Learning Semantics from Relational Database Schema and Data

Ching-Song Don Wei
Department of Computer
Information Systems
BMCC, City University of New York
New York City, NY, USA
cwei@bmcc.cuny.edu

Jiann-Gwo Doong
Department of Information Management
China University of Technology
Taipei, Taiwan
sdoong@cute.edu.tw

Peter A. Ng
Department of Computer Science
Indiana University-Purdue University Fort Wayne
Fort Wayne, IN, USA
ngp@ipfw.edu

Abstract—Ontology has been proposed to resolve the issues of semantic heterogeneity in data interoperability and data sharing among heterogeneous systems because it provides an effective approach to represent the knowledge about data of a specific domain and has been applied in many fields, such as Semantic Web, e-commerce and information retrieval, etc. However, building ontology for information source manually is not only hard and error-prone, but also very personal if there is no common guideline. We propose a framework for data interoperability using semantic Web service to resolve the semantic heterogeneity in healthcare environment. We found that learning ontology from existing information resources is a good solution to explicitly express the semantics of information source, but some semantics may be missing during the ontology learning process. Since relational database is widely used for storing source data, in this paper a new approach of learning OWL ontology from a relational database is proposed. In order to acquire the complete conceptual information, a group of learning rules are used to obtain OWL ontology, including classes, properties, property characteristics, cardinalities and instances. The proposed learning rules are being implemented as a prototype.

Keywords—Ontology; Ontology Learning; OWL; Relational Database; Relational Model

III. INTRODUCTION

Ontologies are formal and explicit specifications of a shared conceptualization which consists of objects, concepts, and other entities in some areas of interest and their relationships among them [1]. The concept of interest is described by classes, properties of each concept, and restrictions on properties. The most recent development for standard ontology language is Ontology Web Language (OWL) [2]. With the resource description framework (RDF) [13], and RDF schema, it facilitates machine interpretability of web content, whereas XML supports syntactical heterogeneity.

Ontology has been proposed to resolve the issues of semantic heterogeneity in data interoperability and data sharing among heterogeneous systems because it provides an effective approach to represent the knowledge about data of a specific domain and has been applied in many fields, such as Semantic Web, e-commerce and information retrieval, etc. However, building ontology for information source manually is not only hard and error-prone, but also very personal if there is no common guideline. In addition, extracting accurate knowledge from databases is very crucial in information interoperability. Since most of the information systems use relational database to store data, there is a great need to develop an effective and efficient approach to automatically learn semantics from relational database and represent it in ontology format. In order to make certain the quality of learned semantics, both the database schema and data are considered in the learning rules. In this paper, the proposed learning rules can generate OWL ontology from relational database, including classes, properties, property characteristics, class hierarchy, cardinalities, and instances.

Section 2 reviews related works. Section 3 discusses the definition of relational database and detailed learning rules that can extract semantics from relational database schema and data. The ontology generated by the proposed method and some existing methods are compared in Section 4. Section 5 gives a conclusion and states the future research.

IV. RELATED WORKS

Many research works have been published in the area of generating ontology from database schema. A detailed ontology development guide in [12] describes six steps for creating an ontology for a specific concept. In brief, these six steps are: 1. Determine the domain and scope of the ontology; 2. Consider reusing existing ontologies; 3. Enumerate important terms in the ontology; 4. Define the classes and the class hierarchy; 5. Define the properties of classes; and 6. Define the facets of the slots (properties). In this research, for the application of data sharing and exchanging among clinical databases, both source and target databases will be converted into ontology for further processing. For the application of Semantic Web Service and healthcare data interoperability, ontology is used to describe the functionality of Semantic Web Services in addition to the conversion of databases.

The issue of extracting semantics from a database schema was addressed and the issue of transforming a relational data model to an object-oriented model was addressed in [3] and [4], respectively. But these two methods did not have sufficient information for constructing ontologies from databases. Even though the object-oriented model is closed to an ontological model, there are some differences between them, such as the constraints and hierarchy of properties captured by the ontological model [5].

A reverse engineering technique is proposed [6] to construct a domain specific ontology based on the schemas of databases.
and a set of interesting queries. The process determines primary keys, foreign keys, and inclusion dependencies by analyzing the database schemas. This analysis also helps determine the main entities which are essential within the domain of interest. However, the refinement process for the created ontology may also eliminate the important information from the ontology.

Based on the degree of semantic recovery, this technique of reverse engineering can be classified as three types.

A. Direct transformation

[7] and [8] proposed a direct generation process that transforms each table to a class and each attribute to a property. If a table has foreign keys which reference to other tables, then for each foreign key a new property is added to the class corresponding to the reference table whose value is an instance of the class representing the referenced table.

In the description of a relational OWL project [9], a presentation format for data and schema information is proposed based on OWL. The representation for schema includes tables, columns, data type, primary keys, foreign keys, and the relations among each other.

Since this direct transformation does not fully recover the entity and relationship of a database schema, the transformed ontology does not have sufficient information recovered for semantic interoperability application.

B. Semantic recovery

According to [10], the GUI forms of a clinic information system contain the full semantic information that can be used for interoperability. The direct transformation transforms only the database schemes to classes in OWL without recovering the entity and relationship information, i.e. the semantic information. In order to recover the semantics, the reverse engineering of normalization process should be applied. A tool called DB2OWL [11] recognizes the concepts (classes) from tables, the object properties from integrity constraints and data type properties from non-key columns. The algorithm only uses a key-based approach to generating the ontology components and the mapping documents. However, this approach does not check the records of tables to identify the hierarchy of classes and the cardinality information.

C. Complete Semantic recovery

According to [5], the learning process generates, as its result, complete semantic information including classes, properties and property characteristics, class hierarchy, cardinality, and instances. However, it does not describe the mapping description document. In this paper, this method will be discussed and adopted with a mapping method.

V. THE MAPPING BETWEEN ONTOLOGY AND RELATIONAL DATABASE

Let \( A = \{A_1, A_2, \ldots, A_n\} \) denote a set of attributes with their corresponding domain \( D = \{D_1, D_2, \ldots, D_n\} \). In the relational model [12], each relation \( R \) is representing a table \( T \), which consists of a number of columns. Each column is named by an attribute \( A_i \). Then, let \( S_i = \{A_i; D_1, A_2; D_2, \ldots, A_n; D_n\} \) be a relational schema. A relation \( R_i \), defined by a relational schema \( S_i \), is a set of mapping from the attribute names to their corresponding domains. This means, a relation \( R_i(A_1, A_2, \ldots, A_n) = \{<a_1, a_2, \ldots, a_n> | \text{for each } 1 \leq k \leq n, a_i \text{ is an element of } A_i; D_i\} \). In [5], a series of functions are defined as follows. The function \( \text{attr}(R_i) \) returns the attributes for a relation \( R_i \). The function \( \text{dom}(A_i) \) returns the allowable value of attribute \( A_i \). For the functions of keys, the function \( \text{pkey}(R_i) \) returns the primary key of the relation \( R_i \) and the function \( \text{fkey}(R_i) \) returns the foreign key of the relation \( R_i \). The inclusion dependence and equivalence are equivalent only the functional relations in learning the ontology from relational database when considering the values (tuple) relations. We shall give the definitions of inclusion dependence and equivalence.

- Definition 1 – Inclusion dependence. Given relations \( R_1 \) and \( R_2 \) in a database, let \( A_1 = \{A_1, A_2, \ldots, A_m\}, A_j \subseteq \text{attr}(R_i) \) and \( A_2 = \{B_1, B_2, \ldots, B_n\}, A_2 \subseteq \text{attr}(R_2) \).

Let \( t(A_1) \text{ and } t(A_2) \) represent the tuple values of \( A_1 \) and \( A_2 \) respectively. If, for each \( t(A_1) \) in \( R_1 \), there exists \( t(A_2) \) in \( R_2 \) such that \( t(A_1) = t(A_2) \), then \( A_1 \) and \( A_2 \) are of inclusion dependency, denoted as \( R_1(A_1) \subseteq R_2(A_2) \). It is a generalization of referential integrity.

- Definition 2 – Equivalence. Given relations \( R_1 \) and \( R_2 \) in a database, let \( A_1 = \{A_1, A_2, \ldots, A_m\}, A_1 \subseteq \text{attr}(R_1) \) and \( A_2 = \{B_1, B_2, \ldots, B_n\}, A_2 \subseteq \text{attr}(R_2) \). If \( R_1(A_1) \subseteq R_2(A_2) \) and \( R_2(A_2) \subseteq R_1(A_1) \), then \( A_1 \) and \( A_2 \) are of equivalence, denoted as \( R_1(A_1) = R_2(A_2) \).

In addition to the inclusion dependence and equivalence, the relational model also includes integrity constraints such as entity constraint and referential integrity.

A. Learning Rules

In this paper, we consider a relation contains either data or no data. The method, proposed by [5], considers the values in relations without taking many-to-many relationship into consideration. The DB2OWL method, proposed in [11], does not consider the values in relations. By generalizing these methods, the following learning rules are proposed for creating the classes (concept), properties, hierarchy, cardinality and instances from a relational database.

B. Rules for Creating Classes

During normalization processes, the original relations may be decomposed into sub-relations iteratively. Therefore a class may be derived from information of a relation or information spreading across multiple relations in a database. It is also possible that the database design splits a table into multiple tables just for different categories of data.

1) Process I. Integrate information scattered in several relations into one class which describes a concept or an entity in the real world.

a) Rule 1. Check the table data for table integration.

For the relations \( R_1, R_2, \ldots, R_i \) in a database, if \( A_1 = \text{pkey}(R_1), A_2 = \text{pkey}(R_2), \ldots, A_i = \text{pkey}(R_i) \), where \( \text{dom}(A_1) = \text{dom}(A_2) = \ldots = \text{dom}(A_i) \) and \( R_i(A_1) = R_2(A_2) = \ldots = R_i(A_i) \), then the information spread across \( R_1, R_2, \ldots, R_i \) should be integrated into an ontological class \( C_i \).
In reality, it is very often that we need to use more than one attribute to be the primary key of a relation. Then \( pkey(R) = A_i \), which contains a list of one or more attributes from the set \( A_i \) of attributes, \( dom(A_i) = dom(A_{i1}) \times dom(A_{i2}) \times \ldots \times dom(A_{ik}) \), for \( A_i = \{ A_{i1}, A_{i2}, \ldots, A_{ik} \} \). \( R(A_i) \) is one or more columns of primary key values. For this case, \( R(A_i) = R(A_{i1}, A_{i2}, \ldots, A_{ik}) \) is a relation of \( k \) columns. The abovementioned Rule 1 can be extended into multiple columns.

When multiple relations are used to describe the same entity, the information should be integrated into one ontological class.

Example 1:
As in the sample of database schema described in Table 1, the relation Patient and the relation Patient_Detail should be integrated into one ontological class Patient because Patient(PatientID) = Patient_Detail(PatientID). Therefore, in OWL ontology, the class Patient is as follows.

\[
<\text{owl:Class}\ rdf:ID="\text{Patient}"/>
\]

<table>
<thead>
<tr>
<th>Relation</th>
<th>Primary Key</th>
<th>Foreign Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient (PatientID string, FirstName string, Address string, Gender char, DOB datetime)</td>
<td>PatientID</td>
<td>N/A</td>
</tr>
<tr>
<td>Patient_Detail (PatientID string, InsuranceCompany string, InsurancePolicy string, email/Address string)</td>
<td>PatientID</td>
<td>PatientID references to PatientID in Patient relation</td>
</tr>
<tr>
<td>Physician (PhysicianID string, FirstName string, LastName string, Address string, Gender char, DepartmentID string)</td>
<td>PhysicianID</td>
<td>DepartmentID</td>
</tr>
<tr>
<td>Staff (StaffID string, FirstName string, LastName string, Address string, Gender char, JobCode string, DepartmentID string)</td>
<td>StaffID</td>
<td>JobCode, DepartmentID</td>
</tr>
<tr>
<td>IsPatientOf (PatientID string, PhysicianID string)</td>
<td>StaffID, PatientID</td>
<td>StaffID, PatientID</td>
</tr>
<tr>
<td>Department (DepartmentID string, DepartmentName string)</td>
<td>DepartmentID</td>
<td>N/A</td>
</tr>
<tr>
<td>Job (JobID string, JobTitle string)</td>
<td>JobCode</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Assume that PhysicianID is a subset of StaffID. Since Physician(PhysicianID) is a subset of Staff(StaffID), then the class Staff, which is a superclass of a subclass Physician, is created via OWL ontology. This will be addressed in the later section. As well, we will also address the relationship between classes, such as IsPatientOf later.

b) Rule 2. Check the table structure for table integration.

For relations \( R_i \) and \( R_j \) in database, assume that \( att(R) = pkey(R) = fkey(R) = \{ A_i, A_j \} \) where \( A_i \cap A_j = \emptyset \), and \( A_i \cup A_j = att(R) \). If \( A_i \) refers to \( R_i \), via \( pkey(R_i) \) and \( A_j \) refers to \( R_j \) via \( pkey(R_j) \), then the many-to-many relationship should be expressed by object properties. The creating process will be described in Rule 6 of Process III.

When a relation \( R \) is used only to relate other relations \( R_i \) and \( R_j \) in a many-to-many relationship, then relation \( R \) does not need to map to a class. All attributes of \( R, att(R) \), can be divided into two disjoint subsets of \( A_i \) and \( A_j \), each of them participating in a referential constraint with \( R_i \) and \( R_j \).

Example 2:
In the sample of relational database schema described in Table 1, the relation IsPatientOf consists of two attributes, PatientID and PhysicianID. These two attributes are primary keys and also are foreign keys of relation IsPatientOf. PatientID is referring to Patient relation and PhysicianID, a subset of StaffID, is referring to Physician relation. The purpose of IsPatientOf relation is solely to create the many-to-many relationship between the two relations, Patient and Physician, i.e. a patient may have more than one physician and a physician may have more than one patient.

![Diagram](https://via.placeholder.com/150)

2) Process II. Create ontological classes.

a) Rule 3. Created an ontological class \( C_i \) for the relation \( R_i \).

An ontological class \( C_i \) can be created for the relation \( R_i \) if \( R_i \) is not used to relate the other relations as in Rule 1 or if there does not exist a relation that can be integrated with \( R_i \) as in Rule 2. In other words, \( R_i \) does not satisfy Rule 1 and Rule 2 but meets one of the following conditions:

- **Condition 1.** \( \|pkey(R_i)\| = 1 \).
- **Condition 2.** \( \|pkey(R_i)\| > 1 \), and there exists an \( A_i \), where \( A_i \subseteq pkey(R_i) \) and \( A_i \subset fkey(R_i) \).

If a relation \( R_i \) is used to describe an entity rather than to describe the many-to-many relationship between relations or if it is used to describe partial information of an entity and should be integrated with the other relation, then \( R_i \) can be mapped into one ontological class.

Example 3:
Consider the relational database schema sample given in Table 1. Create four ontological classes, Physician, Staff, Department and Job for the relations Physician, Staff, Department and Job, respectively. Rule 1 and 2 cannot be applied and these relations have only one primary key. As a result, using OWL ontology, we create four corresponding classes which are as follows.

a) Rule 4. Check tables that implement the foreign key relationship.

For relations \( R_i \) and \( R_j \), let \( A_i \subseteq \text{attr}(R_i) \) and \( A_j \subseteq \text{attr}(R_j) \). Let \( A_i = fkey(R_i), A_j = pkey(R_j) \), and \( A_i \subset pkey(R_j) \). Let \( R(A_i) \subseteq R(R_j) \). Then an object property \( OP \) will be
created based on $A_i$: Let the ontological classes $C_i$ and $C_j$ be created for the relations $R_i$ and $R_j$ respectively. An OP is created for $C_i$, where the domain and range of OP are $C_i$ and $C_j$ respectively.

b) Rule 5. Check a table that implements the foreign key relationship with partial primary.

For relations $R_i$ and $R_j$, two ontological object properties, called the “has-part” and “is-part-of” object property are created if the relations $R_i$ and $R_j$ meet the following two conditions:

Condition 1. $pkey(R_i) = 1$;
Condition 2. $fkey(R_i) \subseteq pkey(R_j)$, where $fkey(R_i)$ refers to $R_j$ and $fkey(R_j) \subseteq attr(R_j)$.

Let $C_i$ and $C_j$ be the classes corresponding to $R_i$ and $R_j$ respectively. Then, $C_i$ and $C_j$ are the domain and range of “is-part-of” object property, respectively. And $C_i$ and $C_j$ are the domain and range of “has-part” object property, respectively. These two properties are of inverse properties.

Example 4:

In the relational database schema sample given in Table 1, consider the relationships “work_for” and “has_Employee” as the “is-part-of” and has-part object properties, respectively. The is-part-of object property “work_for” can be created. The domain of the is-part-of object property “work_for” is the relation Staff and the range of the is-part-of object property is the relation Department. Likewise, the has-part object property “has_Employee” can be created. The domain of the has-part object property is the relation Department and the range of the has-part object property is the relation Staff. The corresponding OWL description is as follows.

Example 5:

In the database schema sample given in Table 1, the relation Staff identifies the relationship between the two relations Department and Job. According to the semantic of Staff table, an object property “has_job” can be created, where its domain and range are Department and Job respectively. Therefore, the object property can be as follows.

D. Rules for Creating Class/Property Hierarchy

1) Process V. Identify the class hierarchy in ontological structure.

a) Rule 9. Check the inheritance relationship of two relations to determine the hierarchy relationship of their corresponding classes.

For relations $R_i$ and $R_j$, if $A_i = pkey(R_i), A_j = pkey(R_j)$ and $R(A_i) \subseteq R(A_j)$, then the class/property of the $R_i$ is a subclass/subproperty of the class/property of $R_j$. Since the classes and properties can be organized in a hierarchy, this rule determines the inheritance relationship between classes or properties. If two relations in a given database have their inheritance relationship, then their two corresponding ontological classes or properties can be organized in a hierarchy.

Example 7:

As in the sample of relational database schema given in Table 1, the ontological class Physician is a subclass of the class
Staff according to Rule 9. The following ontology description describes this hierarchical relationship.

<owl:Class rdf:ID="Physician">
  <rdfs:subClassOf rdf:resource="# Staff"/>
</owl:Class>

E. Rules for Learning Cardinality

1) Process VI. Identify the cardinality constraints.

The cardinality specifies the constraint among classes and is one of the OWL properties. This property can be learned from the constraint of attributes in relations [5].

   a) Rule 10. Create minCardinality and maxCardinality of a property P.
      
      For relation \( R_i \) and \( A \in \text{attr}(R_i) \), if \( A = p\text{key}(R_i) \) or \( A = f\text{key}(R_i) \), then the minCardinality and maxCardinality of the property P corresponding to \( A \) is 1.

   b) Rule 11. Create minCardinality of a property P when A is NOT NULL.
      
      For relation \( R_i \) and \( A \in \text{attr}(R_i) \), if \( A \) is set as NOT NULL, then minCardinality of the property P corresponding to \( A \) is 1.

   c) Rule 12. Create maxCardinality of a property P when A is UNIQUE.
      
      For relation \( R_i \) and \( A \in \text{attr}(R_i) \), if \( A \) is set as UNIQUE, then maxCardinality of the property P corresponding to \( A \) is 1.

F. Rules for Learning Instances

1) Process VI. The instances in ontological class \( C_i \) will be created from the tuples of its (i.e., \( C_i \)) corresponding relation \( R_i \).

   a) Rule 13. Create instances
      
      For an ontological class \( C_i \), which corresponds to a number of relations, says, \( R_1, R_2, \ldots, R_n \), in a given database, then every tuple \( t \in R_1 \times R_2 \times \ldots \times R_n \) can be an instance of \( C_i \).

Given a relational database, we could apply Rule 1 to Rule 13 to construct automatically its corresponding database by means of the OWL ontology. The conversion between database and ontology can be achieved with the intent that the converted system can support data sharing and interoperability as well as dealing with the semantic heterogeneity problem.

VI. Experiments

The sample of database schema and data as described in Table 1 is used to create an ontology using various algorithms and tools. Our evaluation focuses on the proper creation of classes, object properties for foreign keys, datatype properties for attributes, hierarchy of classes, and instance in the ontology. The tools or algorithms include Datamaster, [5], [9], Topbraid, and the method proposed in this paper. The more information is captured by the ontology, the more semantics of information is preserved during the conversion. For evaluating these algorithms, all rules are used for creating the ontology manually. For evaluating these tools, the sample database is inputted into the tools for generating the ontology. The proposed method is found to restore more semantics of data and schema. Currently the proposed method has been implemented in Visual Studio 2008 with Oracle server.

VII. Future Research

In this paper, a set of learning rules is presented for constructing an OWL ontology from a database schema and data automatically. During the construction, both the schema and data are taken into consideration. According to our experiment and analysis, the proposed method can capture the semantics of database schema and data information more efficiently.

This method is integrated into a healthcare information system that promotes information interoperability and data exchange using ontology. Since the construction of an ontology from a database plays an important role for information interoperability, there is a need to improve the existing methods. Our future research includes improvement of learning rules based upon the feedback from the use of healthcare information system and the evaluation of the representation language between database and OWL ontology.

VIII. References


Workshop on Database Interoperability (InterDB 2007), held in conjunction with VLDB 2007, Vienna, Austria, 2007.


TABLE II. EXPERIMENT RESULT

<table>
<thead>
<tr>
<th>Tools or Authors</th>
<th>Classes</th>
<th>Properties</th>
<th>Class Hierarchy</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataMaster</td>
<td>Each table maps to a class in ontology*</td>
<td>Each foreign key maps to an object property in the base class</td>
<td>Each attribute that can’t map to object property maps to a data type property</td>
<td>NO  YES</td>
</tr>
<tr>
<td>(Li 2005)</td>
<td>Patient and Patient_Detail are merged. **</td>
<td>Each foreign key maps to an object property in the base class</td>
<td>Ditto</td>
<td>YES  YES</td>
</tr>
<tr>
<td>(de Laborda 2005)</td>
<td>Patient and Patient_Detail are merged. ***</td>
<td>Each foreign key maps to an object property in the base class</td>
<td>Ditto</td>
<td>YES”</td>
</tr>
<tr>
<td>Topbraid</td>
<td>Each table maps to a class in ontology*</td>
<td>Each foreign key maps to an object property in base class</td>
<td>Ditto</td>
<td>NO  YES</td>
</tr>
<tr>
<td>Proposed method in this paper</td>
<td>Patient and Patient_Detail are merged. **</td>
<td>Each foreign key maps to an object property in the base class**</td>
<td>Ditto</td>
<td>YES  YES</td>
</tr>
</tbody>
</table>

Note:
* There are seven classes created in the ontology: Patient, Patient_Detail, Physician, Staff, IsPatientOf, Department, and Job.
** Only five classes are created; Patient_Detail and IsPatientOf do not map to a class.
*** Only six classes are created; Patient_Detail does not map to a class.
+ When a foreign key attribute is also part of a primary key, a has-part-of object property is created.
++ In addition note +, many-to-many relationships are considered.
! A subclass is created according to data inclusion relationship such as Patient and Patient_Detail.
!! A subclass is created according to key relationship such as Patient and Patient_Detail.