

Automatic Extraction of Ontologies Wrapping Relational Data Sources

Lina Lubyte and Sergio Tessaris

KRDB Research Centre for Knowledge and Data – Free University of Bozen-Bolzano

Abstract. Describing relational data sources (i.e. databases) by means of ontologies constitutes the foundation of most of the semantic based approaches to data access and integration. In spite of the importance of the task this is mostly carried out manually and, to the best of our knowledge, not much research has been devoted to its automatization. In this paper we introduce an automatic procedure for building ontologies starting from the integrity constraints present in the relational sources. Our work builds upon the wide literature on database schema reverse engineering; however, we adapt these techniques to the specific purpose of reusing the extracted schemata (or ontologies) in the context of semantic data access. In particular, we ensure that the underlying data sources can be queried through the ontologies and the extracted ontologies can be used for semantic integration using recently developed techniques in this area.

In order to represent the extracted ontology we adopt a variant of the *DLR-Lite* description logic because of its ability to express the mostly used modelling constraints, and its nice computational properties. The connection with the relational data sources is captured by means of sound views. Moreover, the adoption of this formal language enables us to prove that the extracted ontologies preserve the semantics of the integrity constraints in the relational sources. Therefore, there is no data loss, and the extracted ontology constitutes a faithful wrapper of the relational sources.

1 Introduction

Recent research on ontology languages tailored to data access demonstrates that ontologies (or conceptual models) can be very effective in overcoming several limitations of traditional database systems. In particular, the capability of handling incomplete information has been proved crucial in several important applications of databases, including federated databases [1], data warehousing [2], information integration through mediated schemas [3], and the Semantic Web [4] (for a survey see [5]).

In order to take advantage of semantics charged techniques to access or reuse legacy data, one of the pre-requisites is the definition of wrappers providing formal and machine readable descriptions of the semantics of the underlying data sources. These wrappers are often ontologies which make explicit the assumptions (or constraints) over the stored data. The definition of these wrappers, in

spite of the fact that this is a crucial and error prone process, is usually performed manually with little or no automatic support. Moreover, it requires at the same time a deep understanding of the adopted ontology language and good knowledge of the data source being wrapped.

In this paper we propose a technique that enables the automatic extraction of an ontology from a relational data source; in addition, our algorithm provides mappings which connect the terms from the ontology to the actual data. These mappings are defined as sound views over the logical schema of the relational data source; i.e., similar to the global-as-view (GAV) approach in the information integration literature [3]. We show that the extracted ontologies capture all the constraints of the underlying data sources, and the availability of mappings enables the use of these ontologies to query and integrate the wrapped data.

The adopted ontology language – a variant of the *DLR-Lite* family of languages (see [6]) – is expressive enough to capture commonly used features from Entity-Relationship (ER) [7] and UML class diagrams¹, and at the same time is compatible with the Semantic Web ontology language OWL².

Our ontology extraction technique relies on the availability of the logical schema of the relational data sources as well as constraints (e.g. foreign keys, uniqueness, etc.) providing the actual semantics of the data. In most of the cases this information can be automatically extracted from any DBMS; but we are aware of the fact that often these constraints are not stored in the actual DBMS but enforced by the programs accessing and updating the data or by ad hoc triggers and stored procedures. To account for these cases we provide the possibility to manually annotate the logical schema in order to specify the constraints. Our experience in several projects showed us that while it is relatively easy for data analysts to provide the constraints of a specific database application, the process of writing an ontology describing the same data can be daunting. Moreover, most of the standard database constraints can be discovered by analysing the actual data.

The contributions of this paper are the adaptation of a Description Logic [8] based ontology language compatible with OWL³ including the definition of mappings over relational data (Section 2); the definition of an algorithm to extract an ontology given a relational data source (Section 3); and the formal proof that the extracted ontology fully capture the meaning of the data source using the general concept of *information capacities* (Section 3.1).

2 Formal Framework

In this section we define a formal framework for describing relational sources and their wrapping ontologies. For the input relational source, we adopt a standard relational model with integrity constraints. In order to represent the extracted ontology, we use a variant of *DLR-Lite* [6] description logic detailed below.

¹ <http://www.uml.org>

² See <http://www.w3.org/TR/owl-ref/>.

³ Demonstrated by the availability of a Protégé plugin in Section 5.

2.1 Relational Model, Constraints and Queries

We assume the reader is familiar with the basic notions of relational databases [9]. A *relational schema* \mathcal{R} consists of an alphabet of *relation* symbols, each one with a fixed set of attributes (assumed to be pairwise distinct) with associated datatypes. The number of attributes denotes the *arity* of a relation. We assume that the *database domain* is a fixed denumerable set of elements Δ representing real world objects, and that every element in Δ is denoted uniquely by a constant symbol, called its *standard name* [10]. Moreover, we consider Δ to be partitioned into the datatypes D_i and to contain a special constant null, called the null value⁴. Then, a *database instance* (or simply a *database*) \mathcal{D} over a relational schema \mathcal{R} is an (interpretation) function that maps each relation R in \mathcal{R} into a set $R^{\mathcal{D}}$ of total functions from the set of attributes of R to Δ ⁵.

The ontology extraction procedure takes as input a relational source. We abstract from any specific database implementation by considering an abstract *relational source* \mathcal{DB} , which is a pair (Ψ, Σ) , where Ψ is a relational schema as defined above and Σ is a set of *integrity constraints*, i.e., assertions on the relations that express conditions that are intended to be satisfied by database instances. A database \mathcal{D} over Ψ is said to *satisfy* a set of integrity constraints Σ expressed over Ψ if every constraint in Σ is satisfied by \mathcal{D} . Given a relation r in Ψ and s attribute of r , let A denote the sequence of attributes of r and $r[s]$ the *projection* of r on attribute s [9]. The database integrity constraints that we consider in our framework are the following (for more details see [11]):

- *nulls-not-allowed constraints*, written $nonnull(r, A)$, satisfied in a database when null is not contained in any attribute in A of r ;
- *unique constraints*, written $unique(r, A)$, satisfied in a database when the sequence of attributes A is unique in a relation r . If in addition we have $nonnull(r, A)$, then these correspond to *key constraints*, denoted $key(r, A)$;
- *inclusion dependencies*, written $r_1[s_1] \subseteq r_2[s_2]$ ⁶, satisfied in a database when projections over s_1, s_2 of relations r_1 and r_2 , respectively, are included one in the other. If in addition we have $key(r_2, s_2)$, we call them *foreign key constraints*;
- *exclusion dependencies*, written $(r_1[s_1] \cap r_2[s_2]) = \emptyset$, satisfied in a database when the intersection of the projections over s_1, s_2 of relations r_1, r_2 is empty set;
- *covering constraints*, written $(r_1[s_1] \cup \dots \cup r_m[s_m]) \subseteq r_0[s_0]$, satisfied in a database when the projection of the relation r_0 over s_0 is included in the union of the projections of the respective relations in the set.

⁴ We consider a null value to be different from any other constant and from a null value in any other tuple. Assuming this semantics is not crucial though, different ones can be accommodated.

⁵ I.e., each total function represents a single tuple in $R^{\mathcal{D}}$. We assume set semantics.

⁶ For simplicity, we restrict inclusion, exclusion and covering constraints to projections over single attribute; see last paragraph of Section 2.2

| | | |
|--|--|--------------|
| $R[s] \sqsubseteq R'[s']$ | $\pi_s R^{\mathcal{D}} \subseteq \pi_{s'} R'^{\mathcal{D}}$ | Inclusion |
| $R[s] \text{ disj } R'[s']$ | $\pi_s R^{\mathcal{D}} \cap \pi_{s'} R'^{\mathcal{D}} = \emptyset$ | Disjointness |
| key($R[s_1, \dots, s_k]$) | for all $\phi_1, \phi_2 \in R^{\mathcal{D}}$ with $\phi_1 \neq \phi_2$, we have $\phi_1(s_i) \neq \phi_2(s_i)$ for some $s_i, 1 \leq i \leq k$ | Key |
| $R_1[s_1], \dots, R_k[s_k]$ cover $R[s]$ | $\pi_s R^{\mathcal{D}} \subseteq \bigcup_{i=1 \dots k} \pi_{s_i} R_i^{\mathcal{D}}$ | Covering |

Fig. 1. Syntax and semantics of *DLR-DB* axioms.

2.2 Ontology Language

In this section we present the ontology language we shall deal in the rest of the paper, and we give its semantics in terms of relational models. The ontology language adopted can be seen as an alternative to the use of standard modelling paradigms of ER or UML class diagrams and enables to represent their commonly used modelling constructs (see [12]). The advantage over these formalisms lies on the fact that our adopted ontology language, besides enabling the use of automatic reasoning to support the designer, also represents models that preserve the relational ones (see Section 3.1).

We call a *DLR-DB* system \mathcal{S} a tuple $\langle \mathcal{R}, \mathcal{K} \rangle$, where \mathcal{R} is a *relational schema* as described in Section 2.1 and \mathcal{K} is a set of assertions involving names in \mathcal{R} . The *DLR-DB* ontology language, used to express the constraints in \mathcal{K} , is based on the idea of modelling the domain by means of *axioms* involving the projection of the relation over the attribute. We call \mathcal{K} an *ontology*.

An *atomic formula* is a projection of a relation R over one of its attributes, denoted by $R[s]$. The attributes involved in the projections correspond to key attributes of the respective relations. This reflects the fact that in conceptual models non key attributes are not considered relevant to identify an element of an entity or a relationship (see Example 1). Two attributes are *compatible*, if their datatypes are equal. Then we say that two atomic formulae $R[s]$ and $R'[s']$ are *compatible* iff the two corresponding attributes s and s' are compatible.

Given the atomic formulae $R[s], R'[s'], R_i[s_i]$, an *axiom* is an assertion of the form specified in Figure 1⁷, where all the atomic formulae involved in the same axiom must be compatible. In the same figure, we give the semantics of a *DLR-DB* system $\langle \mathcal{R}, \mathcal{K} \rangle$, which is provided in terms of relational models for \mathcal{R} , where \mathcal{K} plays the role of constraining the set of “admissible” models. A database \mathcal{D} is said to be a *model* for \mathcal{K} if it satisfies all its axioms. The above conditions are well defined because we assumed the compatibility of the atomic formulae involved in the axioms.

Example 1. To provide the intuition on the use of the *DLR-DB* formalism we show a simple example exhibiting some of the modelling constructs defined

⁷ In relational algebra, $\pi_s R^{\mathcal{D}}$ denotes the projection of $R^{\mathcal{D}}$ over attribute s [9].

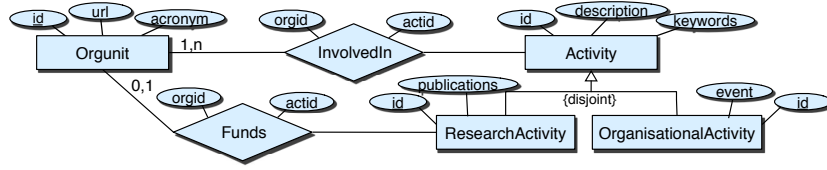


Fig. 2. ER diagram for Example 1

- | | |
|--|--|
| (1) $\text{InvolvedIn}[\text{orgid}] \sqsubseteq \text{Orgunit}[\text{id}]$ | (7) $\text{key}(\text{Activity}[\text{id}])$ |
| (2) $\text{InvolvedIn}[\text{actid}] \sqsubseteq \text{Activity}[\text{id}]$ | (8) $\text{key}(\text{ResearchActivity}[\text{id}])$ |
| (3) $\text{Funds}[\text{orgid}] \sqsubseteq \text{Orgunit}[\text{id}]$ | (9) $\text{key}(\text{OrganisationalActivity}[\text{id}])$ |
| (4) $\text{Funds}[\text{actid}] \sqsubseteq \text{ResearchActivity}[\text{id}]$ | (10) $\text{key}(\text{Funds}[\text{orgid}])$ |
| (5) $\text{Orgunit}[\text{id}] \sqsubseteq \text{InvolvedIn}[\text{orgid}]$ | (11) $\text{ResearchActivity}[\text{id}] \sqsubseteq \text{Activity}[\text{id}]$ |
| (6) $\text{key}(\text{Orgunit}[\text{id}])$ | (12) $\text{OrganisationalActivity}[\text{id}] \sqsubseteq \text{Activity}[\text{id}]$ |
| (13) $\text{ResearchActivity}[\text{id}] \text{ disj } \text{OrganisationalActivity}[\text{id}]$ | |

Fig. 3. *DLR-DB* axioms corresponding to the ER diagram in Figure 2

above. Consider the ER diagram shown in Figure 2, and assume, for the sake of exposition, that we have the underlying relational source containing a relation for each entity and relationship in the diagram. That is, we have relations *Orgunit* of arity 3, *InvolvedIn* of arity 2, etc. Then, to model the constraints reflected in the given ER diagram, we will define the axioms shown in Figure 3 that constrain the relational schema. In particular, axioms (1)–(4) represent role typing constraints, stating that the projection of *InvolvedIn* and *Funds* relations on the *orgid* (resp. *actid*) attribute is of type *Orgunit* (resp. *Activity* and *ResearchActivity*). Axiom (5) instead states mandatory participation, meaning that instances of *Orgunit* projected over the *id* attribute participate to the relation *InvolvedIn* as value for its projection over the *orgid* attribute. The key axioms in (6)–(10) express that, for instance, an object can appear in the *orgid* attribute of *Funds* relation only once. Axioms (11) and (12) correspond to is-a relationships among the respective relations, while axiom (13) states disjointness among the objects of the corresponding projections of relations.

The adopted ontology language is close to the *DLR* family of DLs [13]. This means that the same reasoning mechanism used for *DLR* can be employed for *DLR-DB*. The ability of employing correct and complete automated reasoning enables us to provide well-founded tools to support the maintenance and evolution of the extracted ontology (see [14]). More importantly, by taking away the covering axioms and allowing only unary key constraints, this language corresponds to *DLR-Lite_F* [6]. This implies that we can use the same efficient query answering technique (LOGSPACE in the size of the data) to evaluate conjunctive queries mediated by the ontology.

For the sake of simplicity, in this paper we restricted the atomic formulae to projections over a single attribute; however, our original definition of a *DLR-DB* system captures the notion of composite keys in relational databases. Assume

for example that in our ontology we need to represent the inclusion axiom corresponding to the foreign key constraint spanning over several attributes. To account for these cases, we associate to each relation in the relational schema a set of *components* (each one with a sequence of attributes) which partitions the set of attributes of the relation. Then, the axioms involve the projections of relations over their components instead of single attributes (see [11] for details). Importantly, we can still show that the reasoning mechanism used for \mathcal{DLR} and $\mathit{DLR-Lite}_{\mathcal{F}}$ can be employed for $\mathit{DLR-DB}$ extended with components.

3 Ontology Extraction

The principles of our ontology extraction process are based on ideas used in database reverse engineering (DBRE) literature.⁸ Roughly speaking, the essence of our extraction technique is to *reverse* the standard database modelling process [15], namely, that of translating ER model to the relational one. The benefit of such approach is that it can be shown that our algorithm, though heuristic in general, is able to reconstruct the original ER diagram. In this way, we can formally prove that our extraction procedure preserves semantics of constraints in the relational database (see Section 3.1).

The ontology extraction algorithm consists of two steps: (i) a classification scheme for relations is derived by analysing the constraints in the relational source, (ii) the actual ontology is generated, together with a set of sound views (i.e., GAV mappings [3]) that connect the extracted ontology with the source schema. Specifically, given a relational source $\mathcal{DB} = (\Psi, \Sigma)$ as input, the algorithm generates the $\mathit{DLR-DB}$ system $\mathcal{S} = \langle \mathcal{R}, \mathcal{K} \rangle$ with an ontology \mathcal{K} and a set of views in \mathcal{R} , defined over the source schema Ψ of \mathcal{DB} . So, \mathcal{R} can be seen as a new schema containing set of view definitions, and axioms of the extracted ontology \mathcal{K} are over names in \mathcal{R} . In such setting, every ontology term has an associated view over the data sources (see Section 4 for an example).

Given a relation r in the source schema Ψ , in the following we will denote by A a sequence of all attributes of r , K the set of key attributes of r such that $\mathit{key}(r, K)$, and \mathbf{FK} the set of all foreign keys of r such that $r[FK_i] \subseteq r'[K']$, for each foreign key $FK_i \in \mathbf{FK}$ that references key K' of relation r' , where $1 \leq i \leq n$ and n – number of foreign keys of r . Then, each relation r in Ψ is classified as one of the following:

- *base relation*, if K and \mathbf{FK} do not share attributes;
- *specific relation*, if K is among \mathbf{FK} (i.e., key is also foreign key) and if *one* of the following holds
 - (a) $|\mathbf{FK}| = 1$, i.e., r has single foreign key, or
 - (b) $r'[FK'_i] \subseteq r[K]$, i.e., r is referred to by the foreign key of other relation;
- *relationship relation*, if K is entirely composed of foreign keys and $|\mathbf{FK}| > 1$;
- *ambiguous relation*, if it does not satisfy any of the above conditions.

⁸ We discuss how our proposed framework relates to DBRE approaches in Section 6.

| Relation type | Views in \mathcal{R} | Corresponding axiom in \mathcal{K} |
|--|--|---|
| base relation r $key(r, K)$ | $R = \pi_{A-FK}(r)$ | $key(R[K])$ |
| specific relation r $key(r, K)$ $r[K] \subseteq r'[K']$ | $R = \pi_{A-FK}(r)$ | $key(R[K])$ $R[K] \subseteq R'[K']$ |
| $r_1[K_1] \cap r_2[K_2] = \emptyset$ $r'[K'] \subseteq r_1[K_1] \cup \dots \cup r_m[K_m]$ | | $R_1[K_1] \text{ disj } R_2[K_2]$ $R_1[K_1], \dots, R_m[K_m] \text{ cover } R'[K']$ |
| base or specific relation r $r[FK_i] \subseteq r'[K'], FK_i \neq K$ $nonnull(r, FK_i)$ $unique(r, FK_i)$ $r'[K'] \subseteq r[FK_i]$ | $R'' = \pi_{K,K'}(r' \bowtie r)$ | $key(R''[K])$ $R''[K] \subseteq R[K]; R''[K'] \subseteq R'[K']$ $R[K] \subseteq R''[K]$ $key(R''[K'])$ $R'[K'] \subseteq R''[K']$ |
| relationship relation r $r[FK_i] \subseteq r'[K']$ $r'[K'] \subseteq r[FK_i]$ | $R = \pi_A(r)$ | $R[FK_i] \subseteq R'[K']$ $R'[K'] \subseteq R[FK_i]$ |
| ambiguous relation r | repeat steps as for relationship relations | |

Table 1. Summary of the extraction procedure

The intuition for the above classification scheme comes directly from the process of translating the ER model to relational model. In particular, a base relation r results from mapping an entity to a relation, and its foreign key FK_i – from “embedding” a one-to-one or one-to-many (i.e., functional) relationship between entities corresponding to (base or specific) relations r and r' . A specific relation instead follows from translating a sub-entity. The condition (b) in discovering a specific relation is needed due to the fact that such relation (i.e., with key being a foreign key) may also represent a one-to-one relationship between two entities mapped to a single relation. A relationship relation results from mapping a many-to-many relationship between entities corresponding to base or specific relations. Finally, note that in order not to mislay any relation during the extraction process, we put all “non-standard” relations to the ambiguous relations category.

Once relations in the source \mathcal{DB} are classified, the actual algorithm derives the ontology and views (i.e. mappings) by means of $DLR-DB$ system. Table 1 shows the corresponding axioms and view definitions that are generated for each category of relations and integrity constraints imposed on them. Note that the table also reflects the order in which the distinct relations are processed, that is, we start by creating axioms and mappings for base relations, then specific ones, followed by the corresponding structures for the foreign keys of base and specific relations, etc. Furthermore, observe that for ambiguous relations our automated algorithm uses heuristics which “prefers” to recover elements corresponding (in ER terms) to many-to-many, possibly n -ary relationships. However, we also provide the possibility for a user to manually define their intended meaning (see Section 5 for discussion).

It is easy to see that the whole two-step process is linear in the number of relations in the source schema.

3.1 Correctness and completeness of the technique

Our proposed ontology extraction technique can be seen as a *schema transformation* as defined in [16]. An important consideration in such a process (i.e., transforming one data model into another) is the potential for loss of information. We evaluate the correctness of our schema extraction procedure using the relative *information capacities* of the source and target schemata. In this section we briefly outline the main principles of this analysis; for full details and the actual proofs the reader is referred to [11].

In the following we denote by \mathcal{S} and \mathcal{T} source and target schemata corresponding to the input relational source \mathcal{DB} and relational schema \mathcal{R} of the extracted *DLR-DB* system. Let $\mathcal{D}_{\mathcal{S}}$ and $\mathcal{D}_{\mathcal{T}}$ be consistent instances of schemata \mathcal{S} and \mathcal{T} , respectively. An *equivalence preserving mapping* between the instances of \mathcal{S} and \mathcal{T} is a bijection $\mu : \mathcal{D}_{\mathcal{S}} \rightarrow \mathcal{D}_{\mathcal{T}}$. Then \mathcal{S} and \mathcal{T} are said to be *equivalent* via μ , denoted $\mathcal{S} \equiv \mathcal{T}$. Given schemas \mathcal{S} and \mathcal{T} , a (*schema*) *transformation* is a total function $\mathcal{M} : \mathcal{S} \rightarrow \mathcal{T}$. \mathcal{M} is an *equivalence preserving transformation* if it induces an equivalence preserving mapping. To this end, we can show the following:

Theorem 1. *The ontology extraction procedure is an equivalence preserving schema transformation.*

The actual proof of the above theorem can be found in [11]. Roughly speaking, we devise a bijective transformation for the respective models and we show that the constraints of the original schema and extracted ontology are satisfied by the models generated by this transformation.

The fact that our extraction procedure is equivalence preserving not only shows that there is no information loss, so the extracted schema can be used to access the data, but that we can evaluate queries expressed using the extracted ontology by simply expanding the generated views. This is no longer true in the case that the ontology is going to be modified; in this case, more sophisticated query answering techniques must be adopted in order to guarantee completeness (e.g. query rewriting [6]).

4 Ontology Extraction by Example

Consider the relational schema with constraints detailed below (keys are underlined).

| | |
|---|---|
| Orgunit(<u>id</u> , url, acronym, actid) | Activity(<u>id</u> , description, keywords) |
| ResearchActivity(<u>id</u> , publications) | OrganisationalActivity(<u>id</u> , events) |
| InvolvedIn(orgid, actid) | |
| (1) InvolvedIn [orgid] \subseteq Orgunit [id] | (4) Orgunit [id] \subseteq InvolvedIn [orgid] |
| (2) InvolvedIn [actid] \subseteq Activity [id] | (5) <i>unique</i> (Orgunit, actid) |
| (3) Orgunit [actid] \subseteq ResearchActivity [id] | (6) ResearchActivity [id] \cap OrganisationalActivity [id] = \emptyset |

At the initial step of the extraction process, relations *Orgunit* and *Activity* are classified as base relations, *ResearchActivity* and *OrganisationalActivity* as specific relations, while relation *InvolvedIn* satisfies the condition required for relationship relations.

The ontology generated from this relational source is the one given in Example 1 of Section 2. The extracted schema with view definitions is given below (we denote with **serif** and *slanted* font, respectively, relation names in the extracted and source schema).

$$\begin{aligned} \mathbf{Orgunit} &= \pi_{id,url,acronym}(\mathit{Orgunit}) \\ \mathbf{Activity} &= \pi_{id,description,keywords}(\mathit{Activity}) \\ \mathbf{ResearchActivity} &= \pi_{id,publications}(\mathit{ResearchActivity}) \\ \mathbf{OrganisationalActivity} &= \pi_{id,events}(\mathit{OrganisationalActivity}) \\ \mathbf{Funds} &= \pi_{id,id}(\mathit{Orgunit} \bowtie \mathit{Activity}) \\ \mathbf{InvolvedIn} &= \pi_{orgid,actid}(\mathit{InvolvedIn}) \end{aligned}$$

Now, consider the extracted ontology provided in Example 1 and the set of views above denoting mappings between the ontology and the actual sources. Suppose we want to know the the pairs of organisational units and research activities that those organisational units fund. The corresponding conjunctive query we would formulate is

$$q(x, y) \leftarrow \mathbf{Orgunit}(x, w, z), \mathbf{Funds}(x, y), \mathbf{ResearchActivity}(y).$$

To answer this query, it is enough to substitute each atom in the body of q with its corresponding query in the view definition, and to evaluate it over the actual data sources (we recall the reader that a conjunctive query can be translated into an equivalent SQL select-project-join (SPJ) query using standard translation [9]).

5 Implementation and Case Study

In order to evaluate the applicability of our approach, it is crucial to test it with real-world schemas. To automate the ontology extraction process for wrapping the underlying data sources, we have implemented the ontology extraction algorithm in a prototype system. The implementation allows to display the extracted ontology with annotated views using the ICOM Ontology Design tool [17]. However, in order to fully leverage the available and well-established techniques for using such wrappers for data access, we have also implemented an automatic ontology extraction support plug-in on top of the OBDA plug-in⁹ for Protégé¹⁰. The OBDA plug-in provides facilities to design Ontology Based Data Access (OBDA) system components (i.e., data sources and mappings). It supports the definition of relational data sources and GAV like mappings to link the concepts in the *DL-Lite_A* ontology [18] to the data in the defined sources. It also provides support for conjunctive query answering (by using SPARQL syntax), a service

⁹ <http://obda.inf.unibz.it/protege-plugin>

¹⁰ <http://protege.stanford.edu>

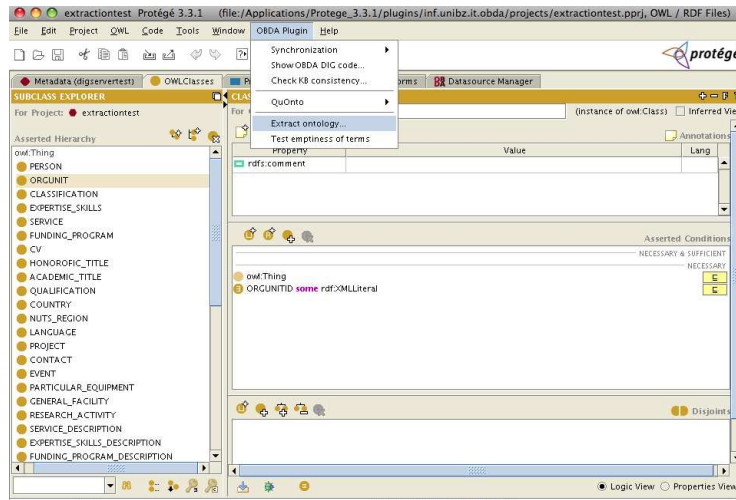


Fig. 4. Ontology extraction plug-in for Protégé

commonly offered by OBDA centric reasoners. A notable aspect of $DL\text{-}Lite_{\mathcal{A}}$ description logic used as ontology language in OBDA plug-in is that it admits query answering (with incomplete information) that is LOGSPACE in the size of the data at the sources. Even more importantly, it allows to reformulate query answering in terms of the evaluation of suitable SQL queries issued over the sources. Our ontology extraction framework fits thus very well into such OBDA setting (see [19] for interesting scenarios).

The goal of our plug-in is to provide a framework for the automated support for deriving the wrapping ontology from existing data sources together with an automatic generation of mappings. The ontology engineer can then explore the obtained ontology, possibly refine it, and formulate conjunctive queries over the this ontology using the OBDA plug-in. Figure 4 shows the screenshot of the ontology (displayed with OWL Protégé plug-in) automatically extracted from the source schema, where the data sources are specified by using the *Datasource Manager* tab of the OBDA plug-in. The generated mappings (i.e., views associated to ontology terms) are also manifested in the latter tab. The user at this point can pose queries over the resulting ontology, and the answers are returned from the underlying data source by taking into account the mappings (see OBDA plug-in website and [20] for details). It is worth noting however that since our ontology language supports n -ary roles, while $DL\text{-}Lite_{\mathcal{A}}$ (and OWL) does not, the extracted ontology must be *reified* [21].

We next report the results of a case study with CERIF database schema. CERIF (Common European Research Information Format)¹¹ is the standard EU recommendation used to harmonise databases on research projects. The schema is strongly structured containing 123 relations and is fairly rich in terms

¹¹ <http://cordis.europa.eu/cerif/src/toolkit.htm>

(a) Classified relations

| Relation type | #Classified |
|---------------|-------------|
| Base | 52 |
| Relationship | 25 |
| Specific | 0 |
| Ambiguous | 46 |

(b) Extracted axioms

| Axiom | #Extracted |
|--------------|------------|
| Inclusion | 68 |
| Key | 63 |
| Disjointness | 0 |
| Covering | 0 |

Table 2. Summary for extracting ontology from CERIF database schema

of integrity constraints explicitly declared through the DDL code. Table 2 shows the outcome of analysing the constraints over CERIF schema that result in classified relations (a) and the corresponding axioms derived for each class of relations (b). Note that those numbers do not include axioms for ambiguous relations.

From the table it can be seen that the tool produced correct¹² ontology constructs for 77 relations (out of 123). We were particularly interested in the usefulness of our approach for the category of ambiguous relations. We have identified that all 46 relations classified as ambiguous can be divided into two types: (i) those having their set of foreign keys properly included in their keys, and (ii) those having keys properly included in the set of their foreign keys, where the number of foreign keys is at least 3 and keys span at least 2 foreign keys. For instance,

- (i) PERSON_RESEARCH_INTEREST(LANGUAGE,TRANS_TYPE,PER_ID,KEYWORDS),
- (ii) PROJ_PERSON(PROJ_ID,PER_ID,PROJ_PER_ROLE,PROJ_PER_START,PROJ_PER_END)

where keys are underlined and PER_ID in (i) is the single foreign key, while in (ii) all PROJ_ID, PER_ID and PROJ_PER_ROLE are foreign keys. By carefully analysing the schema, we have derived that the intended meaning behind both types of the above relations are, in ER terms, relationships between entities corresponding to the referenced relations. In particular, LANGUAGE, TRANS_TYPE can be treated as a “hidden” foreign key referencing a “hidden” relation.

We are currently extending the original extraction algorithm in order to capture the above cases. The idea is to use an *iterative* approach for the extraction process. That is, first derive the initial classification as described in Section 3 with the corresponding axioms, then analyse ambiguous relations again by taking into account the above cases and manifest to the user possible suggestions.

6 Related Work

As we have mentioned, we build our method on top of the existing results in the area of *database reverse engineering* (DBRE) [22]. DBRE is defined as a process of recovering a conceptual model that represents the meaning of the logical

¹² With “correct” we mean the intended meaning when following the principled methodologies of relational database design from ER diagrams

schema by examining an existing database system to identify the database contents and their interrelationships. Approaches to recovering a conceptual schema from a relational database have appeared in the literature over the years [23–26]. Four main sources (and their combination) have been explored for finding evidence to construct a conceptual schema from a logical database: structures and integrity constraints of the database schema [23, 24], application programs that access the database [25], data instances stored in the database [26], and users and designers [27]. Moreover, because reverse engineering of relational databases is a complex task, all existing approaches are conditioned by a set of restrictive assumptions, namely, relational schemas are supposed to be normalised (3NF, BCNF), and the constraints on the schema are available (e.g., keys and foreign keys, inclusion and exclusion dependencies).

Even though there is a close connection between this area and the framework that we propose, there are however important differences. First, DBRE approaches usually produce just a pictorial representation of a conceptual model, without formal mappings that link the obtained schema to the database, and are thus used for “documenting” the database. Our approach, instead, is tailored for the direct use of the extracted ontology – that of accessing the data. In this setting, views generated during the extraction process that connect the derived ontology with the data sources play a crucial role. Second, most of DBRE methods are informal and do not specify quality of the outcome. On the contrary, we provide formal results showing that the extracted ontology represents all information sources and does not represent any extra information not present in the sources (see Section 3.1).

There are several works coming closer to ours that arose in the context of the Semantic Web and Information Integration, and that bring together relational databases and ontologies. Astrova [28] uses DBRE techniques to build ontologies with the purpose of *migrating* relational database content into the ontology (i.e., data-intensive web sites to the Semantic Web). Thus, while our framework uses the extracted ontology for accessing the data stored in external sources by means of sound views, here data is to be stored in the reverse engineered ontology. The work by Volz et al. [29]¹³ provides a framework for creating metadata by generating Web pages from an available database which finally leads to the *deep annotation* of the database. Then, such annotations can be used for two purposes: for querying the database through an ontology, and for migrating database content to ontology-based instance data. The database structure is first manually described on a Web page and then one of the means to create mappings between the ontology and the database is a semi-automatic process that integrates DBRE techniques. The advantage of our approach over this work, as well as the aforementioned one, is that our framework enables the representation of a *formal* ontology wrapping relational sources which allows for *automated reasoning* to support the designer. The proposals on data source ontology wrapping as SWARD [31] and Virtuoso¹⁴ systems support the auto-

¹³ Later version of this work has also appeared as part of WonderWeb project (see [30]).

¹⁴ <http://virtuoso.openlinksw.com/wiki/main/>

mated generation of RDF views over relational data sources, enabling to access the underlying data using RDF query languages. The advantage of our technique over all the above mentioned works is that we ensure *faithfulness* of the obtained model, meaning that it fully captures the meaning of the data source being wrapped.

D2R MAP [32] and R2O [33] provide means to declaratively state, respectively, ontology-to-database and database-to-ontology mappings. The mapping language of D2R MAP is based on RDF¹⁵, while R2O uses XML. Both languages allow to define expressive and explicit correspondences between components of the two models. A similar work in [34] presents an approach to map data stored in relational databases into the Semantic Web using RDF query language (e.g., SPARQL [35]). A different approach of [36] describes a (semi-)automatic mapping discovery between database relations and ontologies. In [18] a set of pre-existing sources is linked to the ontology (expressed in description logic) by defining expressive mappings. These works however require an existing target ontology the relations are mapped onto.

7 Conclusions and Current Work

We have described an heuristic procedure for extracting an ontology from a relational database schema. The mappings between the extracted ontology and the underlying database are defined by associating views over the original data to each term in the ontology. To represent the extracted ontology, instead of a graphical notation, we employ an ontology language thus retaining its precise semantics. Our extraction procedure integrates and enhances standard database reverse engineering techniques by relying on constraints defined over the database schema, (i.e., key and foreign key structure, restrictions on attributes and dependencies between relations), as well as standard methodologies for database design. We ensure that the underlying data sources can be queried through the extracted ontology by expanding the defined views.

We are aware of the fact that many databases have not been designed following the disciplined methodologies, so their schemas exhibit “idiosyncrasies”, as coined by Blaha et al [37]. To this purpose we are starting to experiment with other real database schemas to identify other design patterns and enrich with them our extraction algorithm. We also realise that often databases do not include all the relevant constraints on the data that were planned at design phase. Indeed, in most of the cases these constraints are enforced by the code accessing and updating the data. To the same extent, we are designing a tool to provide a support to explore the logical schema of a database and to facilitate the annotation of the schema by means of standard database constraints.

¹⁵ <http://www.w3.org/RDF/>

References

1. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous and autonomous databases. *ACM Computing Surveys* **22**(3) (1990) 183–236
2. Calvanese, D., Giacomo, G.D., Lenzerini, M., Nardi, D., Rosati, R.: Data integration in data warehousing. **10**(3) (2001) 237–271
3. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of PODS02. (2002) 233–346
4. Heflin, J., Hendler, J.: A portrait of the semantic web in action. *IEEE Intelligent Systems* **16**(2) (2001) 54–59
5. Wache, H., Vogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hubner, S.: Ontology-based integration of information - a survey of existing approaches. In: Proc. of IJCAI-01 Workshop: Ontologies and Information Sharing. (2001) 108–117
6. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. of Automated Reasoning* **39**(3) (2007) 385–429
7. Chen, P.: The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems (TODS)* **1**(1) (1976) 9–36
8. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F., eds.: *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, U.K. (2003)
9. Abiteboul, S., Hull, R., Vianu, V.: *Foundations of Databases*. Addison-Wesley (1995)
10. Levesque, H.J., Lakemeyer, G.: *The Logic of Knowledge Bases*. The MIT Press (2001)
11. Lubyte, L., Tessaris, S.: Extracting ontologies from relational databases. Technical report, KRDB group – Free University of Bozen-Bolzano (2007) Available at <http://www.inf.unibz.it/krdb/pub/TR/KRDB07-4.pdf>.
12. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on uml class diagrams. *Artificial Intelligence* **168**(1) (2005) 70–118
13. Calvanese, D., De Giacomo, G., Lenzerini, M.: Identification constraints and functional dependencies in description logics. In: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001). (2001) 155–160
14. Lembo, D., Lutz, C., Suntisrivaraporn, B.: Tasks for ontology design and maintenance. Deliverable D05, TONES EU-IST STREP FP6-7603 (2006)
15. Elmasri, R., Navathe, S.B.: *Fundamentals of Database Systems*. Fourth edn. Addison Wesley Publ. Co. (2004)
16. Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: The use of information capacity in schema integration and translation. In: Proc. of VLDB'93, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1993) 120–133
17. Fillottrani, P.R., Franconi, E., Tessaris, S.: The new icom ontology editor. In: Proc. of the 19th Int. Workshop on Description Logics (DL'06). (2006)
18. Poggi, A., Lembo, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Linking data to ontologies. *J. on Data Semantics* **X** (2008) 133–173
19. Rodriguez-Muro, M., Lubyte, L., Calvanese, D.: Realizing ontology based data access: A plug-in for protégé. In: Proc. of the Workshop on Information Integration Methods, Architectures, and Systems (IIMAS 2008). (2008) 286–289

20. Calvanese, D., Giacomo, G.D., Horridge, M., et al.: Software tools for ontology interoperation. Deliverable D25, TONES EU-IST STREP FP6-7603 (2008)
21. Noy, N., Rector, A.: Defining n-ary relations on the semantic web. Technical report, W3C Recommendation (2006) <http://www.w3.org/TR/swbp-n-aryRelations/>.
22. Hainaut, J.L.: Database reverse engineering: models, techniques and strategies. In: Proc. of the 10th Conference on ER Approach. (1998)
23. Markowitz, V.M., Makowsky, J.A.: Identifying extended entity-relationship object structures in relational schemas. *IEEE Transactions on Software Engineering* **16**(8) (1990) 777–790
24. Chiang, R.H.L., Barron, T.M., Storey, V.C.: Reverse engineering of relational databases: extraction of an eer model from a relational database. *Data and Knowledge Engineering* **12**(2) (1994) 107–142
25. Andersson, M.: Extracting an entity-relationship schema from a relational database through reverse engineering. In: Proc. of Int. Conf. on Entity-Relationship Approach (ER'94). (1994) 403–419
26. Alhajj, R.: Extracting an extended entity-relationship model from a legacy relational database. *Information Systems* **26**(6) (2003) 597–618
27. Johannesson, P.: A method for transforming relational schemas into conceptual schemas. In: Proc. of the Int. Conf. on Data Engineering (ICDE'94). (1994) 190–201
28. Astrova, I.: Reverse engineering of relational databases to ontologies. In: ESWS. (2004) 327–341
29. Volz, R., Handschuh, S., Staab, S., Stojanovic, L., Stojanovic, N.: Unveiling the hidden bride: deep annotation for mapping and migrating legacy data to the semantic web. *Web Semantics* **2**(1) (2004) 187–206
30. Volz, R., Handschuh, S., Staab, S., Studer, R.: Ontolift demonstrator. Deliverable Del 12, WonderWeb IST-2001-33052 (2004)
31. Petrini, J., Risch, T.: Processing queries over RDF views of wrapped relational databases. In: Proc. of the 1st Int. Workshop on Wrapper Techniques for Legacy Systems (WRAP 2004). (2004)
32. Bizer, C.: D2R MAP - a database to RDF mapping language. In: Int. World Wide Web Conference (WWW 2003). (2003)
33. Barrasa, J., Corcho, O., Gomez-Perez, A.: An extensible and semantically based database-to-ontology mapping language. In: Workshop on Semantic Web and Databases (SWDB 2004). (2004)
34. de Laborda, C.P., Conrad, S.: Database to semantic web mapping using RDF query languages. In: Proc. of the 25th Int. Conf. on Conceptual Modeling (ER 2006), Springer (2006)
35. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. Technical report, W3C Recommendation (2008) <http://www.w3.org/TR/rdf-sparql-query/>.
36. An, Y., Borgida, A., Mylopoulos, J.: Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. In: Int. Conf. on Ontologies, Databases and Applications of Semantics (ODBASE'05). (2005) 1152–1169
37. Blaha, M.R., Premerlani, W.J.: Observed idiosyncracies of relational database designs. In: Proc. of the Working Conf. on Reverse Engineering. (1995)