OWL Ontology Extraction from Relational Databases via Database Reverse Engineering

Lili Lin, Zhuoming Xu*
College of Computer and Information, Hohai University, Nanjing, Jiangsu 210098, China
Email: {linlili, zmxu}@hhu.edu.cn

Ying Ding
School of Library and Information Science, Indiana University, Bloomington, IN 47405, USA
Email: dingying@indiana.edu

Abstract—The main purpose of the Semantic Web is driving the evolution of the current Web by enabling users to find, share, and combine information more easily. OWL ontologies play a key role in this effort. It is widely believed that the majority of current Web data sources are powered by relational databases (RDB). Thus developing approaches and tools for extracting OWL ontologies from RDB is helpful in bridging the gap between the existing Web data sources and the Semantic Web. This paper proposes a formal approach to automatic extraction of OWL ontologies from RDB using database reverse engineering (DBRE) technologies. The DBRE-based approach first identifies different relational structures to capture the natural domain semantics hidden in the relational schemas and data. Then it performs an automatic RDB-to-OWL schema translation by following a set of predefined translation rules that are based on the conceptual correspondences between RDB schema and OWL DL ontology. Our prototype implementation and case studies show that the proposed approach is feasible and effective.

Index Terms—ontology development, schema translation, relational database, database reverse engineering, OWL ontology

I. INTRODUCTION

The success of the Semantic Web [1] depends heavily on quick and cheap construction of Web ontologies. The importance of ontologies to the Semantic Web has prompted the establishment of the OWL ontology language [2,3] and the development of various OWL-aware ontology tools. Unfortunately, manual ontology development using current OWL ontology editors, such as Protégé, remains a tedious and time-consuming task that can easily result in a knowledge acquisition bottleneck. On the other hand, the majority of current Web content is dynamic content powered by relational databases (RDBs) [4,5], in which abundant domain semantics has been (implicitly) encoded. Therefore, it is valuable to investigate approaches and tools for extracting such domain semantics from RDBs and then constructing OWL ontologies in an automatic or semi-automatic way.

Extraction of domain semantics encoded in an RDB usually relies on database reverse engineering (DBRE) techniques [6-8]. However, to the best of our knowledge, existing approaches or tools [9-16] tend to extract incomplete or unnatural domain semantics from relational databases, without considering the conceptual correspondences between the database forward engineering and reverse engineering. For instance, current solutions often interpret the use of a primary-foreign key (i.e., the attribute or attribute set that is not only the primary key but also a foreign key) in a relational schema as the implementation of is-a hierarchy (i.e., subtype/supertype relationship) in the conceptual model of the relational database. But, in fact, the use of a primary-foreign key can also be the result of the translation of a one-to-one or one-to-many binary relationship in database forward engineering process [6]. Thus, it is inappropriate to always generate an rdfs:subClassOf axiom in the target OWL ontology when detecting a primary-foreign key in the relational schema. Moreover, when constructing the target ontology, other important ontology class axioms (such as owl:disjointClasses and owl:equivlentClass) and cardinality constraints (owl:minCardinality, owl:maxCardinality, and owl:cardinality) are often unreasonably created or even ignored by the existing methods or tools. As a result, the target ontology is semantically incomplete or unnatural, which is unable to fully capture the domain knowledge of such Semantic Web applications that their data is powered by the underlying relational database.

In this paper, we propose an automatic, DBRE-based approach to ontology extraction from a given relational database, aiming at extracting richer and more natural semantics from the database and constructing the target ontology in the OWL DL language. We also examine the implementability and efficiency of our algorithms through development of a prototype tool, R2OWL, and case study experiments. The experimental study indicates that our approach is effective and an automatic extraction tool is realizable using Java language.
The remainder of the paper is organized as follows. Sect. II reviews related work. We explicate our DBRE-based ontology extraction approach in Sect. III. Sect. IV presents our prototype tool, R2OWL, followed by our experimental results. Finally, we conclude our work in the last section.

II. RELATED WORK

Several studies have investigated the approaches for extracting ontology from relational databases using DBRE techniques.

Stojanovic, et al. [9] captured relational constructs (i.e., relations, attributes, attribute types, primary keys, foreign keys) from a relational schema through reverse engineering and provided a set of simple rules for translating relational constructs into semantically equivalent ontology constructs (i.e., classes, properties, axioms). During the translation process, they also provided user assistance when some kind of ambiguities occurs and domain semantics cannot be inferred. In order to discover more hidden semantics embedded within a RDB than that of [9], Astrova [10] proposed an approach to reverse engineering of RDB to ontology by an analysis of keys, data, attributes correlations as well as their combination. The reverse engineering process includes classification of relations, mapping relations, mapping attributes, mapping relationships, and mapping constraints. Moreover, Astrova, et al. [11] proposed a reverse engineering approach to ontology extraction from HTML-forms. They first extracted domain semantics by analyzing the HTML forms to restructure the underlying relational database schema, and then constructed the target ontology from the database schema through a set of pre-defined translation rules. During the semantics extraction process, the identification of relations, attributes and other relational constructs must be accomplished by users (e.g., domain experts) who are knowledgeable about the application domain. Finally, the target ontology was automatically generated with the predefined translation rules. Built upon the reverse engineering techniques, Labyte, et al. [12] introduced an automatic procedure for building ontologies starting from the integrity constraints present in the relational sources. They adopted a variant of the DLR-Lite description logic [17] for representing the extracted ontology due to its ability to express the mostly used modeling constraints and its nice computational properties. Summing up these works, all the target ontologies are expressed in the RDFS language (or F-Logic [18], DLR-Lite [17]) which does not have enough expressive power to capture all explicit and implicit semantics encoded in the relational database.

In order to exploit the expressive power of OWL, Li, et al. [13] firstly explicated a set of rules with formal notations for translating RDB schema into OWL ontology through analysis of relational schema and data. The rules were organized for learning classes, properties, hierarchy and cardinality. But some rules in their work failed to cover certain types of entity or relationship, such as n-ary, many-to-many binary relationships with additional attributes. Benslimane, et al. [14] also proposed an approach for automated migration of data intensive web pages into ontology-based semantic web. Different from [11], they provided a way to identification of is-a hierarchy and adopted OWL for expressing the target ontology. However, their approach involves a great deal of human participation and tends to generate an unstable ontology because HTML forms are often modified due to the requirements change of practical applications [19]. Other similar works [15,16] all put their emphasis on the analysis of the correlation between primary keys and foreign keys for constructing ontologies. However, all of them made a hypothesis that is-a hierarchy can be always detected through the use of primary-foreign keys in the relational schema, which is not in accord with the real situation in database design [6]. One exception is Cerbah’s solution [20], which generated is-a hierarchy with further refinement from the relational instances.

The comparison of the existing DBRE-based ontology extraction approaches is shown as Table I. Even though

<table>
<thead>
<tr>
<th>Work</th>
<th>Knowledge source</th>
<th>Ontology language</th>
<th>Degree of automation</th>
<th>Translation of primary-foreign keys</th>
<th>Axioms &amp; constraints in the ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>[9]</td>
<td>RDB schema</td>
<td>RDFS</td>
<td>Semi-automatic</td>
<td>is-a hierarchy</td>
<td>No</td>
</tr>
<tr>
<td>[10]</td>
<td>RDB schema &amp; data</td>
<td>F-Logic</td>
<td>N/A</td>
<td>is-a hierarchy</td>
<td>No</td>
</tr>
<tr>
<td>[11]</td>
<td>HTML forms</td>
<td>F-Logic</td>
<td>Semi-automatic</td>
<td>is-a hierarchy</td>
<td>No</td>
</tr>
<tr>
<td>[12]</td>
<td>RDB schema</td>
<td>DLR-Lite</td>
<td>Automatic</td>
<td>is-a hierarchy</td>
<td>Yes</td>
</tr>
<tr>
<td>[13]</td>
<td>RDB schema &amp; data</td>
<td>OWL</td>
<td>Automatic</td>
<td>is-a hierarchy</td>
<td>Yes</td>
</tr>
<tr>
<td>[14]</td>
<td>HTML forms</td>
<td>OWL</td>
<td>Semi-automatic</td>
<td>is-a hierarchy</td>
<td>Yes</td>
</tr>
<tr>
<td>[15]</td>
<td>RDB schema &amp; data</td>
<td>OWL</td>
<td>Automatic</td>
<td>is-a hierarchy</td>
<td>No</td>
</tr>
<tr>
<td>[16]</td>
<td>RDB schema &amp; data</td>
<td>OWL</td>
<td>Automatic</td>
<td>is-a hierarchy</td>
<td>No</td>
</tr>
<tr>
<td>[20]</td>
<td>RDB schema &amp; data</td>
<td>OWL</td>
<td>Semi-automatic</td>
<td>N/A</td>
<td>No</td>
</tr>
</tbody>
</table>
those approaches try to capture as much domain semantics as possible from the combination of relational schema and data, drawbacks still exist as follows: (1) some important OWL class axioms, such as the Disjoint Classes and Equivalent Classes axioms, are ignored during the ontology construction process; (2) cardinality constraints, i.e., $\text{owl:minCardinality}$, $\text{owl:maxCardinality}$ and $\text{owl:cardinality}$, are not correctly or naturally created in the resulting OWL ontology; (3) the $\text{rdfs:subClassOf}$ axiom is always generated from any primary-foreign key, but actually a primary-foreign key could also express an one-to-one (or one-to-many) binary relationship but not necessary an is-a hierarchy. Despite of their weakness, the existing approaches set up a foundation for the proposed work and are of valuable reference to our in-depth research. Compared with them, our approach captures richer and more natural semantics from relational databases, uses the normative web ontology language (OWL DL) as the target ontology language, and results in an easy-to-use ontology extraction tool.

Another type of related work concerns the RDB-to-RDF mapping approaches or tools, which focus on data mapping from relational databases to RDF datasets. Such mappings provide the ability to view existing relational data in the RDF data model. These tools, such as D2RQ [21], Virtuoso RDF View [22], and Triplify [23], offer a virtual SPARQL endpoint over the mapped relational data, or generate RDF dumps, or offer a Linked Data publishing interface. A variety of tools [24] use different mapping mechanisms and languages, making it difficult to share and reuse the mappings. Therefore, the W3C has recently standardized the RDB-to-RDF mapping mechanism and language, namely Direct Mapping [25] and R2RML [26]. All existing mapping approaches/tools generate RDF data expressed as instances of target RDFS/OWL ontologies. The target ontology is usually a simple transformation, typically tables-to-classes and columns-to-properties, from the structure of the relational database. The mapping author can otherwise construct or choose an ontology that captures the domain semantics of the database. In this regard, an extracted ontology from the database using our approach can be used in so-called domain-semantics based customized mappings [24,26].

III. DBRE-BASED ONTOLOGY EXTRACTION METHOD

In practical database applications, modifications to the relational schemas are rarely reflected in the conceptual schemas (e.g., Entity-Relationship, ER). Thus, we consider the relational schemas and data, rather than the conceptual schemas, as the important source of knowledge (i.e., domain semantics) to be extracted when generating OWL ontologies. In order to acquire natural semantics from a given relational database, we must do an in-depth analysis of the conceptual correspondences between database forward engineering and reverse engineering. Furthermore, in order to perform a direct extraction of OWL ontology from a given relational database schema, we need to formalize the two knowledge representation models and then design efficient, automatic algorithms for the relational-to-ontological schema translation.

A. Main Ideas

Theoretically speaking, database reverse engineering, from relational schema to conceptual schema (e.g., ER), can be regarded as a reverse process of the forward engineering [6]. Thus, doing an in-depth analysis of the conceptual correspondences between the ER, relational, and ontological schemas in the forward engineering and reverse engineering processes will help us gain a better understanding on how to extract natural domain semantics as much as possible from the relational database. According to the logical database design methodologies [27,28], we can give a set of clearly defined conceptual correspondences between ER schema and relational schema, as shown in the Forward Engineering part of Fig. 1.

Our previous work [29] has already presented a solid theoretical proof for the semantics preservation of the translation algorithm from ER schema to OWL ontology. Because both ER schema and OWL ontology belong to a conceptual level and there exists an elegant semantic correspondence between the two conceptual schemas, OWL ontology extraction from a relational database can be regarded as a direct reverse engineering process from relational schema to conceptual schema, as if there exists the ER schema as an intermediate model in the engineering process. In order to extract natural semantics for OWL ontology construction, we need to distinguish different structures (e.g., table types) in the RDB schema. After doing so, OWL constructs including, in the abstract syntax [30], Class, DatatypeProperty, ObjectProperty, cardinality, maxCardinality, minCardinality, partial (i.e., subClassOf), DisjointClasses, EquivalentClasses, unionOf, allValuesFrom, and restriction, can easily be acquired through the reverse engineering process from relational schema to OWL ontology, as shown in the Reverse Engineering part of Fig. 1.

B. Preliminaries

Here we restrict our attention to those aspects that constitute the core of a relational database schema. The considered relational elements include relations (tables), attributes (columns), datatypes, primary keys and foreign keys of relational schemas. We assume without loss of generality that all relational schemas are in 3NF. In the following, we give a formal syntax of a relational database schema in Definition 1.

Definition 1. A relational database schema is a tuple $S = (N, \text{attr}, DT, pk, fk)$, where

- $N$ is a finite name set partitioned into: (1) a subset $ET$ of entity table names; each entity table contains rows of instance data describing entities in the real world, (2) a subset $RT$ of relationship table names; each relationship table contains rows of instance data describing the relationships between entities, and (3) a subset $DT$ of datatype names; each datatype is a predefined RDBMS datatype, specifying a value range of the relevant instance data.
Figure 1. Conceptual correspondences between database forward engineering and reverse engineering.
For each \( T \in ET \cap RT \), there is a finite nonempty set \( \text{attr}(T) = \{A_1 : d_1, \ldots, A_n : d_n\} \) with attribute names \( A_1, \ldots, A_n \) and their corresponding datatypes \( d_1, \ldots, d_n \in DT \).

For each \( T \in ET \cap RT \), there is exactly one primary key (PK) \( pk(T) \) whose values uniquely determine each row of the instance data in \( T \), where either \( pk(T) \in \text{attr}(T) \) (in this case \( pk(T) \) is a single-attribute key) or \( pk(T) \subseteq \text{attr}(T) \) (in this case \( pk(T) \) is a composite key with more than one attribute).

For each \( T \in ET \cap RT \), there are \( n \geq 1 \) foreign keys (FKs). If \( n \geq 1 \), then the \( n \) foreign key(s) are \( f_k(T), f_{k_2}(T), \ldots, f_{k_n}(T) \subseteq \text{attr}(T) \), where each value of the attribute(s) in \( f_k(T), i = 1, \ldots, n \) references the relevant value of the attribute(s) in the primary key \( pk(R) \) of another entity table \( R \in ET \), denoted as \( \text{Ref}(f_k(T), pk(R)) \).

The OWL language has several increasingly-expressive sublanguages for different purposes and applications [2,3]. We choose OWL DL to model the resulting OWL ontology because of this sublanguage’s appropriate expressiveness to capture the extracted semantics. OWL DL has two types of syntactic form: exchange syntax, i.e., the RDF/XML syntax that is used to publish and share ontology data over the Web, and the frame-like style abstract syntax that is abstracted from the exchange syntax for facilitating access to and evaluation of the ontologies (being this the reason for describing our approach using the abstract syntax in the present paper). Typically, an OWL DL ontology consists of a set of axioms built using OWL identifiers and constructs. In the following, Definition 2 [29,31], we give a concise definition of an OWL DL ontology that is suitable for capturing the knowledge extracted from a relational database. Note that the complete syntax format for identifiers in Definition 2 is an Internationalized Resource Identifier (IRI) consisting of a base IRI and a fragment identifier.

**Definition 2.** An OWL DL ontology is a tuple \( O = (ID_o, Axiom_o) \), where

- \( ID_o = \text{CID}_o \cup \text{DPID}_o \cup \text{OPID}_o \cup \text{DTID}_o \) is a finite OWL identifier set partitioned into the following subsets that are pairwise disjoint:
  - a subset \( \text{CID}_o \) of class identifiers including user-defined classes and two predefined classes owl:Thing and owl:Nothing,
  - a subset \( \text{DPID}_o \) of datatype property identifiers; datatype properties link individuals to data values,
  - a subset \( \text{OPID}_o \) of object property identifiers; object properties link individuals to individuals,
  - a subset \( \text{DTID}_o \) of data range identifiers; each data range identifier is a predefined XML Schema datatype reference such as xsd:integer.

- \( Axiom_o \) is a finite OWL axiom set partitioned into a subset of class axioms and a subset of property axioms; each axiom is formed by applying OWL constructs (e.g., Class and ObjectProperty) to the identifiers or descriptions that are the basic building blocks of a class axiom and describe the class either by a class identifier or by specifying the extension of an unnamed anonymous class via the OWL ‘Property Restriction’ construct restriction and ‘Boolean combination’ construct unionOf.

C. Ontology Extraction Algorithms

1. Table type identification

The extraction of domain semantics is a knowledge discovery process that can help generate an OWL DL ontology from a given RDB by analyzing not only the database schema but also the instance data. In order to detect different semantics from different relational structures, we should classify different types of entity tables (normal entity, weak entity, subtype entity, and supertype entity, etc.) and various relationship tables including binary relationship (many-to-many, one-to-many, and one-to-one) tables and \( n \)-ary relationship tables. According to the Forward Engineering part of Fig. 1, a particular table type can be detected by analyzing its primary key, foreign key(s), and sometimes the instance data. The detection strategies are as follows:

- **Normal entity table:** it has no foreign key and one primary key. For example, as shown in cases (a), (b), and (c) of Fig. 1, Person, Faculty, and Lab are all normal entity tables.

- **Weak entity table:** it has exactly one primary key and one foreign key, and the foreign key is a subset of the primary key. For example, as shown in case (b) of Fig. 1, Dependent is a weak entity table.

- **Many-to-many binary relationship table:** it has exactly two foreign keys and one primary key, and the primary key is the composite of the two foreign keys. For example, as shown in case (g) of Fig. 1, Participate is a many-to-many binary relationship table.

- **\( n \)-ary relationship table:** it has at last three foreign keys and one primary key, and the primary key is the composite of the three foreign keys. For example, as shown in case (h) of Fig. 1, Take is an \( n \)-ary relationship table.

- **Subtype/supertype entity table:** the subtype table has exactly one foreign key that is also the only primary key of the table; the foreign key-referenced table is a supertype entity table. For example, as shown in case (i) of Fig. 1, AdminStaff, Student and Faculty are all subtype entity tables and Person is the supertype entity table.

- **Entity table containing a one-to-one or one-to-many binary relationship:** it has exactly one primary key and one foreign key, and the primary key is disjoint with the foreign key; if many values of the primary key relate to one value of the foreign key, then the entity table contains a one-to-many binary relationship; otherwise it contains a one-to-one binary relationship. For example, as shown in Fig. 1,
**Course** in case (f) is the entity table containing a one-to-many relationship whereas **Department** in case (d) contains a one-to-one relationship.

- **One-to-many/one-to-one binary relationship table:** it has exactly one primary key and two foreign keys, and the first foreign key is also the primary key whereas the intersection of the second foreign key with the primary key is an empty set; if many values of the primary key relate to one value of the second foreign key, then the table is a one-to-many binary relationship table; otherwise it is a one-to-one binary relationship table. For example, as shown in Fig. 1, **MinorIn** in case (e) is a one-to-many binary relationship table, while **WorkFor** in case (c) is a one-to-one binary relationship table.

Following the above strategies, we give algorithm **TabTypeIdentification** that is used to identify different table types in a given relational database. This algorithm must be executed before the schema translation algorithm **SchemaTrans**. A list of auxiliary predicates is given in Table II. These auxiliary predicates will be used to describe the algorithm steps in this paper.

Now we analyze the time complexity of the algorithm **TabTypeIdentification** by measuring the amount of work done by the algorithm. Notice that the preprocessing (i.e. extraction of database schema elements) is not the operation of this algorithm. Moreover, we can also ignore the instance data operations via predicate **Data()** because they can be implemented using simple, efficient queries supported by any high-performance SQL DBMS. With these in mind, we can use the total number of database schema elements, including tables, attributes, and FK/PK references, to measure the input size of the algorithm. Thus we have $n = n_T + n_A + n_R$, where $n_T$, $n_A$, $n_R$ denote the number of tables, attributes, and FK/PK references, respectively. Since the algorithm analyzes the PK and FK in each table, we can easily conclude that the worst-case time complexity of the algorithm is $O(n)$.

(2) **Schema translation**

The algorithm **SchemaTrans** specifies a set of schema translation rules, following which a relational database schema (**Definition 1**) can be translated into an OWL DL ontology (**Definition 2**).

Notice that the rationale behind the relational-to-ontological schema translation in the **SchemaTrans** algorithm is the existence of natural semantic correspondences between the relational model and the ontological model, as depicted in the Reverse Engineering part of Fig. 1. In addition, due to the straightforward correspondences [30] between the abstract syntax and the exchange syntax of the OWL language, an OWL DL ontology in the exchange syntax can also be easily generated through syntax transformation of the translated OWL DL ontology, even though we only use the abstract syntax to explain the ideas of the proposed algorithm and describe the algorithm steps in this paper.

Now let us analyze the time complexity of the algorithm **SchemaTrans**. The same reason as with the algorithm **TabTypeIdentification**, we can ignore the instance data analysis operations in the algorithm **SchemaTrans**. In addition, since the operations in Step 1, i.e. the translation from relational symbols to OWL class and data range identifiers, and all of the operations for creating datatype/object property identifiers in Step 2 of algorithm **SchemaTrans** can be simultaneously made as sub-operations when creating the OWL class and property axioms in Step 2, we can ignore these sub-operations and consider only the creation of class and property axioms in Step 2 to be the basic operations of algorithm **SchemaTrans**. Therefore, the basic operations of this algorithm are counted as follows:

### Table II.
**A List of Auxiliary Predicates**

<table>
<thead>
<tr>
<th>Category</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity tables</td>
<td>normEntityTab(T)</td>
<td>$T$ is a normal entity table</td>
</tr>
<tr>
<td></td>
<td>weakEntityTab(T)</td>
<td>$T$ is a weak entity table</td>
</tr>
<tr>
<td></td>
<td>ooEntityTab(T)</td>
<td>$T$ is an entity table containing a one-to-one binary relationship</td>
</tr>
<tr>
<td></td>
<td>omEntityTab(T)</td>
<td>$T$ is an entity table containing a one-to-many binary relationship</td>
</tr>
<tr>
<td></td>
<td>subEntityTab(T)</td>
<td>$T$ is a subtype entity table</td>
</tr>
<tr>
<td></td>
<td>superEntityTab(T)</td>
<td>$T$ is a supertype entity table</td>
</tr>
<tr>
<td>Relationship tables</td>
<td>naryRelTab(T)</td>
<td>$T$ is an $n$-ary relationship table</td>
</tr>
<tr>
<td></td>
<td>mmRelTab(T)</td>
<td>$T$ is a many-to-many binary relationship table</td>
</tr>
<tr>
<td></td>
<td>omRelTab(T)</td>
<td>$T$ is a one-to-many binary relationship table</td>
</tr>
<tr>
<td></td>
<td>ooRelTab(T)</td>
<td>$T$ is a one-to-one binary relationship table</td>
</tr>
<tr>
<td>Supertype/subtype relationship</td>
<td>subOf(T, R)</td>
<td>A subClassOf relationship between a subtype entity table $T$ and a supertype entity table $R$</td>
</tr>
<tr>
<td>Keys and data</td>
<td>NonFkSet(T)</td>
<td>The non-foreign key attribute set of a given table $T$</td>
</tr>
<tr>
<td></td>
<td>Data(A)</td>
<td>The values of given attribute(s) $A$</td>
</tr>
</tbody>
</table>

© 2013 ACADEMY PUBLISHER
Algorithm TabTypeIdentification \( (D) \)

Input: A relational database \( D \) including its schema \( S = (N, \text{attr}, \text{DT}, \text{pk}, \text{fk}) \) and instance data.

Output: All table types identified.

\[ \textbf{Tables} \leftarrow \text{ET} \cup \text{RT} \; ; \; \text{assign} \; \text{false} \; \text{to all auxiliary predicates;} \]

while \( \text{Tables} \neq \emptyset \) do {

\[ \begin{align*}
\quad & \text{Get a table } T \in \text{ET} \cup \text{RT} \; ; \\
\quad & \text{Tables} \leftarrow \text{Tables} \setminus \{ T \} \; ; \\
\quad & \text{if } T \text{ has no foreign key and one primary key then } \{ \text{normEntityTab}(T) \leftarrow \text{true} \; ; \text{continue} \}; \\
\quad & \text{if } T \text{ has exactly one foreign key } \beta_1(T) \text{ and one primary key } pk(T) \text{ such that } \beta_1(T) \subseteq pk(T) \text{ then } \{ \text{weakEntityTab}(T) \leftarrow \text{true} \; ; \text{continue} \}; \\
\quad & \text{if } T \text{ has exactly two foreign keys } \beta_1(T), \beta_2(T) \text{ and one primary key } pk(T) \text{ such that } pk(T) = \beta_1(T) \cup \beta_2(T) \text{ then } \{ \text{mntRelTab}(T) \leftarrow \text{true} \; ; \text{continue} \}; \\
\quad & \text{if } T \text{ has } n \text{ foreign keys and one primary key } pk(T) \text{ such that } pk(T) = \beta_1(T) \cup \beta_2(T) \cup \ldots \cup \beta_n, n \geq 3 \text{ then } \{ \text{naryRelTab}(T) \leftarrow \text{true} \; ; \text{continue} \}; \\
\quad & \text{if } T \text{ has exactly one foreign key } \beta_1(T) \text{ and one primary key } pk(T) \text{ such that } \beta_1(T) \subseteq pk(T), \text{ Ref}(\beta_1(T), pk(R)), T \neq R \text{ then } \{ \text{subEntityTab}(T) \leftarrow \text{true} \; ; \text{superEntityTab}(R) \leftarrow \text{true} \; ; \text{subOf}(T, R) \leftarrow \text{true} \; ; \text{continue} \}; \\
\quad & \text{if } T \text{ has exactly one foreign key } \beta_1(T) \text{ and one primary key } pk(T) \text{ such that } \beta_1(T) \cap pk(T) = \emptyset, \text{ Ref}(\beta_1(T), pk(R)), T \neq R \text{ then } \{ \text{omEntityTab}(T) \leftarrow \text{true} \; ; \text{continue} \} \}
\]

else \{ \text{oosEntityTab}(T) \leftarrow \text{true} \; ; \text{continue} \};

\[ \begin{align*}
\quad & \text{if } T \text{ has exactly two foreign keys } \beta_1(T), \beta_2(T) \text{ and one primary key } pk(T) \text{ such that } \beta_1(T) = pk(T), \beta_1(T) \cap pk(T) = \emptyset, \text{ Ref}(\beta_1(T), pk(R)), \text{ Ref}(\beta_2(T), pk(S)), R \neq S \text{ then } \{ \text{omRelTab}(T) \leftarrow \text{true} \; ; \text{continue} \} \}
\]

else \{ \text{oosRelTab}(T) \leftarrow \text{true} \; ; \text{continue} \};
\}


\[ \begin{align*}
\quad & \text{the executing times of creating property axioms in Step 2.1.1 is at most } n_d \text{ and that of creating class axioms in Step 2.1.2 is exactly } n_r; \\
\quad & \text{the total executing times of creating class axioms and creating property restrictions in Step 2.2.2 is at most } 2n_g \text{ and that of creating property axioms in Step 2.2.3 is less than } n_r; \\
\quad & \text{the total executing times of creating class axioms and creating property restrictions in Step 2.3.2 is at most } 4n_g \text{ and that of creating class axioms in Step 2.3.3 is less than } 2n_g; \\
\quad & \text{the executing times of creating class axioms in Step 2.4.2 is at most } n_r, \text{ so do the creation of property axioms in Step 2.4.3;}
\]

\[ \begin{align*}
\quad & \text{the executing times of creating class axioms in Step 2.5.1 is less than } n_g \text{ and that of creating class axioms in Step 2.5.2 and 2.5.3 are all at most } n_g/2.
\]

Summing up the above basic operations, the amount of work done by the algorithm \textbf{SchemaTrans} is:

\[ W(n) = n_g + n_1 + 13n_g < 13(n_2 + n_3 + n_g) = 13n. \]

That is, the worst-case time complexity of the schema translation algorithm is also \( O(n) \).

IV. IMPLEMENTATION AND EXPERIMENTS

A. Prototype Tool

Based on the proposed algorithms, we have implemented a prototype tool, called R2OWL, using Java programming language on J2SE 1.6.0 platform. The tool can take relational schemas and data as input, map them into an OWL DL ontology using our proposed algorithms, and produce the resulting ontology in both the abstract syntax and the RDF/XML syntax. R2OWL has four modules: (1) \textit{extraction} module: it uses JDBC API, more specifically, the java.sql package, to parse the contents of an RDB’s data dictionary in order to extract the physical database schema and then stores the schema data as an-memory data structure (Java class); (2) \textit{identification} module: it uses Java class methods corresponding with algorithm \textbf{TabTypeIdentification} to specify the table types; (3) \textit{translation} module: it uses Java class methods consistent with algorithm \textbf{SchemaTrans} to implement the translation from relational schema to OWL DL ontology in the abstract syntax; (4) \textit{transformation} module: it performs the ontology transformation from the abstract syntax to the RDF/XML syntax. The resulting ontology can be displayed in the tool screen and simultaneously saved as text files.

B. Algorithmic Efficiency Test

We carried out ontology extraction experiments with our R2OWL tool on a PC with configurations as CPU Intel Core2 Duo P8400/2.26GHz, DDR2 2G.

Table III lists the schema sizes of the five relational databases created with Microsoft SQL Server and tested in our experiments. Fig. 2 is a line chart that shows the actual running time of our algorithmic routines in the R2OWL tool testing the five relational databases. The experimental results indicate that: (1) the total time complexity of our ontology extraction approach, including the table-type identification algorithm \textbf{TabTypeIdentification} and the schema translation
Algorithm SchemaTrans (S) 

Input: A relational database schema S = (N, attr, DT, pk, fk) and its instance data; all table types in S have been identified in advance via algorithm TabTypeIdentification.

Output: The resulting OWL DL Ontology O = φ(S) = (ID_x, Axiom_y) that is defined by a translation φ.

1 The translation from relational symbols to OWL class and data range identifiers:

1.1 for each entity/relationship table symbol T ∈ ET ∪ RT, create a class identifier φ(T) ∈ CTID_x;

1.2 for each datatype symbol d ∈ DT, create a data range identifier φ(d) ∈ DTID_y by mapping it to an XML Schema datatype reference;

2 The translation from relational elements to OWL class, datatype/object property identifiers, and class/property axioms:

2.1 for each table T ∈ ET ∪ RT and its non-foreign key attribute set NonFKSet(T) = {A_1, A_2, ..., A_k}, h ≥ 1:

2.1.1 create datatype property identifiers φ(A_1),...φ(A_h) ∈ DPID_x and property axioms:

DatatypeProperty(φ(A_1) domain(φ(T)) range(φ(d_1)));... DatatypeProperty(φ(A_h) domain(φ(T)) range(φ(d_h)));

2.1.2 create a class axiom for the table:

Class(φ(T) partial restriction(φ(A_1) allValuesFrom (φ(d_1)) cardinality(1))... restriction(φ(A_h) allValuesFrom(φ(d_h)) cardinality(1)));

2.2 for each foreign key attribute A ∈ fk(T), T ∈ ET, R ∈ ET, pk(T) = R then add maxCardinality(1) to φ(A)’s restriction;

2.2.1 create an object property identifier φ(A) ∈ OPID_x;

2.2.2 create a class axiom:

Class(φ(A) partial restriction(φ(R) allValuesFrom(φ(T))));

if ooRelTab(T) = true then add maxCardinality(1) to φ(A)’s restriction

else if omRelTab(T) = true and A = pk(T) then add maxCardinality(1) to φ(A)’s restriction;

2.2.3 create a property axiom:

ObjectProperty(φ(A) domain(φ(R)) range(φ(T)));

2.3 for each foreign key attribute A ∈ fk(T), T ∈ ET, subEntityTab(T) = false, i = 1, ..., n (n ≥ 1) such that Ref(fk_i(T), pk(R)), R ∈ ET, R ∈ T and A references the value of attribute B ∈ pk(R),

2.3.1 create a class identifier φ(T_i) ∈ CTID_x and two object property identifiers φ(A), φ(B) ∈ OPID_x;

2.3.2 create two class axioms:

Class(φ(T) partial restriction(φ(A) allValuesFrom(φ(T_i) φ(R))));

Class(φ(R) partial restriction(φ(B) allValuesFrom(φ(T_i) φ(R))));

if omEntityTab(T) = true or weakEntityTab(T) = true then add maxCardinality(1) to φ(A)’s restriction and minCardinality(1) to φ(B)’s restriction;

else if ooEntityTab(T) = true then add maxCardinality(1) to φ(A)’s restriction and minCardinality(1), maxCardinality(1) to φ(B)’s restriction;

2.3.3 create two property axioms:

ObjectProperty(φ(A) domain(φ(R)) range(φ(T_i) φ(R)));

ObjectProperty(φ(B) domain(φ(R)) range(φ(T_i) φ(R)));

2.4 for each foreign key attribute A ∈ fk(T), subEntityTab(T) = true such that fk_i(T) = pk(T) = {A_1, A_2, ..., A_k}, k ≥ 1:

2.4.1 create datatype property identifiers φ(A_1),...φ(A_k) ∈ DPID_x;

2.4.2 create a class axiom:

Class(φ(T) partial restriction(φ(A_1) allValuesFrom (φ(d_1)) cardinality(1))... restriction(φ(A_k) allValuesFrom(φ(d_k)) cardinality(1)));

2.4.3 create property axioms:

DatatypeProperty(φ(A_1) domain(φ(T)) range(φ(d_1)));... DatatypeProperty(φ(A_k) domain(φ(T)) range(φ(d_k)));

2.5 for each T_1, T_2, ..., T_m T ∈ ET, m ≥ 1 such that subEntityTab(T_j) = subEntityTab(T_m) = true, superEntityTab(T) = true and subOf(T_j, T) = subOf(T_m, T) = true,

2.5.1 for i = 1, m, create a class axiom:

Class(φ(T_i) partial φ(T))

2.5.2 if Data(pk(T_i)) ∩ Data(pk(T_j)) = ∅, i ≠ j, 1 ≤ i, j ≤ m then create a class axiom:

DisjointClasses(φ(T_1) φ(T_2) ... φ(T_m));

2.5.3 if Data(pk(T)) = ∪_{i=1}^{n} Data(pk(T_i)) then create a class axiom:

EquivalentClasses(φ(T) unionOf(φ(T_1) φ(T_2) ... φ(T_m)));

The translation from relational symbols to OWL class and data range identifiers is algorithm SchemaTrans, is O(n) according to the contrastive O(n) – line: t = 3n/2 + 100 (the dashed line) in the figure, which validates the aforementioned analytical results in Section III; (2) the running time is mainly spent on the RDB schema extraction (including visualization of the extracted schema elements using the tool), that is, the table-type identification, schema translation and ontology transformation in our approach is efficient.

C. Case Study

Saving space, here we just give a small-scale example to demonstrate the effectiveness of our proposed approach. Fig. 3 is the University database created with Microsoft SQL Server, simply modeling the real-world
semantics of a university. The corresponding ER schema of this relational database is shown in Fig. 4.

Fig. 5 and Fig. 6 are the screenshots of our R2OWL tool running the University database case. The left trees in Fig. 5 and Fig. 6 are all the visualization of the extracted database schema, showing all schema elements in the relational database, where ‘T’ denotes a table, ‘P’ a primary key, ‘F’ a foreign key, ‘R’ a primary-foreign key, ‘C’ a non-key attribute. The right text-areas in Fig. 5 and Fig. 6 display the resulting OWL DL ontology in the abstract syntax and the RDF/XML syntax, respectively. The resulting ontology is consistent with the theoretical output of the algorithm SchemaTrans. That means our approach is feasible and effective.

<table>
<thead>
<tr>
<th>RDB size (n)</th>
<th>Numbers of RDB schema elements</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>100</td>
<td>19</td>
<td>60</td>
<td>21</td>
</tr>
<tr>
<td>#2</td>
<td>200</td>
<td>30</td>
<td>131</td>
<td>39</td>
</tr>
<tr>
<td>#3</td>
<td>300</td>
<td>41</td>
<td>202</td>
<td>57</td>
</tr>
<tr>
<td>#4</td>
<td>400</td>
<td>52</td>
<td>273</td>
<td>75</td>
</tr>
<tr>
<td>#5</td>
<td>500</td>
<td>63</td>
<td>344</td>
<td>93</td>
</tr>
</tbody>
</table>

Figure 2. The running time of algorithmic routines in R2OWL.

Figure 3. Relational schemas of the University database.

Figure 4. ER Schema of the University database.
Figure 5. Screenshot of R2OWL, where the left displays the extracted RDB schema elements and the right shows the resulting OWL DL ontology in the abstract syntax.

Figure 6. Screenshot of R2OWL, where the left displays the extracted RDB schema elements and the right shows the resulting OWL DL ontology in the RDF/XML syntax.
V. CONCLUSIONS

This paper presents a formal approach to automatic extraction of OWL ontologies from RDBs using database reverse engineering technologies. The core contribution of this work is that the proposed approach is more formal and can extract richer and more natural domain semantics from RDBs compared with the existing solutions. There are similar solutions to learning OWL ontology from other kinds of data sources like XML documents [32]. However, given the fact that the majority of current Web data sources are powered by RDBs, our approach can be widely applied in ontology development of Semantic Web applications whose underlying data sources are modeled in the relational model and thus can act as a gap-bridge between existing Web data sources and the Semantic Web. For instance, using our approach, an ontology extracted from the underlying RDB of a deep web site can be employed to annotate the dynamic web pages generated by the web site. This kind of applications, so-called “deep annotation”, has been addressed in [33].

In the future, we plan to extend our approach to support ontology extraction from relational databases whose tables may have vertical partitioning, horizontal partitioning and denormalization, and to better support other syntax formats of the new OWL 2 language.

ACKNOWLEDGMENT

This work was supported in part by the following grants: (1) No. 20090094110015 from the Specialized Research Fund for the Doctoral Program of Higher Education of China, (2) No. BK2008354 and BK2010520 from the National Science Foundation of Jiangsu Province of China, (3) No. 2008135 from the “Six Talent Peaks Program” of Jiangsu Province of China.

REFERENCES

[20] F. Cerbahr, “Learning ontologies with deep class hierarchies by mining the content of relational databases,” in Advances in Knowledge Discovery and Management, Studies in Computational Intelligence, vol. 292, F. Guillet,


Lili Lin was born in China. She received the bachelor’s and master’s degrees in computer science from Hohai University, China, in 2007 and 2010, respectively. She is also a visiting scholar at Indiana University Bloomington, USA since August 2011 till August 2013. She is currently a Ph.D. candidate in computer science at Hohai University, China. She has been involved in several semantic web research grants. Her research interests include ontology engineering, semantic web, and machine learning.

Zhuoming Xu was born in China. He received the bachelor’s and master’s degrees in computer science from Hohai University, China, in 1986 and 1994, respectively, and the Ph.D. in computer science from Southeast University, China in 2005. He was a visiting professor at Indiana University Bloomington, USA, between May 2011 and May 2012.

He is currently a professor of computer science in the College of Computer and Information at Hohai University, Nanjing, China, where he was the Chair (2004–2006) of the Department of Computer Science and Engineering and the Deputy Director (2007–2012) of Science and Technology Office at Hohai University. He has been involved, as principal investigators, in various research grants in the field of data management founded by China. His research interests include databases, Web IR, ontology and semantic web. Prof. Xu was a member of the Council of China Computer Federation from 2008 to 2011, and has been the Vice Chair of the Technical Committee on Computer Application of Jiangsu Computer Federation of China since 2010. He received the Outstanding Paper Award from the IEEE 2003 International Conference on Machine Learning and Cybernetics.

Ying Ding was born in China. She received the bachelor’s degree in information science from Xi’an Electronic Science & Technology University, China in 1993, the master’s degree in information science from the Graduate School of Chinese Academy of Sciences, China in 1996, and the Ph.D. from the School of Computer Engineering at Nanyang Technological University, Singapore in 2001. She is currently an associate professor in the School of Library and Information Science at Indiana University Bloomington, USA. She worked as a researcher at the Free University of Amsterdam, Netherlands from Apr 2000 to Feb 2003 and as a senior researcher at the University of Innsbruck, Austria from Feb 2003 to Jul 2008. She has been involved in various NIH and EU funded research grants. She has published 150+ scientific papers in journals and conferences. Her research interests include semantic web, social network analysis, citation analysis, Web IR, and knowledge management.

Prof. Ding serves as PC member for 120+ international conferences. She is the editorial board member of four ISI indexed journals in information science and semantic web.