated automatically by our RDB2ONT tool and conform to the OWL-RDBO described in Section 2. The OWL domain ontologies are generally created individually and manually by the domain experts, which describe the concepts, properties, and their semantic relationships used in the individual domains. OWL domain ontologies are necessary because they reduce the number of semantic mappings between the individual database ontologies within in the domain (possibly from the potential n² mappings to n). Conversely, without domain ontologies, semantic mappings between the individual database ontologies can be peer-to-peer (i.e., n²) [5, 6]. Semantic mappings between ontologies can be established manually (i.e., define manually by domain experts) or semi-automatically (i.e., use tools to suggest possible mappings which are then validated by the domain experts [3, 7, 10]). For example, consider the following OWL statement that maps "Employee, a relation in some database ontology, to Staff, a relation in a Human Resource (HR) database ontology with namespace of http://a.b.c/HR#" - any OWL-capable application can validate this statement, Since the two relations have the same intensional meaning, their tuples can be combined together even if they are not the same structurally (more examples will be given in Section 5).

4 The RDB2ONT Tool

The RDB2ONT tool has two major components: the OWL Builder and the OWL Writer. The following describe the functionalities of these two components and a formal algorithm used by the RDB2ONT tool to generate OWL ontologies from the underlying relational database systems.
4.1 The OWL Builder

Given a relational database system, the OWL Builder extracts the meta-data and structural constraints from the relational database system, builds an internal common model, and generates an OWL ontology describing the underlying database system. The OWL Builder uses the Java JDBC/ODBC API to extract the meta-data and structural constraints (i.e., relations, attributes, primary keys, foreign keys, cardinalities and participation, etc.) and the Web Ontology Language (OWL) to construct the generated ontologies. The generated ontology is an instance of the OWL-RDBO described in Section 2. The OWL Builder uses the algorithm outlined in the next section to construct the OWL ontologies. This algorithm preserves the structural constraints of the underlying relational database systems by transforming their structural constraints into OWL restrictions and includes them in the generated OWL ontologies.

4.2 The RDB2ONT Algorithm

The following describe the formal steps of the algorithm used by the RDB2ONT tool to generate OWL ontologies from the meta-data and structural constraints of the relational database systems.

Step 1: Generate the instances of the types rdbo:DatabaseName and rdbo:RelationList that describe the relations in the input relational database system. That is, generate the list of relations in the input relational database and add them as instances of the type rdbo:Relation in the type rdbo:RelationList. For example, consider a relational database called dbOne with two relations as shown in Figure 2. Figure 3 shows the OWL ontology describing the list of relations in dbOne with namespace of http://localhost/dbOne.owl, which can be changed by users when exporting ontologies. Lines 6 to 18 list the list of relations in dbOne. Note also that we are reusing the list definition from the DAML Ontology Library (Line 15).

Step 2: Generate the instances of the types rdbo:Relation and rdbo:Attribute to represent the individual relations and attributes in the relational database. That is, for each relation $R_i$ in the relational model, create the instance $R_i$ of type rdbo:Relation to represent the relation $R_i$. For each attribute $A_j$ in relation $R_i$, create the instance $A_j$ of type
Figure 3. An OWL ontology describing the database dbOne.

rdbo:Attribute and $A_j$’s data type as an instance of type rdbo:hasType to represent the attribute $A_j$. Furthermore, if $A_j$ is a foreign key attribute that references another attribute, say $A_j'$ in relation $R'$ then we also need to include this information with the attribute $A_j$. Thus, (i) create the instance $A_j'$ of type rdbo:referenceAttribute and adds it as an inner element of the attribute $A_j$; (ii) create the instance $R'$ of type rdbo:referenceRelation and add it as an inner element of the attribute $A_j$. These two inner elements represent the foreign key information in the input relational database. Foreign key constraints in the relational model will be preserved in Step 4. For example, Figure 4 shows the OWL description of the relation Employee as shown in Figure 2 in a database called dbOne. Similar to the OWL description of the database, Lines 6 to 32 list the attributes in the relation Employee and Lines 35 to 56 describe the individual attributes and their characteristics in the relation Employee. More specifically, (i) Lines 43 to 46 stated that the attribute middleName is nullable (as indicated in Figure 2) and its type is a string; and (ii) Lines 50 to 55 describe the foreign key attribute PayDepositAccount, which references to the attribute ID in the relation Account.

Step 3: Generate the restrictions for instances of type rdbo:Attribute that are identified in Step 2. For each attribute $A_j$ in relation $R_i$ in the relational model, if $A_j$ is not a foreign key attribute and $A_j$ is nullable, then create a restriction for the instance $A_j$ of type rdbo:Attribute with the minimum and maximum cardinality restrictions of $0$ and $1$, respectively. Alternatively, if $A_j$ is not a foreign key attribute and $A_j$ is not nullable, then create a restriction for the instance $A_j$ of type rdbo:Attribute with the cardinality restriction of $1$. Repeat this step for all attributes in all relations in the input relational database. For example, consider the two attributes firstName and middleName in Figure 2.

Since firstName is not nullable, its cardinality restriction is $1$ (Lines 1 to 4 in Figure 5). Similarly, since middleName is nullable, its minimum and maximum cardinality restrictions are $0$ and $1$, respectively (Lines 5 to 9 Figure 5).

Step 4: Generate the restrictions for instances of type rdbo:Attribute generated in Step 2 that references other instances (i.e., restrictions for foreign key attributes). For each attribute $A_j$ in relation $R_i$ in the relational model, if $A_j$ is a foreign key attribute that references attribute $A_j'$ in relation $R'$, then create a restriction for the instance $A_j$ of type rdbo:Attribute with the values of the minimum and maximum participation of the relation $R_i$ with respect to relation $R'$ as the minimum and maximum cardinality restrictions. In addition, if the attribute $A_j$ in relation $R_i$ references attribute $A_j'$ in relation $R'$ then $A_j$ and $A_j'$ are the same semantically. This is implicit in the relational model and true for all foreign key attributes and their referenced attributes so these semantic relationships must be included in the generated OWL ontologies.

The following two examples illustrate the scenarios involving attributes in the underlying relational database systems that defined the relationships (i.e., one-to-one, one-to-many, and many-to-many) among the relations and their how instances and OWL restrictions are then generated to preserve the constraints in the underlying relational model.

Example 1: Consider the one-to-one “PayDepositAccount” relationship between the relations Employee and Account in Figure 2. This relationship states that each employee can setup his/her pay direct deposit to exactly one account and that each account belongs to exactly one employee. Lines 4 to 8 in Figure 6 indicate the participation between PayDepositAccount and ID is exactly one. Thus, every PayDepositAccount is associated with exactly one ID. Lines 11 to 21
Figure 4. An OWL description of the relation Employee in the database dbOne.

indicate: (i) PayDepositAccount has the same intensional meaning as ID in Account; (ii) PayDepositAccount is not nullable and its data type is integer; and (iii) PayDepositAccount is a foreign key attribute that references ID in Account.

Example 2: Consider the one-to-many relationship between the relations Dependent and Employee in Figure 7. This relationship states that each dependent must be a dependent of an employee and each employee can have zero or more dependents. Lines 4 to 7 in Figure 8 indicate the minimum participation between ID and EID is zero. Thus, every ID can be associated with zero or more EID. In the case where the relationship has total participation, the minimum cardinality value on Line 6 is 1. Line 11 indicates ID has the same intensional meaning as EID in Dependent. Lines 12 to 14 indicate ID is not nullable and its data type is integer.

Similarly, the interpretation and OWL description of a many-to-many relationship is similar to that of a one-to-many relationship shown in Figure 8 since a many-to-many relationship is equivalent to two one-to-many relationships combined.

4.3 The OWL Writer

The main function of the OWL Writer is to write the generated OWL ontologies to the file system. To give users maximum flexibility, they can choose namespace URIs for their generated ontologies and locations on the file system.
where the generated ontologies are to be written. Once written, the namespace URIs of the generated ontologies can be distributed so that the generated ontologies describing the underlying relational database systems can be used by applications and systems on the web.

5 Semantic Mapping Schemes and Semantic Interoperability

Ontologies and semantic mappings are the key to success with semantic interoperability. How semantic mappings are defined is not in the scope of this paper so interested readers are referred elsewhere [3, 4, 10]. For our purpose, we assume that semantic mappings between our generated database ontologies and other ontologies are defined either manually by the domain experts or semi-automatically [3]. OWL uses the construct owl:sameAs for stating semantic mappings between ontologies [11] but for simplicity, we will use the short notation \( ur_i:term_i \rightarrow ur_j:term_j \) to represent the owl:sameAs construct (where \( ur_i \) is the source URI and \( ur_j \) is the target URI). For example, the owl:sameAs statement in Section 3 can be written in our short notation as:

\( \text{eng:Employee} \rightarrow \text{hr:Staff} \)

where eng and hr are namespaces of the two ontologies containing the terms Employee and Staff. As an example, if the relation Employee has the three attributes firstName, lastName, and SSN and the relation Staff has the four attributes ID, EName, ssnNumber, and age then the following segment shows a set of data instances extracted and combined semantically from these two relations:

\[ 
\begin{align*}
\text{eng:Employee} & \rightarrow \text{hr:Staff} \\
\text{eng:firstName} & \rightarrow \text{hr:EName} \\
\text{eng:lastName} & \rightarrow \text{hr:SSN} \\
\text{eng:SSN} & \rightarrow \text{hr:ssnNumber}
\end{align*} 
\]

Note that even when the two relations are structurally different from each other, namespace URIs can still be used

\[ \text{uri}_i: \text{term}_i \rightarrow \text{uri}_j: \text{term}_j \]
to indicate the relations to which the data instances belong. It is also worth noting that the same person is known as “John Smith” in one relation and “J. Smith” in another relation. Given this data segment, any OWL-capable applications can use the OWL-RDBO and the generated database ontology to process the integrated result (e.g., removing redundancies, etc.) and customize the integrated data for their applications. As shown in the previous examples, semantic mappings identify semantic relationships between terms but terms that are semantically the same can be described differently. For example, (i) length in meters in one ontology maps to length in inches in another ontology (aka a domain conflict); or (ii) EName in one ontology maps to firstName and lastName in another ontology combined together (aka composite/atomic conflict). OWL does not define transformations on the mappings and data instances [12] so we have defined the following two mapping schemes to address this problem: (i) direct mapping scheme; and (ii) translational mapping scheme.

- A direct mapping scheme maps terms directly to other terms. Each term can have one or more synonyms so it can map directly to one or more terms and it can be mapped to one or more terms. Formally, a direct mapping scheme has the form:

  \[
  \text{uri}_s : \text{term}_i \rightarrow \text{uri}_t : \text{term}_j
  \]

  where \( \text{uri}_s \) and \( \text{uri}_t \) are source and target namespaces.

- A translational mapping scheme is a mapping scheme that requires some translations before terms can be mapped to each other. Formally, translational mapping scheme have the following two forms:

  \[
  \text{uri}_s : \text{term}_i \rightarrow \mathcal{M}(\text{uri}_t : \text{term}_j)
  \]

  \[
  \mathcal{M}(\text{uri}_s : \text{term}_i) \rightarrow \text{uri}_t : \text{term}_j
  \]

  where \( \mathcal{M} \) is an aggregate translational function that applies to instances of its argument before the instances can be compared to each other. The following examples illustrate how the translational mapping scheme works.

- For domain conflicts such as length in meters maps to length in inches, we can use either form to resolve the domain conflicts. For example, we can use the form:

  \[
  \text{uri}_s : \text{length} \rightarrow \mathcal{M}_{\text{Meters2Inches}}(\text{uri}_t : \text{length})
  \]

  where \( \mathcal{M}_{\text{Meters2Inches}}(\text{uri}_t : \text{length}) \) is an aggregate translational function that converts instances of length in meters to inches.

- For composite/atomic conflicts such as EName maps to firstName and lastName combined together, we can use the following two aggregate translational functions for resolving the composite/atomic conflicts:

  * \( \mathcal{M}_{\text{Merge}}(\text{uri}_1 : \text{term}_1, ..., \text{uri}_n : \text{term}_n, \text{delimeter}) \): merges instances of the terms \( \text{uri}_1 : \text{term}_1, ..., \text{uri}_n : \text{term}_n \) into an instance of a single term with delimeter separating the instances.

  * \( \mathcal{M}_{\text{SPLIT}}(\text{uri}_1 : \text{term}, \text{delimeter}, \text{uri}_1 : \text{term}_1, ..., \text{uri}_n : \text{term}_n) \): splits instances of the single term \( \text{uri}_1 : \text{term} \) with separator \( \text{delimeter} \) into instances of the terms \( \text{uri}_1 : \text{term}_1, ..., \text{uri}_n : \text{term}_n \).

  For example, the mapping between EName, and firstName and lastName can be resolved by using the \( \mathcal{M}_{\text{Merge}} \) translational mapping:

  \[
  \text{uri}_s : \text{EName} \rightarrow \mathcal{M}_{\text{Merge}}(\text{uri}_1 : \text{firstName}, \text{uri}_1 : \text{lastName}, " ")
  \]

  where \( \mathcal{M}_{\text{Merge}}(\text{uri}_1 : \text{firstName}, \text{uri}_1 : \text{lastName}, " ") \) is an aggregate translational function that

```xml
1 <rdbo:Relation rdf:ID="Employee">
2   ...  
3 <owl:Restriction>
4   <owl:onProperty rdf:resource="#ID" />
5   <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">0</owl:minCardinality>
6 </owl:Restriction>
7 <rdbo:Relation>
8   ...  
9 <rdbo:Attribute rdf:ID="ID">
10   <owl:sameAs rdf:resource="http://localhost/dbOne/Dependent/Dependent.owl#EID" />
11   <rdbo:isNullable>false</rdbo:isNullable>
12   <rdbo:type>integer</rdbo:type>
13 </rdbo:Attribute>
```

Figure 8. OWL restriction for the one-to-many relationship between Employee and Dependent.
merges instances of firstName and lastName together.

6 RDB2ONT Prototype

A prototype of the RDB2ONT tool has been implemented using the Java language and we have tested our algorithm and our RDB2ONT tool with MySQL, PostgreSQL, and DB2 via JDBC/ODBC. Support for other relational database systems will be included in the future. One of the nice features of the RDB2ONT tool is that it supports both graphical and OWL views of multiple databases. Figure 9 shows a sample running session of the RDB2ONT tool (the inset dialog shows how generated ontologies are exported). More features will be added in the future.

7 Conclusions and Future Work

In this paper, an algorithm and tool for generating OWL ontologies from relational database systems is presented. The tool, RDB2ONT, helps the domain experts to quickly generate and publish OWL ontologies describing the underlying relational database systems while preserving their structural constraints. The generated ontologies are constructed using a set of vocabularies and structures defined in schema that describes relational database systems on the web so they guarantees that user applications can work with data instances that conformed to a set of known vocabularies and structures. The generated ontologies provide a standardized and meaningful way for describing the underlying relational database systems so they “bridge” the semantic gaps between the ontologies describing relational database systems and/or the ontologies describing other data sources on the web such as flat-files, semi-structures, etc. Concepts in OWL ontologies can be defined at multiple levels of granularities thus the generated OWL ontologies can be used to address the semantic heterogeneity problem at multiple levels. Evolutions of database systems in large-scale environments are inevitable so by using the RDB2ONT tool, OWL ontologies can be re-generated with little effort from the domain experts thus speed up the process of facilitating data in the underlying relational database systems with other data sources on the web.

Although the generated OWL ontologies provide the explicit meaning of concepts and their semantic relationships between related concepts, there are still many open research questions that need to be addressed. One of the questions is how to merge the generated OWL ontologies into an integrated OWL ontology so that common views of concepts can be achieved? This would allow users to pose queries on the common views of concepts rather than the concepts defined in the individual ontologies. Merging OWL ontologies will also require the resolution of the structural differences between OWL ontologies. Another question is how can existing OWL ontology inference engines such as Jena 2 [1] or Pellet [8] can be used to infer the semantic relationships between concepts defined in multiple OWL ontologies at multiple levels of granularities? The result of this research is important since it would enable users to combine related data from multiple data sources at multiple levels of granularities. In the real world, different organizations are likely to use different data models, thus tools for generating OWL ontologies automatically and dynamically from other data models (e.g., flat-files, object-oriented, etc.) are also needed. As mentioned previously, having different OWL ontologies describing different data models will allow users to relate semantically related data in different data models together thus enabling these different data sources to inter-operate with each other semantically.

8 Acknowledgements

The authors would like to thank Dave Heaton for implementing the RDB2ONT tool and its many nice features.

References

Figure 9. RDB2ONT sample session.


