Supporting the Design of Ontologies for Data Access

A Preliminary Investigation

Lina Lubyte and Sergio Tessaris

Faculty of Computer Science – Free University of Bozen-Bolzano

Abstract. The benefits of using an ontology over relational data sources to mediate the access to these data are widely accepted and well understood. However, building such ontologies is a time consuming and error-prone process. Recently this problem has been tackled by proposing a technique for automatically wrapping the relational data sources by means of an ontology. In this paper we address the problem of supporting the ontology engineer in the task of enriching of such ontologies. In fact, changes to the ontology can be unsupported by the underlying data sources; which means that it is likely that queries against the enriched ontology return empty answers. To avoid this problem the ontology must be carefully handcrafted, and we propose a first step towards the automatic support of this task. We provide an algorithm, based on query reformulation, for verifying emptiness of a term in the enriched ontology with respect to the data sources. Moreover, we show how this algorithm can be applied to guide the user in designing meaningful wrappers.

1 Introduction

The use of a conceptual model or an ontology to access relational data sources has been shown to be necessary to overcome many important database problems. These include federated databases [1], data warehousing [2], information integration through mediated schemata [3], and the Semantic Web [4] (for a survey see [5]). Ontologies provide a conceptual view of the application domain, therefore they can be employed for navigational (and reasoning) purposes when accessing the data. This provides an additional motivation for supporting the extraction and maintenance of ontologies from database schema [6]. However, in order to fully leverage these technologies for accessing the data, it is necessary to preserve the mapping between data sources and ontologies.

In our previous work we defined a framework for the automatic extraction of ontologies from relational databases (see [7]). The semantic mapping between the database schema and the ontology is captured by associating a view over the source data to each element of the ontology. Thus, the vocabulary over the ontology can be seen as a set of (materialised) views over the vocabulary of the data source (i.e., similar to the technique known as GAV approach in the information integration literature [3]). The advantages of such a scenario are

clear since it enables to access and query the underlying data using a richer ontology vocabulary.

As described in our previous paper, queries over the extracted ontology can be simply evaluated by expanding the corresponding views (by virtue of the extraction algorithm each term has an associated view). Roughly speaking, the reason for this is that the extraction algorithm uses the constraints from the relational source; therefore we are guaranteed that the database is the minimal model for the generated ontology as well (see [8] for details). However, as soon as the extracted ontology is modified, the simple expansion of the views is no longer enough and the newly added constraints and terms must be taken into account. Using an appropriate ontology language, this can be done by means of query rewriting techniques (see [9]).

In most of the cases extracted ontologies are rather "flat", and constitute a bare bootstrap ontology rather than a rich vocabulary enabling enhanced data access. For this reason, the task of enriching the extracted ontology is crucial in order to build a truly effective ontology-based information access system. In this paper we concentrate on this latter task, showing that it is a complex and error prone process; in particular, when the purpose is to be able to access the data by means of the resulting ontology.

The process of modifying a given ontology involves at least the introduction of new axioms and/or new terms.[1] While, from a purely ontological viewpoint, an ontology can be arbitrarily modified; we need to bear in mind that the ultimate purpose of the system is to access or wrap the information available from the data sources. This means that we should be able to use the newly introduced terms in order to retrieve data from the sources.

It is easy to provide examples where perfectly sensible changes lead to terms which are completely useless; in the sense that queries over these terms will always be empty. This not necessarily because they are unsatisfiable in the usual model theoretic meaning, but because there is no underlying data supporting them (see Section 3.1 for examples). Usually, in order to ensure that queries over ontologies wrapping data sources provide sensible answers, these ontologies are carefully handcrafted by taking into account the semantics of the introduced axioms.

To the best of our knowledge, little or no research has been devoted to the support of the ontology engineer in such a complex and error prone task. Our research is directed to techniques and tools to support this modelling process. In this paper we present some preliminary results in this direction, presenting an algorithm for verifying the emptiness of terms in a given ontology w.r.t. a given set of terms representing the "information sources".

[1] We assume that axioms deriving from the data sources shouldn't be dropped, since they reflect the actual semantics of the data.

2 Formal Framework

Let us first introduce the formal framework for representing the ontology and for defining queries over this ontology that are used in the algorithm. The ontology language adopted enables the representation of standard conceptual modelling constructs which are commonly used in Entity-Relationship (ER) or UML class diagrams (see [10]).

2.1 The *DLR-DB* System

We call a *DLR-DB* system $\mathcal{S}$ a triple $\langle \mathcal{R}, \mathcal{P}, \mathcal{K} \rangle$, where $\mathcal{R}$ is a *relational schema*, $\mathcal{P}$ is a *component structure* over $\mathcal{R}$, and $\mathcal{K}$ is a set of assertions involving names in $\mathcal{R}$. In this section we describe these concepts.

We make use of the standard notion of relational model by using named attributes, each with an associated datatype, instead of tuples. Specifically, a *relational schema* $\mathcal{R}$ is a set of relations, each one with a fixed set of attributes (assumed to be pairwise distinct) with associated datatypes. $[s_1 : D_1, \ldots, s_n : D_n]$ denotes a relation having attributes $s_1, \ldots, s_n$ with associated data types $D_1, \ldots, D_n$. We interpret relations over a fixed countable *domain* Δ of datatype elements, which we consider partitioned into the datatypes D_i. 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. More detailed definitions can be found in [7].

We now introduce the concept of named components. The intuition behind a named component is the role name of a relationship in an ER diagram or UML class diagram. The *component structure* $\mathcal{P}$ associates to each relation a mapping from *named components* to sequences of attributes.

Definition 1. *Let R be a relation in $\mathcal{R}$, with attributes $[s_1 : D_1, \ldots, s_n : D_n]$. $\mathcal{P}_R$ is a non-empty (partial) function from a totally ordered set of named components to the set of nonempty sequences of attributes of R. The domain of $\mathcal{P}_R$, denoted $\mathcal{C}_R$, is called the set of* components of R. *For a named component $c \in \mathcal{C}_R$, the sequence $\mathcal{P}_R(c) = [s_{i_1}, \ldots, s_{i_m}]$, where each $i_j \in \{1, \ldots, n\}$, is called the* c-component *of R.*

We require that the sequences of attributes for two different named components are not overlapping, and that each attribute appears at most once in each sequence. I.e., given $\mathcal{P}_R(c_i) = [s_{i_1}, \ldots, s_{i_k}]$ and $\mathcal{P}_R(c_j) = [s_{j_1}, \ldots, s_{j_m}]$, if $s_{i_\ell} = s_{j_r}$ then $c_i = c_j$ and $\ell = r$.

The signature *of a component $\mathcal{P}_R(c)$, denoted $\tau(\mathcal{P}_R(c))$, is the sequence of types of the attributes of the component. Specifically, if the attributes of R are $[s_1 : D_1, \ldots, s_n : D_n]$, the signature of the component $\mathcal{P}_R(c) = [s_{i_1}, \ldots, s_{i_m}]$ is the sequence $[D_{i_1}, \ldots, D_{i_m}]$.*

Two components $\mathcal{P}_R(c_1)$ and $\mathcal{P}_R(c_2)$ are compatible *if the two signatures $\tau(\mathcal{P}_R(c_1))$ and $\tau(\mathcal{P}_R(c_2))$ are equal.*

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

$R[c] \sqsubseteq R'[c']$	$\pi_c R^{\mathcal{D}} \subseteq \pi_{c'} R'^{\mathcal{D}}$	Inclusion
$R[c]$ disj $R'[c']$	$\pi_c R^{\mathcal{D}} \cap \pi_{c'} R'^{\mathcal{D}} = \emptyset$	Disjointness
funct$(R[c])$	for all $\phi_1, \phi_2 \in R^{\mathcal{D}}$ with $\phi_1 \neq \phi_2$, we have $\phi_1(s) \neq \phi_2(s)$ for some s in c	Functionality

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

projection of the relation over the named component. An *atomic formula* is a projection of a relation R over one of its components. The projection of R over the c-component is denoted by $R[c]$. When the relation has a single component, then this can be omitted and the atomic formula R corresponds to its projection over the single component.

Two atomic formulae $R[c]$ and $R'[c']$ are *compatible* iff the two corresponding components $\mathcal{P}_R(c)$ and $\mathcal{P}_{R'}(c')$ are compatible. Given the atomic formulae $R[c], R'[c']$, an *axiom* is an assertion of the form specified in Figure 1, where all the atomic formulae involved in the same axiom must be compatible. In the same figure, there is the semantics of a *DLR-DB* system $\langle \mathcal{R}, \mathcal{P}, \mathcal{K} \rangle$, 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, and for each relation R in $\mathcal{R}$ with components $c_1, \ldots, c_k$, for any $\phi_1, \phi_2 \in R^{\mathcal{D}}$ with $\phi_1 \neq \phi_2$, there is some s in c_i s.t. $\phi_1(s) \neq \phi_2(s)$. The above conditions are well defined because we assumed the compatibility of the atomic formulae involved in the constraints. Note that, in the definition above, we require the satisfiability of all the axioms, and in addition we consider the sequence of attributes of all the components of a relation as a key for the relation itself. This reflects the fact that in conceptual models the additional attributes not belonging to any component are not considered relevant to identify an element of an entity or a relationship.

The use of an ontology language can be seen as an alternative to the use of standard modelling paradigms of ER or UML class diagrams. Within this perspective, relations with a single component can be seen as entities (or classes), while multiple components represent the roles of a relationship (see Section 3.1 for an example). The advantage of an ontology language over these formalisms lies on the fact that it has clear and unambiguous semantics which enable the use of automatic reasoning to support the designer. Note that by considering only components containing single attributes, this ontology language corresponds exactly to *DLR-Lite* (see [11]).

2.2 Conceptual Query Language

To describe the algorithm we make use of so called *conceptual queries* over a *DLR-DB* system $\mathcal{S} = \langle \mathcal{R}, \mathcal{P}, \mathcal{K} \rangle$. We call them "conceptual" because variables range over components (set of tuples) instead of single arguments of the relations

in $\mathcal{R}$. The reason for this is that the "ontological" structure is based on the components; so we don't need a finer query language enabling the retrieval of any attribute to analyse the emptiness of a term.

A *conceptual query*, is an expression of the form

$$q^c(\mathbf{x}) \leftarrow \exists \mathbf{y}.conj(\mathbf{x}, \mathbf{y}),$$

where

- $\mathbf{x}$ is a tuple of variables, so-called *distinguished variables*, each associated with a named component of a relation in $\mathcal{R}$,
- $\mathbf{y}$ is a tuple of existentially quantified variables called *non-distinguished variables*, each associated with a named component of a relation in $\mathcal{R}$,
- $conj(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms of the form $R(x_1, \ldots, x_k, y_1, \ldots, y_l)$, where R is a relation in $\mathcal{R}$ with $k + l$ named components.

A conceptual query q^c over a *DLR-DB* system $\mathcal{S}$ can be *expanded* to a standard conjunctive query q over the relational schema $\mathcal{R}$ by substituting each variable in q^c with a tuple of variables corresponding to the sequence of attributes underlying the component, and by introducing a fresh variable for every attribute not belonging to any component. Such an expansion is well defined, since we assumed components to be non-overlapping. Therefore, a tuple is in the answer to q^c iff it is in the certain answer to the corresponding expanded query q. To answer the expanded queries we can use the technique described in [12]. However, given the fact that components are non-overlapping, we can treat variables in conceptual queries as "singletons" instead of tuples. In this way we can employ the algorithm described in [11], as explained in Section 3.2.

3 Ontology Enrichment

In this section, for the sake of illustration of the matter, we start with a scenario for ontology enrichment. We then provide an algorithm for verifying emptiness of terms in a given ontology w.r.t. a set of database terms. Lastly, we show how this algorithm can be applied for supporting ontology enrichment task in ICom. Since our ontology language allows to specify the constructs used in conceptual modelling, in the rest of this paper we use interchangeably ER and *DLR-DB* terminology.

3.1 Scenario

Suppose an ontology is to be used for providing a semantically driven access to relational data sources in some data intensive application. Instead of wrapping relational data sources by means of such an ontology manually, it is desirable to acquire the core ontology by bootstrapping its design from the available database schema, at the same time inducing set of views over the actual data (i.e., GAV mappings), so linking the ontology to the database. This means that the derived ontology can be directly used to access the data sources, so that queries

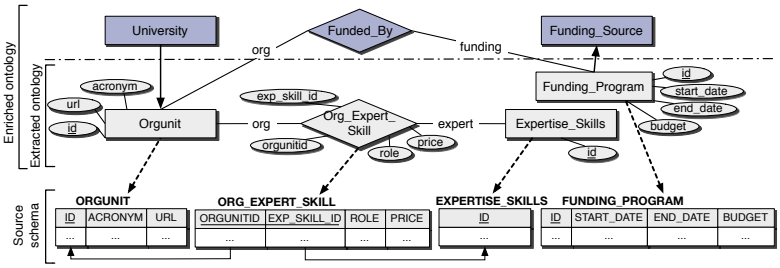

Fig. 2. Ontology enrichment scenario

formulated over this ontology are answered by simply extending the generated views. The algorithm for this task has been developed and presented in [7]. The ontology obtained with such a procedure – which can also be seen as data restructuring – is obviously shallow, reflecting the relational nature of data at the sources. It is therefore an important task to *enrich* it by adding more details to its parts that have not yet been sufficiently described. The query reformulation technique on which we base our emptiness check algorithm (see section 3.2) takes into account only inclusion axioms; we thus consider only those constructs for ontology enrichment that are modelled by means of such axioms. Specifically, one can add *(i)* specialisation or *(ii)* generalisation of an entity or relationship by introducing an axiom $R[c] \sqsubseteq R'[c']$, where R and R' are relations both having either one or more components, *(iii)* relationship between entities by introducing an axiom $R[c] \sqsubseteq R'$, *(iv)* mandatory participation constraint for an entity in a relationship by adding an axiom $R' \sqsubseteq R[c]$. An obvious but crucial aspect here is that enrichments result in the setting where added terms are no longer associated with those over the actual data. As a consequence, queries formulated over the enriched ontology are likely to return empty answers. It is thus beneficial to support this modelling process.

An appealing example of an ontology that mediate the access to data is the one automatically extracted from CERIF (Common European Research Information Format) database[2], EU's recommendation as a tool to harmonise databases on research projects. To illustrate the above arguments, consider a snippet of CERIF database schema in Figure 2 having relations ORGUNIT, ORG_EXPERT_SKILL, EXPERTISE_SKILLS and FUNDING_PROGRAM that contain information about organisational units, expertise skills that those organisations offer, and funding programmes. Employing the information available in this schema, i.e., keys, foreign keys, etc (see [7] for details), an initial ontology is automatically derived together with mappings: a view associated to every element of an extracted ontology (denoted with dashed arrows in Figure 2). Specifically, there are Orgunit, Expertise_Skills, Funding_Program relations with single components, each of them associated with id attribute, and Org_Expert_Skill relation with two components, org and expert, associated with attributes orgunitid and

[2] http://cordis.europa.eu/cerif/src/toolkit.htm

exp_skill_id, accordingly. The extracted ontology is thus the following set of inclusion axioms (functionality axioms are irrelevant for the rest of exposition and so are omitted):

Org_Expert_Skill[org] $\sqsubseteq$ Orgunit
Org_Expert_Skill[expert] $\sqsubseteq$ Expertise_Skills

Suppose now an ontology engineer decides that funding programmes provide funds to organisational units and hence adds a relationship Funded_By between the corresponding entities. Moreover, he/she wants to enrich the domain by adding University entity as a specialisation of organisational units and Funding_Source entity that generalises funding programmes. This triggers new axioms:

Funded_By[org] $\sqsubseteq$ Orgunit
Funded_By[funding] $\sqsubseteq$ Funding_Program
University $\sqsubseteq$ Orgunit
Funding_Program $\sqsubseteq$ Funding_Source

Given the resulting ontology, consider the queries below, having in their bodies only atoms that correspond to the newly added terms:

$q_1(x) \leftarrow \mathsf{University}(x)$,
$q_2(x) \leftarrow \mathsf{Funding\_Source}(x)$,
$q_3(x, y) \leftarrow \mathsf{Funded\_By}(x, y)$,
$q_4(x) \leftarrow \mathsf{Funded\_By}(x, y)$.

It is easy to see that q_1 returns an empty answer: given an object in Orgunit, there are models in which it is contained in University but there are models in which this is not the case. Therefore, this object is not in the certain answer to q_1. Following the same reasoning, Funding_Source includes all objects that appear in Funding_Program entity in all possible models, and thus q_2 is not empty. q_3 will be empty again: there is no way to induce the pairing of objects in Orgunit and Funding_Program. The last query will also return empty answer for the reasons discussed above. However, suppose an ontology engineer adds mandatory participation constraint for Orgunit in relationship Funded_By: Orgunit $\sqsubseteq$ Funded_By[org]. This means that now Funded_By necessarily contains all objects of Orgunit. Consequently, q_4 would now return all organisational units.

3.2 Checking Emptiness

The emptiness test takes a conceptual query with a single atom in its body and decides whether it yields an empty answer w.r.t. a given set of terms representing the information sources. In other words, it returns true if the corresponding expanded query has empty answer. The technique is based on the notion of applicability of an inclusion axiom to an atom of a query, which is the key of query reformulation algorithm in [11]. Intuitively, inclusion axioms are used as rewriting rules taking into account the knowledge in $\mathcal{K}$ that are relevant for determining whether a given term can be reformulated to the one coming from information sources.

As mentioned in Section 2.2, we can directly employ the applicability notion of *DLR-Lite* [11]. Note that an atomic formula $R[c]$ can be written as an atom

$R(\_, \ldots, \_, x, \_, \ldots, \_)$, where x is a bound variable (i.e., corresponds to a distinguished variable) associated to the c-component of R and $\_$ denotes an unbound (i.e., non-distinguished) variable.

Definition 2 (adapted from [11]). *An inclusion axiom I is* applicable *to an atom $R(\_, \ldots, \_, x, \_, \ldots, \_)$ if the right-hand side of I is $R[c]$.*

For $g = R(\_, \ldots, \_, x, \_, \ldots, \_)$, $gr(g, I)$ which indicates the atom obtained from an atom g by applying to it an inclusion axiom $I = R'[c'] \sqsubseteq R[c]$ is defined as $gr(g, I) = R'(z_1, \ldots, z_{i-1}, x, z_{i+1}, \ldots, z_l)$, where z_j, $1 \leq j \leq l$, are fresh variables and l is the arity of R'.

Let us denote by $\mathsf{Sig}(\mathcal{R})$ the set of relation names in $\mathcal{R}$, and by $\mathsf{Sig}(\mathcal{V})$ the subset of $\mathsf{Sig}(\mathcal{R})$ representing information sources, referred to as database terms[3]. We are now ready to define the algorithm $\mathsf{IsEmpty}$.

Algorithm $\mathsf{IsEmpty}(q, \mathcal{K}, \mathsf{Sig}(\mathcal{R}), \mathsf{Sig}(\mathcal{V}))$
Input: a query q, *DLR-DB* set of axioms $\mathcal{K}$, set of all relation names $\mathsf{Sig}(\mathcal{R})$ and a set of database terms $\mathsf{Sig}(\mathcal{V})$
Output: **true** if q yields an empty answer, **false** otherwise
$P := \{q\}$;
repeat
  $P' := P$;
  for each $q \in P'$ **do**
    for each I *in* $\mathcal{K}$ **do**
      if I *is applicable to* g *of* q
        then if *name of* $gr(g, I)$ *is in* $\mathsf{Sig}(\mathcal{V})$
          then return false
          else if *name of* $gr(g, I)$ *is not among atom names of all queries in* P
            then $P := P \cup q[g/gr(g, I)]$
until $P' = P$;
return true

$q[g/g']$ above denotes the query obtained from q by replacing the atom g with a new atom g'.

For a query q, the applicability of each inclusion axiom I in $\mathcal{K}$ to an atom g of q is checked. If an application of such I results in a reformulated atom whose name is among database terms, this term can be expanded to the view over the underlying data. If this is not the case, a new query with the reformulated atom is produced and added to P, verifying beforehand that the reformulated atom name does not already appear among reformulated atom names of queries in P[4].

The correctness of the algorithm follows from the fact that whenever it returns true, i.e. the given query is empty, all the reformulated queries in P have atoms that correspond to the newly added terms. Therefore, the answer to these

[3] I.e. relation names derived from the data sources and linked to them by means of views.

[4] This can happen when $\mathcal{K}$ has cycles.

queries will always be empty once they are evaluated over the actual data. Moreover, the algorithm always terminates, since the number of different atoms that can be generated is polynomial in the size of the input.

For a nice exposition of the algorithm, let us slightly change the enriched ontology in Figure 2: consider Funded_By relationship between University and Funding_Program, where University has mandatory participation. That is, we have, among those of extracted ontology, the axioms

(1) Funded_By[university] $\sqsubseteq$ University
(2) University $\sqsubseteq$ Funded_By[university]
(3) Funded_By[funding] $\sqsubseteq$ Funding_Program
(4) University $\sqsubseteq$ Orgunit,

and suppose we want to verify emptiness of term Funded_By projected on its university component, i.e $P = \{q(x) \leftarrow \mathsf{Funded\_By}(x, y)\}$. At the first execution of the main loop, inclusion axiom (2) is applicable to $\mathsf{Funded\_By}(x, y)$. The reformulated atom $\mathsf{University}(x)$ is not among database terms and is distinct from the head atom of $q \in P$, thus the new query is inserted in P, and now $P = \{q(x) \leftarrow \mathsf{Funded\_By}(x, y), q'(x) \leftarrow \mathsf{University}(x)\}$. At the second execution of the main loop, inclusion axiom (1) is applicable to an atom $\mathsf{University}(x)$. Since the name of the reformulated atom $\mathsf{Funded\_By}(x, y)$ is not among database terms but already appears among the names of atoms of the queries in P, the resulting P is equal to the one of the previous execution, and thus the algorithm returns true.

3.3 Ontology Enrichment Support in ICom

We show in this section how ontology enrichment can be supported with ICom, a tool which allows the user to design multiple UML class diagrams with inter- and intra-model constraints [13]. The tool employs complete logical reasoning to verify the specification of the model, display any inconsistency, devise stricter constraints and infer implicit facts. We extend this reasoning with the emptiness check for a newly added term in the ontology and manifest to the user whenever query over this term is empty.

Let us use the same example from Section 3.1; Figure 3 displays its modelling with ICom. The ontology representing the information sources and the newly added axioms are represented within separate UML class diagrams (referred to as models in ICom). In particular, specialisation and generalisation of entities are represented by simply adding an is-a relation between classes in the corresponding models; a new relationship between classes is accounted for by first "migrating" the entities involved in this relationship into the model for enrichments (done via equivalence relation), and then adding the association between the migrated classes. Note that since in our example FundingSource contains exactly the objects of FundingProgram, we can directly associate the former class to FundedBy. The emptiness check employed by the tool results in highlighting those terms for which there is no underlying data supporting them (see Section 3.1 for a detailed discussion).

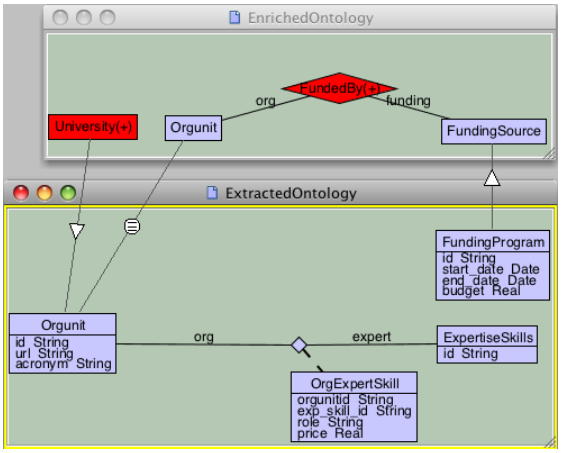

Fig. 3. Ontology enrichment support in ICom

4 Conclusions and Future Work

The problem of supporting ontology engineers in the task of enriching ontologies that act as wrappers over the data sources has been overlooked in Knowledge Representation and Ontology Engineering communities.

In this paper, we have started with a scenario based on our previous work, where we had an ontology bootstrapped from the available data sources, and concentrated on the task of guiding ontology engineer in enriching it, where the purpose was to be able to access the data sources by means of the resulting ontology. We have presented an algorithm for verifying the emptiness of terms in a given ontology w.r.t. a given set of terms representing the actual data. We have also shown how, employing this algorithm, ontology enrichment process can be supported by ICom ontology design tool.

This work is related to [14], where M. Marx shows that the packed fragment of FOL admits view-based rewritings. This means that given a FOL fragment and a database associated to some of its predicates, an arbitrary FOL query involving only implicitly defined predicates can be executed directly as an SQL query over the database extended with the precomputed materialised views that encode the explicit definitions. The restriction to the views is however in some cases too strong, as for instance Funding_Source in Figure 2 is not *definable* in terms of views but can be *reformulated* to a database term.

In the future, we aim at taking a broader view of the problem. So far we have been very restrictive on the ontology language. We will thus investigate how to support the introduction of new terms in ontologies that are defined in more expressive languages. E.g., one possibility could be to adopt language and rewriting technique of [15] so capturing, among others, is-a relationship between roles. Furthermore, the assumption on components to be non-overlapping should be reconsidered. In particular, it is imposed due to undecidability re-

sult for computing the certain answers to conjunctive queries when arbitrary inclusion dependencies are used (see [16] for details). However, since we are only interested in verifying emptiness of a given term, and not computing the certain answers to it, this assumption could be also omitted. We will also concentrate on methodological issues in supporting the modeller. In particular, keeping "reference" queries, that represent information needs, as a way of guiding the modeller in development of the ontology. Another matter is using an explanation to pinpoint what is missing in a definition of a newly introduced term. We expect that our results will constitute the basis for a well-founded methodology for ontology enrichment which can be supported by ontology design tools.

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., De Giacomo, G., 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. Lembo, D., Lutz, C., Suntisrivaraporn, B.: Tasks for ontology design and maintenance. Deliverable D05, TONES EU-IST STREP FP6-7603 (2006)
7. Lubyte, L., Tessaris, S.: Extracting ontologies from relational databases. In: Proc. of the 20th Int. Workshop on Description Logics (DL'07). (2007)
8. 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.
9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Dl-lite: Tractable description logics for ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005). (2005) 602–607
10. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on uml class diagrams. Artificial Intelligence **168**(1) (2005) 70–118
11. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of KR 2006. (2006) 260–270
12. Lembo, D., Lenzerini, M., Rosati, R.: Methods and techniques for query answering. Technical Report D5.1, Infomix Consortium (2004)
13. 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)
14. Marx, M.: Queries determined by views: Pack your views. In: PODS. (2007)
15. Cali, A.: Querying imcomplete data with logic porgrams: Er strikes back. In: ER. (2007) 245–260
16. Lembo, D., Lenzerini, M., Rosati, R.: Methods and techniques for query rewriting. Technical Report D5.2, Infomix Consortium (2004)