Baltic J. Modern Computing, Vol. 4 (2016), No. 3, pp. 466–482

OWLRel: Learning Rich Ontologies from Relational Databases

Lama AL KHUZAYEM and Peter McBRIEN

Department of Computing, Imperial College London, London SW7 2AZ, UK

{l.al-khuzayem11,p.mcbrien}@imperial.ac.uk

Abstract. Mapping between ontologies and relational databases is a necessity in realising the Semantic Web vision. Most of the work concerning this topic has either (1) extracted OWL schemas using a limited range of OWL modelling constructs from relational schemas, or (2) extracted relational schemas from OWL schemas, that represent the OWL schemas as much as possible. By contrast, we propose a general framework that maps between relational databases and schemas expressed in OWL 2. In particular, we regard the transformation from databases to ontologies as being a two-phase process. Firstly, to convert the relational schemas into OWL schemas, and then to enrich the OWL schemas with highly expressive axioms, based on analysing the schemas and the data in the databases. Testing our data analysis heuristics on a number of databases, show that they produce OWL schemas, that include more semantic information than found in their respective relational schemas.

Keywords: Reverse Engineering Databases, Ontology Learning, OWL 2 Ontologies, Database-to-Ontology Mapping, Data Analysis.

1 Introduction

The problem of bridging the gap between ontologies and relational databases is an active area of research (Spanos et al., 2012). Almost 70% of current websites store data in relational databases (Sequeda et al., 2012), and therefore being able to produce ontologies from such databases is considered an important technology to support the development of the Semantic Web. On the other hand, using relational databases for storing and processing ontologies with large datasets is considered a good solution, since the alternative of using Tableaux-based reasoners fails to scale to large datasets (Al Khuzayem et al., 2013).

The work in this paper focuses on establishing bidirectional mappings between relational schemas and ontologies expressed in OWL 2, the latest version of the **web ontology language** (**OWL**) (W3C, 2009). Most previous work (except (Atzeni et al., 2008; Dadjoo and Kheirkhah, 2015)) has performed the transformations at a high level (HL), which involves specifying mappings from constructs in the relational model to

ontological constructs, or *vice versa*. Moreover, most previous work has concentrated on either transforming relational databases to ontologies, or transforming in the opposite direction, without considering the provision of bidirectional mappings between the two models.

By way of contrast, our novel approach, as part of a framework presented in Section 2, generates a **bidirectional transformation** (**BT**) between databases and ontologies. This is performed by taking relational schemas and using them to generate BTs to equivalent OWL schemas. In previous work (Al Khuzayem et al., 2013), we have demonstrated the opposite direction; *i.e.*, generated BTs to relational schemas from OWL schemas. Our approach (which is implemented as a prototype called OWLRel[1]) is distinguished from other work in the area by the following:

1. Transformation between relational and OWL schemas is expressed using directional **both-as-view** (**BAV**) mappings (McBrien and Poulovassilis, 2003), allowing a precise definition of the equivalence between the schemas, that can map data back and forth between them. Therefore, starting from a given schema S_{owl} and generating BAV transformations to schema S_{hdm_2} (as illustrated in Figure 1), would result in generating BAV transformations in the other direction as well *i.e.*, from S_{hdm_2} $\rightarrow S_{owl}$. Hence, the double sided arrows in Figure 1.
2. By transforming via an intermediate low-level language, such as the **hypergraph data model** (**HDM**) (Poulovassilis and McBrien, 1998), we can reuse earlier work on translating between relational and ER, ORM, UML and XML data models (Boyd and McBrien, 2005; McBrien and Poulovassilis, 2001). The work of mapping between relational and OWL models presented here, allows OWL to be mapped to other data models (via the relational model) as illustrated in Figure 1.
3. Our approach includes two aspects which must be taken into consideration in this context (El Idrissi et al., 2013): human intervention and database content analysis. Our approach uses a combination of schema and data analysis to propose semantic information that is difficult to detect from the relational schema alone, and we rely on the user for the validation of these proposals.

Relying on data analysis to discover implicit constructs, cannot always be reliable, since adding or removing some data can easily disprove the discovered constructs. However, the larger the dataset, the more reliable the findings become.

Note that our work is referred to as **reverse engineering databases** in the database community, while in the Semantic Web community, it is termed as **ontology learning** (Cerbah, 2008) or **semantic annotation** (Astrova et al., 2007).

The remainder of this paper is structured as follows. Section 2 presents our framework for mapping between the relational model and OWL, and Section 3 reviews the HDM. Section 4 describes our automatic ontology extraction approach and in Section 5, we demonstrate our ontology enrichment process. We evaluate the prototype tool OWLRel in Section 6 followed by related work in Section 7. Finally, Section 8 concludes this paper.

[1] automed.doc.ic.ac.uk/releases/jars/OWLRel-rel-0-1.jar

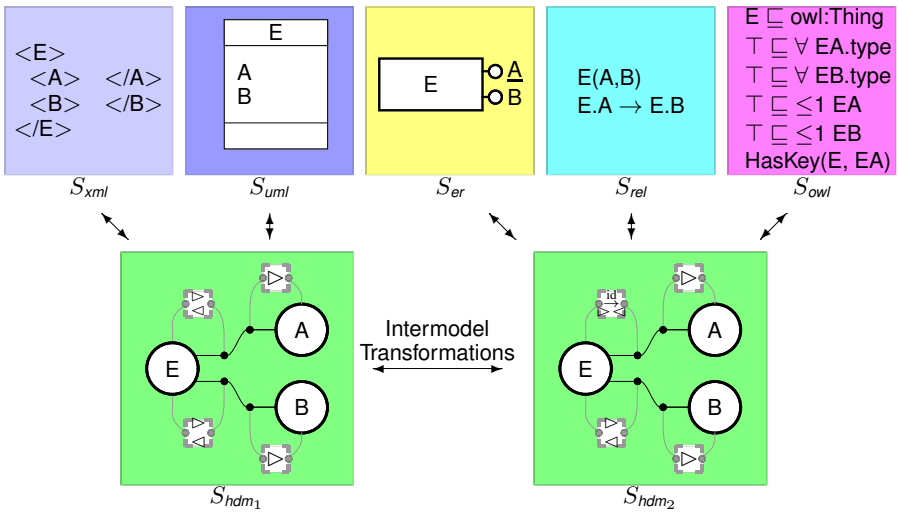


Fig. 1. Transforming between the OWL Model and various Data Models via the HDM

2 The Framework

A general framework for expressing the mappings between relational and OWL models is illustrated in Figure 2. Within this framework, two variants of the relational modelling language, that may be subject to this transformation, are identified. Firstly, we identify a 'basic' relational modelling language r, consisting of tables, keys and foreign keys. The example schema S_r (listed in Figure 3) is expressed using this modelling language. Secondly, we identify an enhanced relational modelling language r^+, which in addition to tables, keys and foreign keys, uses triggers, views or constraints. Analogously, we also identify two variants of the OWL 2 language. The first variant, o, is a subset of the OWL 2 language, that previous approaches, *e.g.*, Spanos et al. (2012), have targeted when applying reverse engineering methods on relational models expressed in r. This consists of the OWL 2 constructs: Class, ObjectProperty, DataProperty, SubClassOf, AllValuesFrom, Cardinality, InverseOf, OneOf and HasKey. The example schema S_o (listed in Figure 6) is expressed in such a restricted OWL 2 language. The second variant, o^+, is an enhanced OWL 2 language, containing rich axioms such as Transitive property and PropertyChain as well as others. We refer to a schema using more complex OWL 2 modelling constructs as S_{o^+}, and its equivalent relational schema S_{r^+}.

Previous approaches which transform relational schemas to OWL, start with schemas in the basic relational model such as S_r, and transform them to equivalent schemas S_o. Approaches that transform in the opposite direction, *i.e.*, OWL to relational, start with schemas such as S_{o^+}, and map them to schemas S_{r^+}.

Thus, previous work on transforming in each direction deals with modelling languages at different levels of complexity, as represented by the two horizontal axes in Figure 2. These transformations are Information Preserving (IP) (Miller et al., 1994). Previous approaches do not deal with the problem of transforming between the two levels (along the vertical axes in Figure 2), which would not be IP.

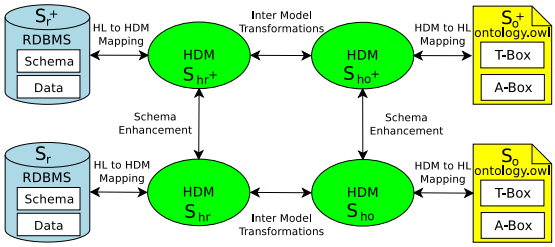


Fig. 2. A General Framework for Transforming between Relational and OWL

Our approach, instead, uses the HDM to represent the four modelling language variants, where S_{hr} is the HDM equivalent of S_r, S_{ho} is the HDM equivalent of S_o, and so on. We then focus on mapping between such HDM schemas. Our approach also benefits from being able to be utilised in different types of applications:

1. **Integrating existing databases and ontologies:** Our approach can integrate existing relational and OWL schemas. Usually, the relational schema will be of the form S_r and the OWL ontology S_{o^+}. We translate $S_r \rightarrow S_{hr} \rightarrow S_{ho}$ and translate $S_{o^+} \rightarrow S_{ho^+}$, and then have the task of reconciling the two schemas[2] S_{ho} and S_{ho^+}. This application will be left for future work.
2. **Creating a database from an existing ontology:** Transforming an OWL knowledge base to a relational database can be implemented by following a transformation pathway in the order of $S_{o^+} \rightarrow S_{ho^+} \rightarrow S_{hr^+} \rightarrow S_{r^+}$. In previous work (Al Khuzayem et al., 2013), we have shown the details of this pathway, and suggested two alternatives for implementing the expressive constructs of OWL 2 RL (Motik et al., 2009), a profile of OWL 2, as triggers or constraints in the relational database.
3. **Extracting a domain-specific ontology from an existing database:** This application is the focus of this paper, in which we implement a $S_r \rightarrow S_{hr} \rightarrow S_{ho} \rightarrow S_{ho^+} \rightarrow S_{o^+}$ transformation pathway. The process of transforming S_r to S_{ho} was reported in previous work (Boyd and McBrien, 2005; Al Khuzayem and McBrien, 2012), so we concentrate on the process of $S_{ho} \rightarrow S_{ho^+}$. Like any other work in the area, we assume that the ontologies make the **unique name assumption** (**UNA**), and that the databases are in **third normal form** (**3NF**). Approaches that perform 3NF normalisation exist in the literature (Yazici and Karakaya, 2007).

3 Preliminaries

An overview of the HDM (Poulovassilis and McBrien, 1998; Boyd and McBrien, 2005; Smith and McBrien, 2006), and how the relational model is represented in the HDM is shown here. An HDM schema S is defined as a tuple $\langle Nodes, Edges, Cons, Types \rangle$ where:

[2] In principle this task could also be done using S_r and S_{hr^+}, but these models do not include the OWL constructs that need to be changed.

- $Nodes$ is a set of nodes in the graph such that each node $\mathsf{node}{:}\langle\langle \mathsf{n}, \mathsf{t}\rangle\rangle$ is identified by its name n, and it is given an associated type $\mathsf{t} \in Types$. A node can also be referred to by a shorthand $\mathsf{node}{:}\langle\langle \mathsf{n}\rangle\rangle$.
- $Types$ is a tuple that contains a finite set of types and a subset of the set of all possible data values consistent with this type.
- $Edges$ is a set of edges in the graph such that each edge has the following scheme: $\mathsf{edge}{:}\langle\langle \mathsf{e}_1, \langle\langle \mathsf{n}_1\rangle\rangle, \langle\langle \mathsf{n}_2\rangle\rangle\rangle\rangle$, where e_1 is the edge's name (can also be unnamed as "_") and n_1 and n_2 are the two nodes that it connects.
- $Schemes$ is the union of $Nodes$ and $Edges$.
- $Cons$ is a set of boolean-valued functions (constraints) where they form the HDM constraint language. The set of constraints used in this paper are:
  - $\mathsf{cons}{:}\langle\langle \subseteq, \mathsf{s}_1, \mathsf{s}_2\rangle\rangle$ is the **inclusion** constraint which states that scheme s_1 is always a subset of scheme s_2.
  - $\mathsf{cons}{:}\langle\langle \not\cap, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{n}\rangle\rangle$ is the **exclusion** constraint which states that all the associate schemes are disjoint from each other.
  - $\mathsf{cons}{:}\langle\langle \cup, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{n}, \mathsf{s}\rangle\rangle$ is the **union** constraint stating scheme s as the union of schemes $s_1, \ldots, s_n$.
  - $\mathsf{cons}{:}\langle\langle \rhd, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{m}, \mathsf{s}\rangle\rangle$ is the **mandatory** constraint stating that every combination of the values that appears in schemes $s_1, \ldots, s_m$ must appear in the edge s connecting those schemes.
  - $\mathsf{cons}{:}\langle\langle \lhd, \mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{m}, \mathsf{s}\rangle\rangle$ is the **unique** constraint stating that every combination of the values that appears in schemes s_1,. . .,s_m must appear no more than once in the edge s connecting those schemes.
  - $\mathsf{cons}{:}\langle\langle \overset{\text{id}}{\rightarrow}, \mathsf{s}_1, \mathsf{s}\rangle\rangle$ is the **reflexive** constraint, stating that any value in s_1 must appear reflexively in the edge s that connects it to s_1.

In addition to referring to schemes directly, constraints may also take joins, projections and selections of schemes as arguments.

3.1 Representing the Relational Model in the HDM

Table 1. Rules for Representing a Relational Schema as an HDM Schema ($S_r \rightarrow S_{hr}$)

Relational Construct	HDM Representation
$\mathsf{table}\langle\langle T\rangle\rangle$	$\mathsf{node}{:}\langle\langle \mathsf{T}, \mathsf{any}\rangle\rangle$
$\mathsf{column}\langle\langle T, C, N, U, t\rangle\rangle$	$\mathsf{node}{:}\langle\langle \mathsf{T{:}C}, \mathsf{t}\rangle\rangle, \mathsf{edge}{:}\langle\langle \_, \mathsf{T}, \mathsf{T{:}C}\rangle\rangle, \mathsf{cons}{:}\langle\langle \rhd, \langle\langle \mathsf{T{:}C}\rangle\rangle, \langle\langle \_, \mathsf{T}, \mathsf{T{:}C}\rangle\rangle\rangle\rangle,$ $\mathsf{cons}{:}\langle\langle \lhd, \langle\langle \mathsf{T}\rangle\rangle, \langle\langle \_, \mathsf{T}, \mathsf{T{:}C}\rangle\rangle\rangle\rangle$
$N = \mathsf{notnull}$	$\mathsf{cons}{:}\langle\langle \rhd, \langle\langle \mathsf{T}\rangle\rangle, \langle\langle \_, \mathsf{T}, \mathsf{T{:}C}\rangle\rangle\rangle\rangle$
$U = \mathsf{unique}$	$\mathsf{cons}{:}\langle\langle \lhd, \langle\langle \mathsf{T{:}C}\rangle\rangle, \langle\langle \_, \mathsf{T}, \mathsf{T{:}C}\rangle\rangle\rangle\rangle$
$\mathsf{primary\_key}\langle\langle T, C\rangle\rangle$	$\mathsf{cons}{:}\langle\langle \overset{\text{id}}{\rightarrow}, \langle\langle \mathsf{T}\rangle\rangle, \langle\langle \_, \mathsf{T}, \mathsf{T{:}C}\rangle\rangle\rangle\rangle$
$\mathsf{primary\_key}\langle\langle T, C_1, C_2\rangle\rangle$	$\mathsf{cons}{:}\langle\langle \overset{\text{id}}{\rightarrow}, \langle\langle \mathsf{T}\rangle\rangle, \langle\langle \_, \mathsf{T}, \mathsf{T{:}C_1}\rangle\rangle \bowtie \langle\langle \_, \mathsf{T}, \mathsf{T{:}C_2}\rangle\rangle\rangle\rangle$
$\mathsf{foreign\_key}\langle\langle FK, T, C, T_f, C_f\rangle\rangle$	$\mathsf{cons}{:}\langle\langle \subseteq, \pi_{\langle\langle \mathsf{T{:}C}\rangle\rangle}\langle\langle \_, \mathsf{T}, \mathsf{T{:}C}\rangle\rangle, \pi_{\langle\langle \mathsf{T_f{:}C_f}\rangle\rangle}\langle\langle \_, \mathsf{T_f}, \mathsf{T_f{:}C_f}\rangle\rangle\rangle\rangle$

A method, summarised in Table 1, for mapping relational schemas to HDM was defined in Poulovassilis and McBrien (1998) and Boyd and McBrien (2005). To illustrate the method, consider the relational schema S_r, depicted in Figure 3 (where primary keys

are underlined, and nullable column names are suffixed by a question mark), which represents a subset of the Northwind database[3]. Later we will show how this schema can be transformed to the OWL schema listed in Figure 6.

Each table is represented as an HDM node (illustrated in Figure 4 by a black outlined circle) with HDM type any. For example, the Emp table is represented by the HDM node node:⟨⟨Emp, any⟩⟩.

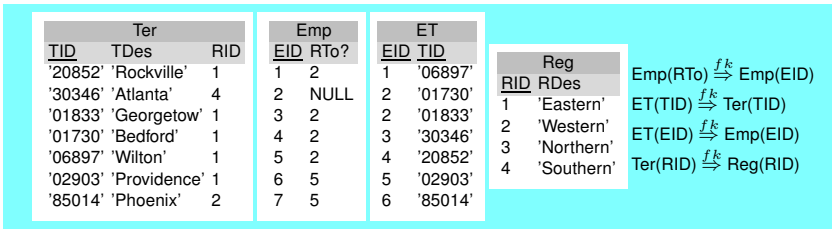
Each column is represented by a node that has an HDM type based on its relational type. For example, column Emp.EID has the type INTEGER, so it is represented in the HDM as node:⟨⟨Emp:EID, int⟩⟩. Moreover, Emp.EID will be connected via an HDM edge, edge:⟨⟨_, Emp, Emp:EID⟩⟩ (illustrated in Figure 4 with a thick black line), to the node that represents the column's table. Since column values only appear with an instance of a table tuple, the edge has a mandatory constraint (illustrated with grey lines) from the column node as cons:⟨⟨▷, node:⟨⟨Emp:EID⟩⟩, edge:⟨⟨_, Emp, Emp:EID⟩⟩⟩⟩. Furthermore, the unique constraint cons:⟨⟨◁, node:⟨⟨Emp⟩⟩, edge:⟨⟨_, Emp, Emp:EID⟩⟩⟩⟩ ensures that each column has a single value per row.

If a column is not nullable, then it must also have a mandatory constraint. Thus, Emp.EID has cons:⟨⟨▷, node:⟨⟨Emp⟩⟩, edge:⟨⟨_, Emp, Emp:EID⟩⟩⟩⟩.

If the column is key, then we state that the column's edge is reflexive. This also applies to column Emp.EID: cons:⟨⟨$\overset{\text{id}}{\rightarrow}$, node:⟨⟨Emp⟩⟩, edge:⟨⟨_, Emp, Emp:EID⟩⟩⟩⟩.

Finally, foreign keys are represented as inclusion constraints. Thus, for the foreign key between Emp.RTo and Emp.EID, we create the following HDM constraint: cons:⟨⟨⊆, node:⟨⟨Emp:RTo⟩⟩, node:⟨⟨Emp:EID⟩⟩⟩⟩.

The result of these transformations is an HDM graph (depicted in Figure 4) that is a forest of two-level trees, with subset constraints linking the leaf nodes.



Ter

TID	TDes	RID
'20852'	'Rockville'	1
'30346'	'Atlanta'	4
'01833'	'Georgetow'	1
'01730'	'Bedford'	1
'06897'	'Wilton'	1
'02903'	'Providence'	1
'85014'	'Phoenix'	2

Emp

EID	RTo?
1	2
2	NULL
3	2
4	2
5	2
6	5
7	5

ET

EID	TID
1	'06897'
2	'01730'
2	'01833'
3	'30346'
4	'20852'
5	'02903'
6	'85014'

Reg

RID	RDes
1	'Eastern'
2	'Western'
3	'Northern'
4	'Southern'

Emp(RTo) $\overset{fk}{\Rightarrow}$ Emp(EID)
ET(TID) $\overset{fk}{\Rightarrow}$ Ter(TID)
ET(EID) $\overset{fk}{\Rightarrow}$ Emp(EID)
Ter(RID) $\overset{fk}{\Rightarrow}$ Reg(RID)

Fig. 3. S_r, Fragment of the Northwind Relational Database Schema and Data. Entries in the Employee table are related to entries in the Territories table via ET. Each employee may optionally (indicated by a question mark) be recorded as reporting to another employee by the RTo. Each territory is in exactly one Region.

4 The Automatic Relational to OWL Transformation

Our automatic ontology extraction approach is performed via a number of steps: i) transform the relational constructs to HDM producing S_{hr}, which was described in

[3] https://northwinddatabase.codeplex.com/

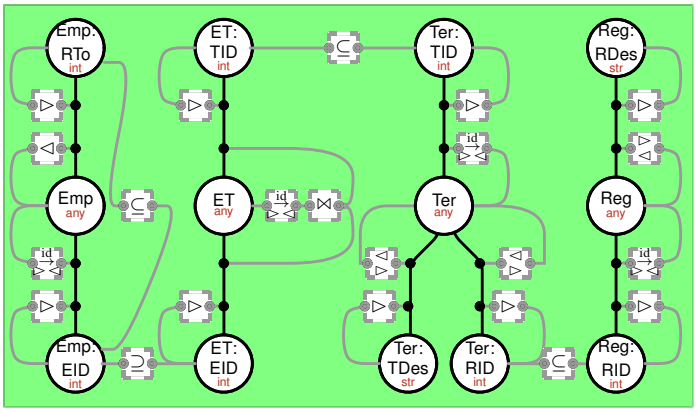


Fig. 4. S_{hr}, HDM Representation of Schema S_r

Section 3.1, ii) perform intermodel transformations on the HDM schema producing S_{ho} and finally, iii) translate the S_{ho} to an OWL schema. The subsections below, explain the final two steps.

4.1 The HDM Intermodel Transformations

Our HDM intermodel transformation process aims at overcoming the fundamental differences between the relational and OWL modelling languages. We use a set of BAV equivalence mappings presented in Boyd and McBrien (2005) to transform the HDM schema, S_{hr} (depicted in Figure 4) which represents the relational schema, to an equivalent (in terms of information capacity) HDM graph which represents the OWL schema.

(A) **Transform an attribute that is a foreign key, but not a primary key, into an ObjectProperty**. To be precise, if the relational schema contains $\mathsf{R}(\underline{\mathsf{K_R}}, \mathsf{K_S}, \ldots)$, $\mathsf{S}(\underline{\mathsf{K_S}}, \ldots)$, $\mathsf{R}(\mathsf{K_S}) \overset{\mathsf{fk}}{\Rightarrow} \mathsf{S}(\mathsf{K_S})$, then in OWL, we represent the $\mathsf{K_S}$ attribute of R as a functional property $\mathsf{P_{RS}}$ between the two classes R and S.

This can be achieved using two BAV equivalence rules (shown below); Inclusion Merge and Unique-Mandatory Redirection. First, applying the **Inclusion Merge** rule on the foreign key represented by the inclusion constraint between node:⟨⟨ET:EID⟩⟩ and node:⟨⟨Emp:EID⟩⟩, will result in merging those two nodes. The mandatory constraint between the node:⟨⟨ET:EID⟩⟩ and the edge:⟨⟨_, ET, ET:EID⟩⟩ is dropped, and any edges or constraints that apply to node:⟨⟨ET:EID⟩⟩ are redirected to node:⟨⟨Emp:EID⟩⟩. Subsequently, the **Unique-Mandatory Redirection** rule allows moving the newly redirected edge (between node:⟨⟨ET⟩⟩ and node:⟨⟨Emp:EID⟩⟩), to become instead between node:⟨⟨ET⟩⟩ and node:⟨⟨Emp⟩⟩. This is due to having unique and mandatory constraints on edge:⟨⟨_, ET, ET:EID⟩⟩. Because the redirected edge now connects two nodes representing classes, it represents an ObjectProperty. All other foreign keys of this type are transformed similarly.

① inclusion_merge(node:⟨⟨Emp:EID⟩⟩, edge:⟨⟨ET:EID, ET, ET:EID⟩⟩)
② unique_mandatory_redirection(edge:⟨⟨ET_Emp:EID, ET, Emp:EID⟩⟩, edge:⟨⟨ET_Emp, ET, Emp⟩⟩)

(B) **Transform an attribute that is a foreign key and a primary key as a SubClassOf axiom**. To be precise, if the relational schema contains $R(\underline{K_R}, \ldots)$, $S(\underline{K_S}, \ldots)$, $R(K_R) \overset{fk}{\Rightarrow} S(K_S)$, then in OWL, we represent the K_R attribute of R by making class R a subclass of S.

This can be achieved by using the BAV equivalence rule, **Identity Node Merge** which allows us to move a foreign key, represented as an inclusion constraint from the column nodes to instead be between the class nodes and that will eventually be mapped into a SubClassOf in the OWL model. The Northwind database, however, does not include an example of this particular case.

(C) **Transform a many-to-many binary relationship table to an ObjectProperty**. To be precise, if the relational schema contains $R(\underline{K_R}, \ldots)$, $S(\underline{K_S}, \ldots)$, $T(\underline{K_R}, \underline{K_S})$, $T(K_R) \overset{fk}{\Rightarrow} R(K_R)$, $T(K_S) \overset{fk}{\Rightarrow} S(K_S)$, then in OWL, we represent the T and its attributes as a property P_T between classes R and S.

This can be achieved using the **Identity Edge Merge** rule (shown below), that allows us to map two edges and a node in the HDM to a single edge representing an object property in the OWL model. For instance, this rule allows us to replace node:⟨⟨ET⟩⟩, edge:⟨⟨_, ET, Emp⟩⟩ and edge:⟨⟨_, ET, Ter⟩⟩ with a single edge:⟨⟨Emp_Ter, Emp, Ter⟩⟩. This newly created edge represents an ObjectProperty since it connects two class nodes.

③ identity_edge_merge(edge:⟨⟨ET_Emp, ET, Emp⟩⟩, edge:⟨⟨ET_Ter, ET, Ter⟩⟩)

(D) **Transformation of attribute constraints**. In general, relational nullable attributes, represent things that may or may not exist in reality. Thus, we must rely on the user to confirm whether a nullable attribute must exist in reality, in which case, we must add a mandatory constraint to the HDM graph. For example, column Emp.RTo may additionally have: cons:⟨⟨▷, Emp, ⟨⟨RTo, Emp, Emp⟩⟩⟩⟩ which will eventually be mapped to a Cardinality in OWL. In the example, we have assumed that the user confirmed that employees do not have to report to another employee, and thus omit the constraint.

Figure 5, illustrates the result of these transformations. Remaining nodes and edges do not need to be transformed as they will be interpreted as classes, datatypes and data properties in the OWL model, as will be described in the next section.

4.2 Transforming the HDM Schema to an OWL Ontology

We now explain how HDM schemas in 'OWL Compatible' form, such as S_{ho} and S_{ho^+}, can be translated to corresponding OWL files S_o and S_{o^+}, respectively. In Table 2, we list some OWL 2 constructs and how we translate them into HDM. The four more important transformations are:

1. HDM nodes without a type restriction are transformed to OWL classes. Thus, the nodes node:⟨⟨Emp⟩⟩, node:⟨⟨Ter⟩⟩ and node:⟨⟨Reg⟩⟩ with HDM type any in Figure 5 are mapped to the OWL classes: Emp, Ter and Reg.

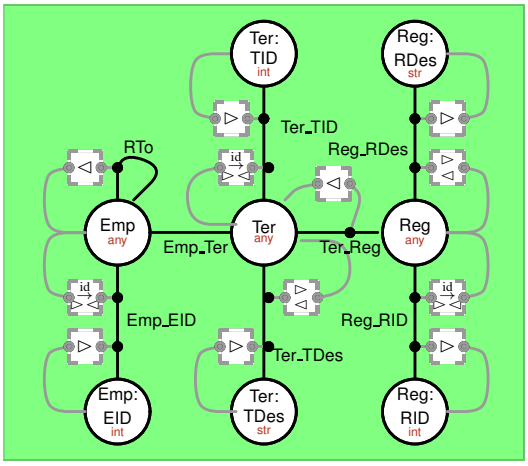


Fig. 5. S_{ho}, HDM Representation of Schema S_o

2. HDM nodes with HDM types are transformed to datatypes. For example, the node node:⟨⟨Emp_EID⟩⟩ with HDM type int is mapped to xsd:integer.
3. Edges which connect two HDM nodes, both without type restrictions, are transformed to object properties. For example, edge:⟨⟨Emp_Ter, Emp, Ter⟩⟩ is mapped to the ObjectProperty Emp_Ter.
4. Edges which connect nodes without type restriction, to nodes with a type restriction, are transformed to data properties. For example, the DataProperty Emp_EID, represents the edge:⟨⟨Emp_EID, Emp, Emp:EID⟩⟩.

Subsequently, combinations of HDM constraints are considered to represent more complex OWL 2 constructs, according to the mappings listed in Table 2. For instance, the combination of the following constraints cons:⟨⟨◁, EID, ⟨⟨Emp_EID, Emp, EID⟩⟩⟩⟩, cons:⟨⟨▷, EID, ⟨⟨Emp_EID, Emp, EID⟩⟩⟩⟩ and cons:⟨⟨$\overset{id}{\rightarrow}$, EID, ⟨⟨Emp_EID, Emp, EID⟩⟩⟩⟩, generates HasKey(Emp,Emp_EID). The complete OWL schema resulting from transforming S_{ho} is listed in Figure 6.

5 HDM Schema Enhancement

In this section, we outline our approach to enriching the ontology produced by the automatic transformation from a relational database. This step is performed by an analysis of the HDM schema S_{ho}, and produces an enriched schema S_{ho^+}. Our schema enhancement process uses schema analysis and data analysis techniques, which are well known in the database community for discovering implicit constructs from relational schemas and data (Cleve et al., 2011). Limited space permits us to list only a subset of the complete set of rules used in our prototype implementation.

Table 2. HDM Representations for Some OWL 2 Constructs

OWL 2 Construct	DL Syntax	HDM Representation
Class	C	node:$\langle\langle C\rangle\rangle$
SubClassOf (SCO)	$C_1 \sqsubseteq C_2$	cons:$\langle\langle \subseteq, C_1, C_2\rangle\rangle$
DisjointWith (DisW)	$C_1 \sqsubseteq \neg C_2$	cons:$\langle\langle \not\cap, C_1, C_2\rangle\rangle$
ObjectProperty (OP)	P	edge:$\langle\langle P, C_1, C_2\rangle\rangle$
DataProperty (DP)	R	edge:$\langle\langle R, C_1, \text{rdfs:Literal}\rangle\rangle$
FunctionalProperty (FOP)	$\top \sqsubseteq (\leq 1\ P)$	cons:$\langle\langle \lhd, C_1, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
InverseFunctionalProperty (InvF)	$\top \sqsubseteq (\leq 1\ P^-)$	cons:$\langle\langle \lhd, C_2, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
ReflexiveProperty (Refl)	$\top \sqsubseteq \exists\ P.Self$	cons:$\langle\langle \overset{id}{\rightarrow}, C_1, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
IrreflexiveProperty (Irre)	$\top \sqsubseteq \neg\exists\ P.Self$	cons:$\langle\langle \not\cap, C_1, \pi_{\langle C_1\rangle}\sigma_{\langle C_1=C_2\rangle}\ \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
SymmetricProperty (Symm)	$P \equiv P^-$	cons:$\langle\langle \subseteq, \pi_{\langle C_2,C_1\rangle}\ \langle\langle P, C_1, C_2\rangle\rangle, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
AsymmetricProperty (Asym)	$P \sqsubseteq \neg P^-$	cons:$\langle\langle \not\cap, \pi_{\langle C_2,C_1\rangle}\ \langle\langle P, C_1, C_2\rangle\rangle, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
TransitiveProperty (Tran)	$P \circ P \sqsubseteq P$	cons:$\langle\langle \subseteq, \pi_{\langle P/P_1.C_1,P/P_2.C_2\rangle}\ P \bowtie P, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
PropertyChain (ProC)	$P_1 \circ \ldots \circ P_n \sqsubseteq P$	cons:$\langle\langle \subseteq, \pi_{\langle P_1.C_1,P_n.C_{n+1}\rangle}\ P_1 \bowtie \ldots \bowtie P_n, \langle\langle P, C_1, C_{n+1}\rangle\rangle\rangle\rangle$
InverseOf (InvO)	$P \equiv Q^-$	cons:$\langle\langle \subseteq, \pi_{\langle C_2,C_1\rangle}\ \langle\langle P, C_1, C_2\rangle\rangle, \langle\langle Q, C_2, C_1\rangle\rangle\rangle\rangle$, cons:$\langle\langle \subseteq, \pi_{\langle C_1,C_2\rangle}\ \langle\langle Q, C_2, C_1\rangle\rangle, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
SubPropertyOf (SPO)	$P \sqsubseteq Q$	cons:$\langle\langle \subseteq, \langle\langle P, C_1, C_2\rangle\rangle, \langle\langle Q, C_3, C_4\rangle\rangle\rangle\rangle$
Cardinality (Crd)	$= nP$	node:$\langle\langle =nP\rangle\rangle$, edge:$\langle\langle P\_IE, =nP, \text{owl:Thing}\rangle\rangle$, cons:$\langle\langle \lhd^n, =nP, \langle\langle P\_IE, =nP, \text{owl:Thing}\rangle\rangle\rangle\rangle$, cons:$\langle\langle \rhd^n, =nP, \langle\langle P\_IE, =nP, \text{owl:Thing}\rangle\rangle\rangle\rangle$, cons:$\langle\langle \subseteq, \langle\langle P\_IE, =nP, \text{owl:Thing}\rangle\rangle, \langle\langle P, C, D\rangle\rangle\rangle\rangle$
MinCardinality (MinC)	$\geq nP$	node:$\langle\langle \geq nP\rangle\rangle$, edge:$\langle\langle P\_IE, \geq nP, \text{owl:Thing}\rangle\rangle$, cons:$\langle\langle \lhd^n, \geq nP, \langle\langle P\_IE, \geq nP, \text{owl:Thing}\rangle\rangle\rangle\rangle$, cons:$\langle\langle \subseteq, \langle\langle P\_IE, \geq nP, \text{owl:Thing}\rangle\rangle, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
MaxCardinality (MaxC)	$\leq nP$	node:$\langle\langle \leq nP\rangle\rangle$, edge:$\langle\langle P\_IE, \leq nP, \text{owl:Thing}\rangle\rangle$, cons:$\langle\langle \rhd^n, \leq nP, \langle\langle P\_IE, \leq nP, \text{owl:Thing}\rangle\rangle\rangle\rangle$, cons:$\langle\langle \subseteq, \langle\langle P\_IE, \leq nP, \text{owl:Thing}\rangle\rangle, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
UnionOf (UniO)	$C_1 \sqcup C_2$	cons:$\langle\langle \cup, C_1, C_2\rangle\rangle$
HasKey (Key)		cons:$\langle\langle \rhd, C_1, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$, cons:$\langle\langle \lhd, C_1, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$, cons:$\langle\langle \overset{id}{\rightarrow}, C_1, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$

5.1 Schema Analysis

In general, **Schema Analysis** (**SA**) can be anything from detecting similarities in names or value domains (Cleve et al., 2011) to spotting specific structural patterns. In our case, we use the structure of the schema to propose possible useful OWL 2 constructs. We represent our heuristics as rules of the form pattern $\rightsquigarrow$ action, where pattern is a pattern to match against the constructs of S_{ho}, and action is either a set of BAV transformations to apply to S_{ho}, or instructions to perform further data analysis (which if positive, will add BAV transformations to S_{ho}). Below, we list some of the SA heuristics we have implemented in our prototype.

Object Property Characteristics Heuristics: In OWL, the domain and range of Symmetric and Transitive properties must match (Lacy, 2005). Hence, when we search for such constraints on properties, we only need to consider a property P between a class D and itself, or between D and a superclass C of D. In addition, we propose the heuristic that Reflexive are most likely to occur on the same type of property, and hence also search for such constraints.

Given the above, we now consider whether P is Functional or not. A Functional property cannot be Transitive (Lacy, 2005), and is not useful if Reflexive (since it would only relate instances to themselves). Finally, Symmetric properties are less likely to be

$\exists$ RTo.$\top \sqsubseteq$ Emp (1)
$\top \sqsubseteq \forall$ RTo.Emp (2)
$\top \sqsubseteq (\leq 1$ RTo) (3)
$\exists$ Emp_EID.$\top \sqsubseteq$ Emp (4)
$\top \sqsubseteq \forall$ Emp_EID.xsd:integer (5)
$\top \sqsubseteq (\leq 1$ Emp_EID) (6)
$\exists$ Emp_Ter.$\top \sqsubseteq$ Emp (7)
$\top \sqsubseteq \forall$ Emp_Ter.Ter (8)
$\exists$ Ter_TID.$\top \sqsubseteq$ Ter (9)
$\top \sqsubseteq \forall$ Ter_TID.xsd:string (10)
$\top \sqsubseteq (\leq 1$ Ter_TID) (11)
$\exists$ Ter_TDes.$\top \sqsubseteq$ Territories (12)
$\top \sqsubseteq \forall$ Ter_TDes.xsd:string (13)
$\top \sqsubseteq (\leq 1$ Ter_TDes) (14)
$\exists$ Ter_Reg.$\top \sqsubseteq$ Ter (15)
$\top \sqsubseteq \forall$ Ter_Reg.Reg (16)
$\top \sqsubseteq (\leq 1$ Ter_Reg) (17)
$\exists$ Reg_RID.$\top \sqsubseteq$ Reg (18)
$\top \sqsubseteq \forall$ Reg_RID.xsd:integer (19)
$\top \sqsubseteq (\leq 1$ Reg_RID) (20)
$\exists$ Reg_RDes.$\top \sqsubseteq$ Reg (21)
$\top \sqsubseteq \forall$ Reg_RDes.xsd:string (22)
$\top \sqsubseteq (\leq 1$ Reg_RDes) (23)
HasKey(Emp, Emp_EID) (24)
HasKey(Ter, Ter_TID) (25)
HasKey(Reg, Reg_RID) (26)
Emp $\sqsubseteq \ = 1$ Reg_RID (27)
Ter $\sqsubseteq \ = 1$ Ter_TDes (28)

Fig. 6. S_o, OWL Schema (in DL syntax) Representing Schema S_r

$\top \sqsubseteq \neg \exists$ RTo.Self (29)
RTo $\sqsubseteq \neg$ RTo$^-$ (30)
RTo $\circ$ RTo $\equiv$ Emp_Emp (31)
$\top \sqsubseteq \forall$ Emp_Emp$^-$.Emp (32)
$\top \sqsubseteq \forall$ Emp_Emp.Emp (33)
Emp_Ter $\circ$ Ter_Reg $\equiv$ Emp_Reg (34)
$\top \sqsubseteq \forall$ Emp_Reg$^-$.Emp (35)
$\top \sqsubseteq \forall$ Emp_Reg.Reg (36)
RTo $\circ$ Emp_Ter $\equiv$ Emp_Ter_chain (37)
$\top \sqsubseteq \forall$ Emp_Ter_chain$^-$.Emp (38)
$\top \sqsubseteq \forall$ Emp_Ter_chain.Reg (39)
RTo $\circ$ RTo$^-$ $\equiv$ Emp_Emp_chain (40)
$\top \sqsubseteq \forall$ Emp_Emp_chain$^-$.Emp (41)
$\top \sqsubseteq \forall$ Emp_Emp_chain.Emp (42)

Fig. 7. Additional Axioms in Schema S_{o+}

Functional, since it would restrict the instances to be in pairs. Hence, we only search for these three types of property constraints when the edge is non-functional:

SA_1: edge:$\langle\langle P, D, D\rangle\rangle \wedge \neg$ cons:$\langle\langle \lhd, D, P\rangle\rangle \rightsquigarrow$
DA_{symm}(edge:$\langle\langle P, D, D\rangle\rangle$); DA_{refl}(edge:$\langle\langle P, D, D\rangle\rangle$); DA_{tran}(edge:$\langle\langle P, D, D\rangle\rangle$)

SA_2: edge:$\langle\langle P, C, D\rangle\rangle \wedge \neg$ cons:$\langle\langle \lhd, C, P\rangle\rangle \wedge$ cons:$\langle\langle \subseteq, D, C\rangle\rangle \rightsquigarrow$
DA_{symm}(edge:$\langle\langle P, C, D\rangle\rangle$); DA_{refl}(edge:$\langle\langle P, C, D\rangle\rangle$); DA_{tran}(edge:$\langle\langle P, C, D\rangle\rangle$)

Note that DA_{symm}, DA_{refl}, and DA_{tran} are further data analysis heuristics that will be described in the next section.

If circumstances where a property might be Symmetric or Reflexive, we consider it sensible to also identify whether they are Asymmetric or Irreflexive properties. Here, it is possible that the property might be Functional, so there is no restriction on the property being Functional in the rules.

SA_3: edge:$\langle\langle P, D, D\rangle\rangle \rightsquigarrow$
DA_{asym}(edge:$\langle\langle P, D, D\rangle\rangle$); DA_{irre}(edge:$\langle\langle P, D, D\rangle\rangle$)

SA_4: edge:$\langle\langle P, C, D\rangle\rangle \wedge$ cons:$\langle\langle \subseteq, D, C\rangle\rangle \rightsquigarrow$
DA_{asym}(edge:$\langle\langle P, C, D\rangle\rangle$); DA_{irre}(edge:$\langle\langle P, C, D\rangle\rangle$)

Again, DA_{asym} and DA_{irre} are further data analysis heuristics that will be described in the next section.

Property Chain Heuristics: PropertyChains of length two can, in principle, be formed from any two HDM edges that represent object properties, when the two edges share at least one common node. In the rules below, we identify cases in which a Prop-

ertyChain is more likely to have interesting semantics that should be represented in the OWL ontology.

SA_5 detects that a Functional property P_c, between a node and itself, represents a 'child' type relationship, and hence we can chain the property with itself to form a PropertyChain P_g, representing a 'grandchild' type relationship (in the rules, the notation P_c/P_1 indicates aliasing of P_c as P_1 and similarly for the class C/C_1).

SA_5: edge:$\langle\langle P_c, C, C\rangle\rangle \wedge$ cons:$\langle\langle \lhd, C, P_c\rangle\rangle \rightsquigarrow$
addEdge($\langle\langle P_g, C, C\rangle\rangle$); addJoin($\langle\langle \bowtie, P_c/P_1, P_c/P_2\rangle\rangle$);
addProjection($\langle\langle \pi_{\langle P_1.C_1, P_2.C_2\rangle}, \langle\langle \bowtie, P_c/P_1, P_c/P_2\rangle\rangle\rangle\rangle$);
addCons($\langle\langle \subseteq, \langle\langle \pi_{\langle P_1.C_1, P_2.C_2\rangle}, \langle\langle \bowtie, P_c/P_1, P_c/P_2\rangle\rangle\rangle\rangle, \langle\langle P_g, C, C\rangle\rangle\rangle\rangle$)

For example, applying this rule to the HDM schema in Figure 5, the Functional property RTo from Emp to Emp can be chained with itself to give a PropertyChain, Emp_Emp (shown in rules (31)–(33) depicted in Figure 7), that relates an employee with his/her second-line managers.

SA_6 detects that a Functional property P_a, from class D, represents a kind of 'attribute' E of class D, and that if another class C is related to D by another property P, then a PropertyChain P_{da} can be formed making E a 'denormalised attribute' of C.

SA_6: edge:$\langle\langle P, C, D\rangle\rangle \wedge$ edge:$\langle\langle P_a, D, E\rangle\rangle \wedge$ cons:$\langle\langle \lhd, D, P_a\rangle\rangle \rightsquigarrow$
addEdge($\langle\langle P_{da}, C, E\rangle\rangle$); addJoin($\langle\langle \bowtie, P, P_a\rangle\rangle$);
addProjection($\langle\langle \pi_{\langle P.C, P_a.E\rangle}, \langle\langle \bowtie, P, P_a\rangle\rangle\rangle\rangle$);
addCons($\langle\langle \subseteq, \langle\langle \pi_{\langle P.C, P_a.E\rangle}, \langle\langle \bowtie, P, P_a\rangle\rangle\rangle\rangle, \langle\langle P_{da}, C, E\rangle\rangle\rangle\rangle$)

For example, we can create a PropertyChain, called Emp_Reg (shown in rules (34)–(36)), that concatenates Emp_Ter and Ter_Reg to relate employees to regions. The data in the Northwind database, lead to such employees being related to only one region (validating that it is an interesting type of property to detect).

SA_7 detects that a Functional property P_a, from class C, represents a kind of 'attribute' D of class C, and that if another class E is related to D by another property P, then a PropertyChain P_{ca} can be formed.

SA_7: edge:$\langle\langle P_a, C, D\rangle\rangle \wedge$ edge:$\langle\langle P, D, E\rangle\rangle \wedge$ cons:$\langle\langle \lhd, C, P_a\rangle\rangle \rightsquigarrow$
addEdge($\langle\langle P_{ca}, C, E\rangle\rangle$); addJoin($\langle\langle \bowtie, P_a, P\rangle\rangle$);
addProjection($\langle\langle \pi_{\langle P_a.C, P.E\rangle}, \langle\langle \bowtie, P_a, P\rangle\rangle\rangle\rangle$);
addCons($\langle\langle \subseteq, \langle\langle \pi_{\langle P_a.C, P.E\rangle}, \langle\langle \bowtie, P_a, P\rangle\rangle\rangle\rangle, \langle\langle P_{ca}, C, E\rangle\rangle\rangle\rangle$)

For example, we can create a PropertyChain, called Emp_Ter_chain (shown in rules (37)–(39)), that concatenates the two properties RTo and Emp_Ter. This produces a property that relates an employee with his/her manager's territories.

SA_8 detects that where a class C is related to another class D via a Functional property P, we can treat D as being a 'parent' of C, and form a PropertyChain P_s between the edge and its inverse, to represent instances of C that are 'siblings' of each other (in the rules, the notation P/P_1 indicates aliasing of P as P_1).

SA_8: edge:$\langle\langle P, C, D\rangle\rangle \wedge$ cons:$\langle\langle \lhd, C, P\rangle\rangle \rightsquigarrow$
addEdge($\langle\langle P_s, C, C\rangle\rangle$); addJoin($\langle\langle \bowtie_D, P/P_1, P/P_2\rangle\rangle$);
addProjection($\langle\langle \pi_{\langle P_1.C, P_2.C\rangle}, \langle\langle \bowtie_D, P/P_1, P/P_2\rangle\rangle\rangle\rangle$);
addCons($\langle\langle \subseteq, \langle\langle \pi_{\langle P_1.C, P_2.C\rangle}, \langle\langle \bowtie_D, P/P_1, P/P_2\rangle\rangle\rangle\rangle, \langle\langle P_s, C, C\rangle\rangle\rangle\rangle$)

For example, chaining the property RTo with its inverse property RTo^- results in a self-referencing PropertyChain on Emp, called Emp_Emp_chain (shown in rules (40)–

(42)), which contains the employees that are related to each other through a particular manager (*i.e.*, colleagues).

Class Axioms Heuristic: If two classes, C_1 and C_2, have SubClassOf axioms with the same class C, then C_1 and C_2 may intersect with each other, or they may be disjoint and/or class C might be their union. These cases can be validated by data analysis.

SA_9: $\mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{C}_1, \mathsf{C}\rangle\!\rangle \wedge \mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{C}_2, \mathsf{C}\rangle\!\rangle \rightsquigarrow$
$\mathsf{DA}_{\mathsf{disj}}(\mathsf{node}{:}\langle\!\langle \mathsf{C}_1\rangle\!\rangle, \mathsf{node}{:}\langle\!\langle \mathsf{C}_2\rangle\!\rangle)$; $\mathsf{DA}_{\mathsf{union}}(\mathsf{node}{:}\langle\!\langle \mathsf{C}_1\rangle\!\rangle, \mathsf{node}{:}\langle\!\langle \mathsf{C}_2\rangle\!\rangle, \mathsf{node}{:}\langle\!\langle \mathsf{C}\rangle\!\rangle)$;
$\mathsf{DA}_{\mathsf{inter}}(\mathsf{node}{:}\langle\!\langle \mathsf{C}_1\rangle\!\rangle, \mathsf{node}{:}\langle\!\langle \mathsf{C}_2\rangle\!\rangle, \mathsf{node}{:}\langle\!\langle \mathsf{C}\rangle\!\rangle)$

5.2 Data Analysis

Data Analysis (**DA**) means mining the database content in order to detect possible, useful, implicit properties (Cleve et al., 2011). Conducting a DA technique on its own can be very expensive and time consuming. Therefore, instead of using DA *per se*, it is usually applied as a confirmation mechanism for other detection techniques (Cleve et al., 2011). In our case, we use it as a supplementary method to the SA technique. Each DA heuristic rule has the form $\mathsf{pattern} : \mathsf{condition} \mid \mathsf{probability} \rightsquigarrow \mathsf{action}$ where pattern is a pattern to match a DA instruction from SA, condition is a query to execute against S_{ho} that must return true, and probability is a query generating a number $[0, 1]$. We rely on a domain expert to validate that proposed additions in action to the ontology are correct (guided by the probabilities associated with each rule).

Symmetric and Asymmetric Heuristics: DA heuristics such as $\mathsf{DA}_{\mathsf{symm}}$, $\mathsf{DA}_{\mathsf{asym}}$, $\mathsf{DA}_{\mathsf{refl}}$, $\mathsf{DA}_{\mathsf{irre}}$ and $\mathsf{DA}_{\mathsf{tran}}$ all take the same general form. We illustrate the approach here by presenting $\mathsf{DA}_{\mathsf{symm}}$ and $\mathsf{DA}_{\mathsf{asym}}$.

$\mathsf{DA}_{\mathsf{symm}}(\mathsf{edge}{:}\langle\!\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\!\rangle)$: $\{\langle y, x\rangle | \langle x, y\rangle \in \langle\!\langle \mathsf{P,C,D}\rangle\!\rangle\} - \langle\!\langle \mathsf{P,C,D}\rangle\!\rangle = \emptyset \mid \frac{\mathsf{totalS}-\mathsf{notS}}{\mathsf{totalS}} \rightsquigarrow$
$\mathsf{addCons}(\langle\!\langle \pi_{\langle \mathsf{D,C}\rangle}, \langle\!\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\!\rangle\rangle\!\rangle)$; $\mathsf{addCons}(\langle\!\langle \subseteq, \langle\!\langle \pi_{\langle \mathsf{D,C}\rangle}, \langle\!\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\!\rangle\rangle\!\rangle, \langle\!\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\!\rangle\rangle\!\rangle)$

$\mathsf{DA}_{\mathsf{asym}}(\mathsf{edge}{:}\langle\!\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\!\rangle)$: $\mathsf{notS} > 0 \wedge \mathsf{totalS} = 0 \mid 1 \rightsquigarrow$
$\mathsf{addCons}(\langle\!\langle \pi_{\langle \mathsf{D,C}\rangle}, \langle\!\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\!\rangle\rangle\!\rangle)$; $\mathsf{addCons}(\langle\!\langle \not\subset, \langle\!\langle \pi_{\langle \mathsf{D,C}\rangle}, \langle\!\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\!\rangle\rangle\!\rangle, \langle\!\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\!\rangle\rangle\!\rangle)$

where totalS is the number of symmetric instances of P calculated as:
$\mathsf{totalS} = 2.|\{\langle \mathsf{y}, \mathsf{x}\rangle | \langle \mathsf{x}, \mathsf{y}\rangle \in \langle\!\langle \mathsf{P,C,D}\rangle\!\rangle\} \cap \langle\!\langle \mathsf{P,C,D}\rangle\!\rangle|$
and notS is the number of non-symmetric instances of P, calculated as:
$\mathsf{notS} = \{\{\langle \mathsf{y}, \mathsf{x}\rangle | \langle \mathsf{x}, \mathsf{y}\rangle \in \langle\!\langle \mathsf{P,C,D}\rangle\!\rangle\} - \langle\!\langle \mathsf{P,C,D}\rangle\!\rangle\}$.

For example, an instance of employee l via the Functional property RTo may report to employee m (manager). The manager m, however, can not report back to l. This means that the property is Asymmetric. Moreover, an employee can not report to him/herself so, it is Irreflexive. These axioms have been added to the ontology as in the DL rules (29)–(30) shown in Figure 7.

DisjointWith Heuristic: Any two HDM nodes representing classes that have disjoint sets of instances can lead to an OWL DisjointWith being proposed. The general rule is of the form:

$\mathsf{DA}_{\mathsf{disj}}(\mathsf{node}{:}\langle\!\langle \mathsf{C}_1\rangle\!\rangle, \mathsf{node}{:}\langle\!\langle \mathsf{C}_2\rangle\!\rangle)$: $\langle\!\langle C_1\rangle\!\rangle \cap \langle\!\langle C_2\rangle\!\rangle = \emptyset \mid 1 \rightsquigarrow$
$\mathsf{addCons}(\langle\!\langle \not\subset, \mathsf{C}_1, \mathsf{C}_2\rangle\!\rangle)$

The Northwind database does not contain foreign keys that can be mapped to SubClassOf. However, if we envisage that the database contained: Places(PID, PName), Ter(TID) $\overset{fk}{\Rightarrow}$ Places(PID), Reg(RID) $\overset{fk}{\Rightarrow}$ Places(PID) then, we would have two SubClassOf axioms. Data analysis in this case, would reveal that classes Ter and Reg should be disjoint.

6 Results and Discussion

The approach presented in this paper was implemented as a tool called OWLRel. It builds on the AutoMed data integration system[4] (Boyd et al., 2004), and hence can work with databases in many formats (Postgres, Microsoft SQL Server . . . etc.).

6.1 The Test Databases

Both, the Northwind Database (part of which was used as the running example in this paper) and the Mondial Database[5] were tested on our system. Table 3 lists the schema sizes of those two relational databases along with the Wine Database (OWLRelWine), which was generated by our system from the Wine Ontology[6] through the path $S_{o+} \rightarrow S_{ho+} \rightarrow S_{hr+} \rightarrow S_{hr} \rightarrow S_r$. This path simply produces a database that is limited to tables, columns, PKs and FKs. All other OWL axioms are ignored.

Table 3. RDB and OWL Schema Details. OWL Schema Construct Names are Abbreviated as Listed in Table 2

	RDB Schema (S_r) Details					OWL Schema (S_o) Details						
Schema Name	S_r size	Table	Attribute	PK	FK	Class	OP	DP	FOP	Key	SCO	SPO	Crd
Northwind	127	13	88	13	13	11	11	75	9	11	0	0	15
Mondial	139	20	69	19	31	19	30	41	26	18	0	0	48
OWLRelWine	676	147	164	147	218	137	16	1	6	0	186	5	0
Wine	-	-	-	-	-	138	16	1	6	0	200	5	-

Table 4. Number of Detected Axioms from the Enrichment Process. OWL Schema Construct Names are Abbreviated as Listed in Table 2

S_{o+}	InvO	Symm	Asym	Tran	Refl	Irre	Crd	MinC	MaxC	ProC	DisW	UniO
Northwind	0	0	1	0	0	1	6	1	1	17	0	0
Mondial	0	0	2	1	0	2	7	4	4	47	0	0
OWLRelWine	5	1	3	1	0	4	6	8	7	21	43	1
Wine	5	1	0	1	0	0	6	5	21	0	39	1

[4] automed.doc.ic.ac.uk
[5] http://www.dbis.informatik.uni-goettingen.de/Mondial/
[6] http://www.w3.org/TR/owl-guide/wine.rdf

6.2 OWLRel's Automatic Transformation Evaluation

Listed in Table 3, are the OWL schema details for the Northwind, the Mondial, the reconstructed Wine (OWLRelWine) and the original Wine (Wine) ontologies.

To evaluate our automatic transformation, we compared the original Northwind and Mondial databases to the ones that were created automatically from our system, by the reverse transformation of their respective ontologies. By inserting the Northwind and Mondial ontologies that resulted from the automatic transformation back into our system following the path: $S_o \rightarrow S_{ho} \rightarrow S_{hr} \rightarrow S_r$, we finished up with databases that are identical to the original ones.

6.3 OWLRel's Enrichment Evaluation

In Table 4, we list the number of axioms proposed from the schema enhancement phase for all three databases mentioned earlier. Regarding the Northwind and Mondial databases, we used basic human judgment to check whether a proposed axiom was useful or not. If the data in the database is not complete, our system might detect a wrong axiom. For example, our system suggested that a property like mergesWith with domain and range Sea in the Mondial ontology is Asymmetric, while in reality it should be Symmetric. Regarding the Wine database, we used the original Wine ontology (Wine) as a reference model and compared our reconstructed Wine ontology (OWLRelWine) against it. Comparing OWLRelWine which resulted from the automatic transformation, with the original Wine ontology after running the Pellet reasoner (Sirin et al., 2007), produced 100% precision and 2.59% recall. After our schema enhancement phase, the precision dropped to 99.25%, but the recall ascended to 77.35%. Although the precision had dropped, we consider that the extra axioms that our system has generated are infact correct and useful. For example, the system detected the properties hasMaker, producesWine and locatedIn as both Irreflexive and Asymmetric which in reality is true. Moreover, all proposed PropertyChain axioms were interesting. For instance, the system proposed chaining the two properties, hasMaker and producesWine, which resulted in a property containing wines produced by the same maker.

7 Related Work

Most of the approaches that extract a domain-specific ontology by reverse engineering a database, such as: Sequeda et al. (2012), Astrova (2004) and Tirmizi et al. (2008), suffer from one or more of the following problems (Spanos et al., 2012): 1) Do not map highly expressive OWL features. 2) Miss interpret primary keys. 3) Do not properly identify class hierarchies.

The survey (Spanos et al., 2012) points out that transforming a relational schema on its own is not enough, and some sort of data inspection is needed in order to solve problems 1 & 3. Problem 3 was addressed by the work of Astrova (2004) and Cerbah (2008) in which they apply data mining techniques to detect class hierarchies. Moreover, primary keys have been interpreted in various ways in the OWL model. Some approaches translate a PK into an InverseFunctional DataProperty with MinCardinality

1, or use MinCardinality and MaxCardinality 1, or use Functional property and Cardinality 1. Only a few, like us, have considered using the HasKey construct such as Sequeda et al. (2012) and Vysniauskas et al. (2011). Furthermore, eliciting the highly expressive constructs of OWL, such as HasValue and Transitive and Symmetric properties, was only considered by Astrova et al. (2007). However, their rules for generating these constructs are ambiguous and do not rely on data analysis. In contrast to this, we have proposed an approach that analyses the schema and the data to suggest possible highly expressive OWL 2 features for a given RDBMS and represented primary keys using the HasKey construct, thus providing solutions for problems 1 & 2 above.

Using an intermediate language when mapping between relational and OWL was considered in the work of Atzeni et al. (2008) and Dadjoo and Kheirkhah (2015). The former work, however, is the only work that provides a BT via an intermediate language. Nevertheless, their work is restricted to OWL-Lite and does not consider data analysis.

8 Conclusions and Future Work

Approaches that transform relational databases to ontologies often drop many of the expressive features of OWL. This is because, generally, relational databases are in a basic form and thus, transforming them would produce an ontology with limited expressibility. In this paper, we have presented a conceptual framework and its current implementation that maps a relational schema to an exact OWL schema and then enhance it with rich OWL 2 constructs, detected from schema and data analysis. We then rely on the user for the verification of these features. Testing the approach on a number of medium-sized databases showed very promising results. Future work will be directed towards adding more heuristics, carrying-out more exhaustive evaluation and encorporating machine learning.

References

Al Khuzayem, L., Liu, Y. and McBrien, P. (2013), Transforming OWL 2 RL Schemas to Relational Schemas with Open-world or Closed-world Semantics, Technical report, AutoMed 38. http://www.doc.ic.ac.uk/automed/techreports/owl2rltorelational.ps.

Al Khuzayem, L. and McBrien, P. (2012), Knowledge Transformation using a Hypergraph Data Model, *in* 'ICCSW', pp. 1–7.

Astrova, I. (2004), Reverse Engineering of Relational Databases to Ontologies, *in* 'The Semantic Web: Research and Applications', Springer, pp. 327–341.

Astrova, I., Korda, N. and Kalja, A. (2007), Rule-Based Transformation of SQL Relational Databases to OWL Ontologies, *in* 'Proc. of the 2nd Inter. Conf. on Metadata & Semantics Research'.

Atzeni, P., Del Nostro, P. and Paolozzi, S. (2008), Ontologies and Databases: Going Back and Forth, *in* 'Proc. of the 4th ODBIS', Citeseer.

Boyd, M., Kittivoravitkul, S., Lazanitis, C., McBrien, P. and Rizopoulos, N. (2004), AutoMed: A BAV Data Integration System for Heterogeneous Data Sources, *in* 'Advanced Information Systems Engineering', Springer, pp. 511–524.

Boyd, M. and McBrien, P. (2005), 'Comparing and Transforming between Data Models via an Intermediate Hypergraph Data Model', *DS* **IV**, 69–109.

Cerbah, F. (2008), Mining the Content of Relational Databases to Learn Ontologies with Deeper Taxonomies, *in* 'Inter. Conf. on WI-IAT.', Vol. 1, IEEE, pp. 553–557.

Cleve, A., Meurisse, J.-R. and Hainaut, J.-L. (2011), Database Semantics Recovery through Analysis of Dynamic SQL Statements, *in* 'Data Semantics XV', Springer, pp. 130–157.

Dadjoo, M. and Kheirkhah, E. (2015), 'An Approach for Transforming of Relational Databases to OWL Ontology', *International Journal of Web & Semantic Technology (IJWesT)* **06**(01), 19–28.

El Idrissi, B., Baina, S. and Baina, K. (2013), Automatic Generation of Ontology from Data Models: A Practical Evaluation of Existing Approaches, *in* 'IEEE 7th Inter. Conf. on RCIS', pp. 1–12.

Lacy, L. (2005), *OWL: Representing Information Using the Web Ontology Language*, Trafford Publishing, Victoria BC, Canada.

McBrien, P. and Poulovassilis, A. (2001), A Semantic Approach to Integrating XML and Structured Data Sources, *in* 'Proc. of CAiSE', Vol. 2068 of *LNCS*, pp. 330–345.

McBrien, P. and Poulovassilis, A. (2003), Data Integration by Bi-Directional Schema Transformation Rules, *in* 'Proc. of ICDE', IEEE, pp. 227–238.

Miller, R., Ioannidis, Y. and Ramakrishnan, R. (1994), 'Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice', *IS* **19**(1), 3–31.

Motik, B., Grau, B., Horrocks, I., Wu, Z., Fokoue, A. and Lutz, C. (2009), 'OWL 2 Web Ontology Language Profiles'. http://www.w3.org/2007/OWL/draft/ED-owl2-profiles-20090420/all.pdf.

Poulovassilis, A. and McBrien, P. (1998), 'A General Formal Framework for Schema Transformation', *DKE* **28**(1), 47–71.

Sequeda, J., Arenas, M. and Miranker, D. (2012), On Directly Mapping Relational Databases to RDF and OWL, *in* 'Proc. of the 21st WWW', pp. 649–658.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A. and Katz, Y. (2007), 'Pellet: A Practical OWLDL Reasoner', *WS* **5**(2), 51–53.

Smith, A. C. and McBrien, P. (2006), Inter Model Data Exchange of Type Information via a Common Type Hierarchy, *in* 'DISWEB'.

Spanos, D.-E., Stavrou, P. and Mitrou, N. (2012), 'Bringing Relational Databases into the Semantic Web: A Survey', *Semantic Web* **3**(2), 169–209.

Tirmizi, S. H., Sequeda, J. and Miranker, D. (2008), Translating SQL Applications to the Semantic Web, *in* 'DEXA', Springer, pp. 450–464.

Vysniauskas, E., Nemuraite, L., Butleris, R. and Paradauskas, B. (2011), 'Reversible Lossless Transformation From OWL 2 Ontologies into Relational Databases', *IJITCA* **40**(4), 293–306.

W3C (2009), 'OWL 2 Web Ontology Language New Features and Rationale'. http://www.w3.org/TR/2009/WD-owl2-new-features-20090611/.

Yazici, A. and Karakaya, Z. (2007), *Computational Science – ICCS 2007: 7th International Conference, Beijing, China, May 27 - 30, 2007, Proceedings, Part II*, Springer Berlin Heidelberg, Berlin, Heidelberg, chapter JMathNorm: A Database Normalization Tool Using Mathematica, pp. 186–193.

Received April 29, 2016 , accepted April 30, 2016