Databases and Information Systems IX
G. Arnicans et al. (Eds.)

doi:10.3233/978-1-61499-714-6-43

Extracting OWL Ontologies from Relational Databases Using Data Analysis and Machine Learning

Lama AL KHUZAYEM [a,b,1] and Peter MCBRIEN [a]

[a] *Department of Computing, Imperial College, London SW7 2AZ, UK*
[b] *Computer Science Department, Faculty of Computing and IT, King Abdulaziz University, Jeddah 21583, Saudi Arabia*

Abstract. Extracting OWL ontologies from relational databases is extremely helpful for realising the Semantic Web vision. However, most of the approaches in this context often drop many of the expressive features of OWL. This is because highly expressive axioms can not be detected from database schema alone, but instead require a combined analysis of the database schema and data. In this paper, we present an approach that transforms a relational schema to a basic OWL schema, and then enhances it with rich OWL 2 constructs using schema and data analysis techniques. We then rely on the user for the verification of these features. Furthermore, we apply machine learning algorithms to help in ranking the resulting features based on user supplied relevance scores. Testing our tool on a number of databases demonstrates that our proposed approach is feasible and effective.

Keywords. Ontology Learning, OWL 2 Ontologies, Data Analysis, Machine Learning.

1. Introduction

Constructing ontologies, which are the basic building blocks of the **Semantic Web** (**SW**), from scratch is tedious, error-prone, time-consuming, and requires deep understanding of the domain [1,2,3]. Attention has therefore shifted towards the generation of ontologies from existing data. The work in this paper focuses on extracting ontologies from relational models. The issue that arises in this context is that only ontologies with limited expressivity can be obtained in an automated fashion. Although leading ontology definition languages such as the **Web Ontology Language** (**OWL**) [4] have great expressive capabilities, they are currently not used to their full potential. This is because richer and more expressive axioms like PropertyChain, TransitiveProperty and SymmetricProperty require more advanced detection techniques than just analysing the database schema. Most previous work (e.g., [1,2,5,6]) have avoided this extra task, leaving it for the users to manually detect and insert the desired axioms.

[1]Corresponding Author: Lama Al Khuzayem, Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2AZ, United Kingdom; E-mail: l.al-khuzayem11@imperial.ac.uk.

The approach described in this paper, which is developed as a tool called OWLRel[2], takes this additional step and aims at finding more complex axioms through applying a combination of schema and data analysis techniques. This paper extends the work presented in [7], in which it was demonstrated how to automatically extract an OWL ontology from a relational schema by generating a **Bidirectional Transformation** (**BT**), using the **Hypergraph Data Model** (**HDM**) [8], then enriching the OWL schema with more expressive axioms using schema and data analysis. Because of the subjective nature of the domain, we rely on users to validate these suggested axioms. Thus, our approach benefits from the two aspects that must be taken into consideration in this context [3]: human intervention and database content analysis.

For example, consider the Emp table shown in the database schema depicted in Figure 1. From the database schema, we can generate an ontology that contains an Emp class from the Emp table. The EID column will be transformed into a data property. The RTo? column will be transformed into an object property. The schema does not say anything about the characteristics of the object property. However, by doing an inspection on the data, one can detect that the RTo object property is an AsymmetricProperty (if someone reports to his/her manager, the manager does not report back to that person) and an IrreflexiveProperty (a person does not report back to himself). Our approach also applies schema analysis heuristics, which detects that chaining the property RTo with itself, or with its inverse, produces new properties that could possibly be useful or interesting. For instance, chaining RTo with RTo produces a property that relates employees with their second line managers, while chaining RTo with RTo^{-} produces a property that contains the employees that are colleagues of each other. These proposals would not be possible without the use of schema and data analysis.

One problem that arises in this situation is that the large number of suggested axioms (where the schema is large) might overwhelm the user. Hence, in this paper, we focus on how we can help the users by filtering or sorting the suggestions according to their relevance. Since relevance is very subjective, **Machine Learning** (**ML**) algorithms were used to rank the suggested axioms taking into consideration knowledge gathered from previous users over multiple sessions. Our tool is also able to continuously learn from its new users, which helps in improving the results over time.

The remainder of this paper is structured as follows. Section 2 reviews the HDM and how the relational model is presented in HDM. Section 3 describes our automatic ontology extraction approach and in Section 4, we demonstrate our ontology enrichment process. Section 5 explains how we apply machine learning to our extraction approach followed by related work in Section 6. Finally, Section 7 concludes this paper.

2. Preliminaries

An overview of the HDM [8,9,10], and how the relational model is represented in the HDM is given here. An HDM schema S is defined as a tuple $\langle Nodes, Edges, Cons, Types \rangle$ where:

- $Nodes$ is a set of nodes in the graph such that each node $\text{node:}\langle\langle \text{n}, \text{t} \rangle\rangle$ is identified by its name n, and it is given an associated type $\text{t} \in Types$. A node can also be referred to by a shorthand $\text{node:}\langle\langle \text{n} \rangle\rangle$.

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

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

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

- *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: $\mathsf{edge}{:}\langle\langle \mathsf{e_1}, \mathsf{n_1}, \mathsf{n_2}\rangle\rangle$, where $\mathsf{e_1}$ is the edge's name (can also be unnamed as "_") and $\mathsf{n_1}$ and $\mathsf{n_2}$ 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.
  - $\mathsf{cons}{:}\langle\langle \not\oslash, \mathsf{s_1}, \ldots, \mathsf{s_n}\rangle\rangle$ is the **exclusion** constraint which states that all the associate schemes are disjoint from each other.
  - $\mathsf{cons}{:}\langle\langle \cup, \mathsf{s}, \mathsf{s_1}, \ldots, \mathsf{s_n}\rangle\rangle$ is the **union** constraint stating scheme s as the union of schemes $s_1, \ldots, s_n$.
  - $\mathsf{cons}{:}\langle\langle \triangleright, \mathsf{s_1}, \ldots, \mathsf{s_m}, \mathsf{s}\rangle\rangle$ 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.
  - $\mathsf{cons}{:}\langle\langle \triangleleft, \mathsf{s_1}, \ldots, \mathsf{s_m}, \mathsf{s}\rangle\rangle$ 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.
  - $\mathsf{cons}{:}\langle\langle \overset{\mathrm{id}}{\rightarrow}, \mathsf{s_1}, \mathsf{s}\rangle\rangle$ 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.

2.1. Representing the Relational Model in the HDM

A method, summarised in Table 1, for mapping relational schemas to HDM was defined in [8] and [9]. To illustrate the method, consider the relational schema S_r, depicted in Figure 1 (where primary keys are underlined, and nullable column names are suffixed by a question mark), which represents a subset of the Northwind database[3].

Each table is represented as an HDM node (illustrated in Figure 2 by a black outlined circle) with HDM type any. For example, the Emp table is represented by the HDM node $\mathsf{node}{:}\langle\langle \mathsf{Emp}, \mathsf{any}\rangle\rangle$.

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

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 2 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 2) that is a forest of two-level trees, with subset constraints linking the leaf nodes.

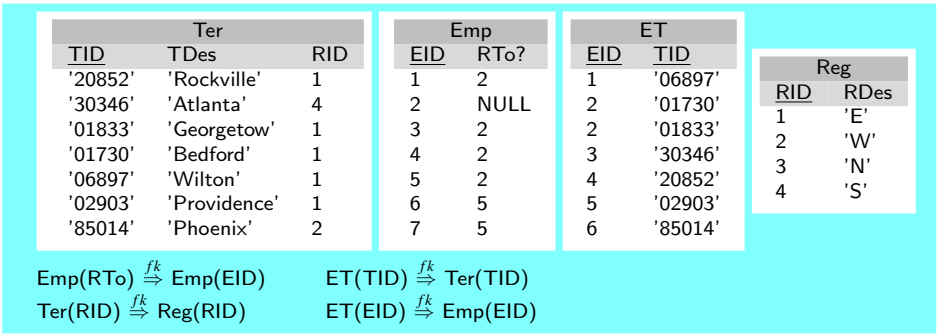

Figure 1. S_r, Fragment of the Northwind Relational Database Schema and Data. Entries in the Employee table are related to entries in the Territories table via ET. Each employee may optionally (indicated by a question mark) be recorded as reporting to another employee by the RTo. Each territory is in exactly one Region.

3. 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 Section 2.1, 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, review the final two steps which were explained in [7].

3.1. 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 equivalence

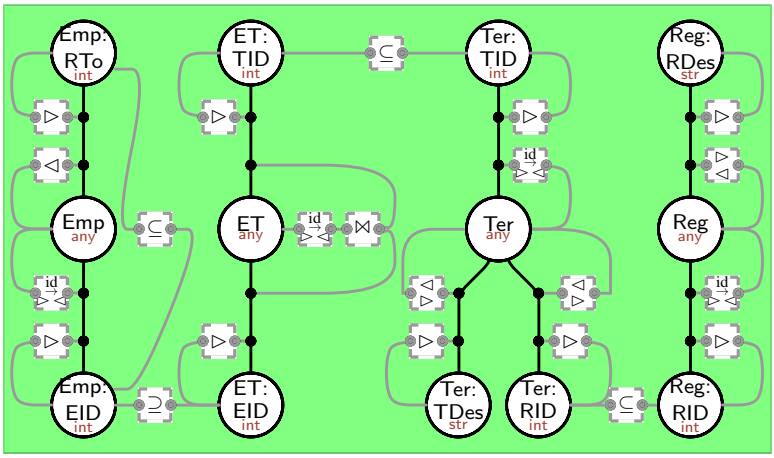

Figure 2. S_{hr}, HDM Representation of Schema S_r

both-as-view (**BAV**) mappings [11], presented in [9], to transform the HDM schema, S_{hr} (depicted in Figure 2) 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. This can be achieved using two BAV equivalence rules: Inclusion Merge and Unique-Mandatory Redirection. For instance, using the rules shown below, we can transform the foreign key represented by the inclusion constraint between node:⟨⟨ET:EID⟩⟩ and node:⟨⟨Emp:EID⟩⟩, to become an object property represented by the edge edge:⟨⟨_, ET, Emp⟩⟩. 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**. This can be achieved by using the BAV equivalence rule, Identity Node Merge. 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. This can be achieved using the Identity Edge Merge rule. For instance, the rule shown below 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 object property since it connects two class nodes.
③ identity_edge_merge(edge:⟨⟨ET_Emp, ET, Emp⟩⟩, edge:⟨⟨ET_Ter, ET, Ter⟩⟩)

Figure 3, 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.

3.2. Transforming the HDM Schema to an OWL Schema

We now review 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

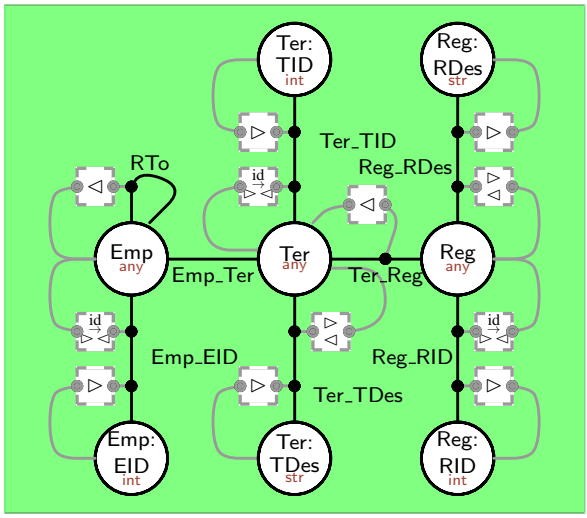

Figure 3. S_{ho}, HDM Representation of Schema S_o

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 3 are mapped to the OWL classes: Emp, Ter and Reg.
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 4.

4. HDM Schema Enhancement

This section presents an extension to the schema enhancement approach, presented in [7], in which we explain new heuristics. Our schema enhancement process applies schema analysis and data analysis heuristics on the HDM schema S_{ho} producing an enriched schema S_{ho^+}.

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	$C_1 \sqsubseteq C_2$	cons:$\langle\langle\subseteq, C_1, C_2\rangle\rangle$
ObjectProperty	P	edge:$\langle\langle P, C_1, C_2\rangle\rangle$
DataProperty	R	edge:$\langle\langle R, C_1, \text{rdfs:Literal}\rangle\rangle$
FunctionalProperty	$\top \sqsubseteq (\leq 1\ P)$	cons:$\langle\langle\lhd, C_1, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
TransitiveProperty	$P \circ P \sqsubseteq P$	cons:$\langle\langle\subseteq, \pi_{\langle P/P_1.C_1, P/P_2.C_2\rangle} P \bowtie P, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
IntersectionOf	$C_1 \sqcap C_2$	node:$\langle\langle NC_1\rangle\rangle$, cons:$\langle\langle\not\lhd, C_1, NC_1\rangle\rangle$, cons:$\langle\langle\cup, \text{owl:Thing}, C_1, NC_1\rangle\rangle$, node:$\langle\langle NC_2\rangle\rangle$, cons:$\langle\langle\not\lhd, C_2, NC_2\rangle\rangle$, cons:$\langle\langle\cup, \text{owl:Thing}, C_2, NC_2\rangle\rangle$, node:$\langle\langle UNC_1\_NC_2\rangle\rangle$, cons:$\langle\langle\cup, UNC_1\_NC_2, NC_1, NC_2\rangle\rangle$, node:$\langle\langle IC_1\_C_2\rangle\rangle$, cons:$\langle\langle\not\lhd, IC_1\_C_2, UNC_1\_NC_2\rangle\rangle$, cons:$\langle\langle\cup, \text{owl:Thing}, IC_1\_C_2, UNC_1\_NC_2\rangle\rangle$
HasValue	$\exists\ P.\{a\}$	node:$\langle\langle VP.a\rangle\rangle$, edge:$\langle\langle P\_IE, VP.a, a\rangle\rangle$, cons:$\langle\langle\rhd, VP.a, \langle\langle P\_IE, VP.a, a\rangle\rangle\rangle\rangle$, cons:$\langle\langle\subseteq, \langle\langle P\_IE, VP.a, a\rangle\rangle, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
Cardinality	$= nP$	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	$\geq nP$	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	$\leq nP$	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:$\langle\langle\subseteq, \langle\langle P\_IE, \leq nP, \text{owl:Thing}\rangle\rangle, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$
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:$\langle\langle\xrightarrow{\text{id}}, C_1, \langle\langle P, C_1, C_2\rangle\rangle\rangle\rangle$

$$\exists\ \text{RTo}.\top \sqsubseteq \text{Emp} \quad (1)$$
$$\top \sqsubseteq \forall\ \text{RTo}.\text{Emp} \quad (2)$$
$$\top \sqsubseteq (\leq 1\ \text{RTo}) \quad (3)$$
$$\exists\ \text{Emp\_EID}.\top \sqsubseteq \text{Emp} \quad (4)$$
$$\top \sqsubseteq \forall\ \text{Emp\_EID}.\text{xsd:integer} \quad (5)$$
$$\top \sqsubseteq (\leq 1\ \text{Emp\_EID}) \quad (6)$$
$$\exists\ \text{Emp\_Ter}.\top \sqsubseteq \text{Emp} \quad (7)$$
$$\top \sqsubseteq \forall\ \text{Emp\_Ter}.\text{Ter} \quad (8)$$
$$\exists\ \text{Ter\_TID}.\top \sqsubseteq \text{Ter} \quad (9)$$
$$\top \sqsubseteq \forall\ \text{Ter\_TID}.\text{xsd:string} \quad (10)$$
$$\top \sqsubseteq (\leq 1\ \text{Ter\_TID}) \quad (11)$$
$$\exists\ \text{Ter\_TDes}.\top \sqsubseteq \text{Territories} \quad (12)$$
$$\top \sqsubseteq \forall\ \text{Ter\_TDes}.\text{xsd:string} \quad (13)$$
$$\top \sqsubseteq (\leq 1\ \text{Ter\_TDes}) \quad (14)$$
$$\exists\ \text{Ter\_Reg}.\top \sqsubseteq \text{Ter} \quad (15)$$
$$\top \sqsubseteq \forall\ \text{Ter\_Reg}.\text{Reg} \quad (16)$$
$$\top \sqsubseteq (\leq 1\ \text{Ter\_Reg}) \quad (17)$$
$$\exists\ \text{Reg\_RID}.\top \sqsubseteq \text{Reg} \quad (18)$$
$$\top \sqsubseteq \forall\ \text{Reg\_RID}.\text{xsd:integer} \quad (19)$$
$$\top \sqsubseteq (\leq 1\ \text{Reg\_RID}) \quad (20)$$
$$\exists\ \text{Reg\_RDes}.\top \sqsubseteq \text{Reg} \quad (21)$$
$$\top \sqsubseteq \forall\ \text{Reg\_RDes}.\text{xsd:string} \quad (22)$$
$$\top \sqsubseteq (\leq 1\ \text{Reg\_RDes}) \quad (23)$$
$$\text{HasKey}(\text{Emp}, \text{Emp\_EID}) \quad (24)$$
$$\text{HasKey}(\text{Ter}, \text{Ter\_TID}) \quad (25)$$
$$\text{HasKey}(\text{Reg}, \text{Reg\_RID}) \quad (26)$$
$$\text{Emp} \sqsubseteq\ = 1\ \text{Reg\_RID} \quad (27)$$
$$\text{Ter} \sqsubseteq\ = 1\ \text{Ter\_TDes} \quad (28)$$

Figure 4. S_o, OWL Schema (in DL syntax) Representing Schema S_r

4.1. Schema Analysis

Our **Schema Analysis** (**SA**) heuristic rules, presented in this section, take 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 [12]. 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 is most likely to occur on the same type of property, hence also search for it.

Given the above, we now consider whether P is Functional or not. A Functional property cannot be Transitive [12], and is not useful if Reflexive (since it would only relate instances to themselves). Finally, Symmetric properties are less likely to be 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: $\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{D}, \mathsf{D}\rangle\rangle \wedge \neg\, \mathsf{cons}{:}\langle\langle \lhd, \mathsf{D}, \mathsf{P}\rangle\rangle \rightsquigarrow$
$\mathsf{DA}_{\mathsf{symm}}(\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{D}, \mathsf{D}\rangle\rangle); \mathsf{DA}_{\mathsf{refl}}(\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{D}, \mathsf{D}\rangle\rangle); \mathsf{DA}_{\mathsf{tran}}(\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{D}, \mathsf{D}\rangle\rangle)$

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

Note that $\mathsf{DA}_{\mathsf{symm}}$, $\mathsf{DA}_{\mathsf{refl}}$, and $\mathsf{DA}_{\mathsf{tran}}$ are 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: $\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{D}, \mathsf{D}\rangle\rangle \rightsquigarrow$
$\mathsf{DA}_{\mathsf{asym}}(\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{D}, \mathsf{D}\rangle\rangle); \mathsf{DA}_{\mathsf{irre}}(\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{D}, \mathsf{D}\rangle\rangle)$

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

Again, $\mathsf{DA}_{\mathsf{asym}}$ and $\mathsf{DA}_{\mathsf{irre}}$ are further data analysis heuristics that will later be described.

Property Cardinality Restriction Heuristic: MinCardinality and MaxCardinality axioms can be proposed on any non-Functional/ non-InverseFunctional edge. We rely on data analysis to confirm the MaxCardinality of the instances of the class on this property.

SA_5: $\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\rangle \wedge \neg\, \mathsf{cons}{:}\langle\langle \lhd, \mathsf{C}, \mathsf{P}\rangle\rangle \rightsquigarrow$
$\mathsf{DA}_{\mathsf{card}}(\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\rangle)$

Class Axioms Heuristic: An IntersectionOf expression can be between classes or between classes and other expressions such as the HasValue expression. Although an IntersectionOf expression that contains a HasValue expression is complicated to search for, nevertheless, it can be found in relational databases by exploiting schema and data analysis. Here, the condition is that the schema should include a hierarchy (at least one SubClassOf) and the action is a data analysis heuristic.

SA_6: $\mathsf{cons}{:}\langle\langle \subseteq, \mathsf{D}, \mathsf{C}\rangle\rangle \rightsquigarrow$
$\mathsf{DA}_{\mathsf{interHasValue}}(\mathsf{node}{:}\langle\langle \mathsf{C}\rangle\rangle, \mathsf{node}{:}\langle\langle \mathsf{D}\rangle\rangle)$

4.2. Data Analysis

Each **Data Analysis** (**DA**) heuristic rule has the form pattern : condition | probability $\rightsquigarrow$ 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).

Transitive Heuristic: DA heuristics such as $\mathsf{DA_{symm}}$, $\mathsf{DA_{asym}}$, $\mathsf{DA_{refl}}$, $\mathsf{DA_{irre}}$ and $\mathsf{DA_{tran}}$ all take the same general form. We illustrate the approach by presenting $\mathsf{DA_{tran}}$. In the rule below, the notation $\mathsf{P/P_1}$ indicates aliasing of P as $\mathsf{P_1}$.

$\mathsf{DA_{tran}}(\mathsf{edge}{:}\langle\langle \mathsf{P,C,D}\rangle\rangle)$: $\mathsf{totalT} \neq \mathsf{notT} \mid \frac{\mathsf{totalT}-\mathsf{notT}}{\mathsf{totalT}} \rightsquigarrow$
$\mathsf{addCons}(\langle\langle \bowtie_{\mathsf{P/P_1.D=P/P_2.C}}, \langle\langle \mathsf{P,C,D}\rangle\rangle, \langle\langle \mathsf{P,C,D}\rangle\rangle\rangle\rangle)$;
$\mathsf{addCons}(\langle\langle \pi_{\langle \mathsf{P/P_1.C,P/P_2.D}\rangle}, \langle\langle \bowtie_{\mathsf{P/P_1.D=P/P_2.C}}, \langle\langle \mathsf{P,C,D}\rangle\rangle, \langle\langle \mathsf{P,C,D}\rangle\rangle\rangle\rangle\rangle\rangle)$;
$\mathsf{addCons}(\langle\langle \subseteq, \langle\langle \pi_{\langle \mathsf{P/P_1.C,P/P_2.D}\rangle}, \langle\langle \bowtie_{\mathsf{P/P_1.D=P/P_2.C}}, \langle\langle \mathsf{P,C,D}\rangle\rangle, \langle\langle \mathsf{P,C,D}\rangle\rangle\rangle\rangle\rangle\rangle, \langle\langle \mathsf{P,C,D}\rangle\rangle\rangle\rangle)$

where totalT is the number of transitive instances of P calculated as:
$\mathsf{totalT} = |\{\langle \mathsf{x,z}\rangle | \langle \mathsf{x,y}\rangle \in \langle\langle \mathsf{P,\ C,\ D}\rangle\rangle; \langle \mathsf{y,z}\rangle \in \langle\langle \mathsf{P,\ C,\ D}\rangle\rangle\}|$
and notT is the number of non-transitive instances of P, calculated as:
$\mathsf{notT} = |\{\{\langle \mathsf{x,z}\rangle | \langle \mathsf{x,y}\rangle \in \langle\langle \mathsf{P,\ C,\ D}\rangle\rangle; \langle \mathsf{y,z}\rangle \in \langle\langle \mathsf{P,\ C,\ D}\rangle\rangle\} - \langle\langle \mathsf{P,\ C,\ D}\rangle\rangle\}|$.

The Transitive heuristic, was found not to apply to the Northwind database.

Min/Max Cardinality Heuristic: The MaxCardinality axiom can be proposed from counting the instances of data involved in properties while a MinCardinality axiom of 1 can be added on any non-Functional / non-InverseFunctional property. The action below pertains to the MaxCardinality axiom. The BAV transformations for adding a MinCardinality axiom is very similar and can be found in Table 2.

$\mathsf{DA_{card}}(\mathsf{edge}{:}\langle\langle \mathsf{P,C,D}\rangle\rangle)$: $\mathsf{N} = \mathit{count}\langle\langle P,C,D\rangle\rangle \mid 1 \rightsquigarrow$
$\mathsf{addNode}(\langle\langle \leq \mathsf{NP}\rangle\rangle)$; $\mathsf{addEdge}(\langle\langle \mathsf{P\_IE},\ \leq\ \mathsf{N\ P,\ D}\rangle\rangle)$; $\mathsf{addCons}(\langle\langle \triangleright^{N},\ \leq\ \mathsf{N\ P,}\ \langle\langle \mathsf{P\_IE}\rangle\rangle\rangle\rangle)$;
$\mathsf{addCons}(\langle\langle \subseteq,\ \mathsf{P\_IE,\ P}\rangle\rangle)$

Suppose that we count the maximum number of territories that an employee can work in is 8, therefore we can propose to add to the HDM schema a MaxCardinality constraint between the node $\langle\langle \mathsf{Emp}\rangle\rangle$ and the edge $\langle\langle \mathsf{Emp\_Ter,\ Emp,\ Ter}\rangle\rangle$. The MaxCardinality construct can then be added to the ontology as in the DL rule: $\mathsf{Emp} \sqsubseteq \leq\ 8\ \mathsf{Emp\_Ter.Ter}$.

IntersectionOf including HasValue Heuristic: The DA heuristic for finding an IntersectionOf that contains a HasValue expression is different than the previous heuristics. In here, the condition is an algorithm (presented in Algorithm 1) that returns a set of axioms.

$\mathsf{DA_{interHasValue}}(\mathsf{node}{:}\langle\langle \mathsf{C}\rangle\rangle, \mathsf{node}{:}\langle\langle \mathsf{D}\rangle\rangle)$: $\mathit{checkIntersectionOfHasValue}(C,D) \neq \emptyset \mid 1 \rightsquigarrow$
$\mathsf{addNode}(\langle\langle \mathsf{VP.a}\rangle\rangle)$; $\mathsf{addEdge}(\langle\langle \mathsf{P\_IE,\ VP.a,\ a}\rangle\rangle)$; $\mathsf{addCons}(\langle\langle \triangleright, \mathsf{VP.a}, \langle\langle \mathsf{P\_IE, VP.a, a}\rangle\rangle\rangle\rangle)$;
$\mathsf{addCons}(\langle\langle \subseteq, \langle\langle \mathsf{P\_IE, VP.a, a}\rangle\rangle, \langle\langle \mathsf{P, C_1, C_2}\rangle\rangle\rangle\rangle)$; $\mathsf{addNode}(\langle\langle \mathsf{NVP.a}\rangle\rangle)$;
$\mathsf{addCons}(\langle\langle \not\ni, \mathsf{VP.a, NVP.a}\rangle\rangle)$; $\mathsf{addCons}(\langle\langle \cup, \mathsf{owl{:}Thing, VP.a, NVP.a}\rangle\rangle)$;
$\mathsf{addNode}(\langle\langle \mathsf{NC}\rangle\rangle)$; $\mathsf{addCons}(\langle\langle \not\ni, \mathsf{C, NC}\rangle\rangle)$; $\mathsf{addCons}(\langle\langle \cup, \mathsf{owl{:}Thing, C, NC}\rangle\rangle)$;
$\mathsf{addNode}(\langle\langle \mathsf{UNVP.a\_NC}\rangle\rangle)$; $\mathsf{addCons}(\langle\langle \cup, \mathsf{UNVP.a\_NC, NVP.a, NC}\rangle\rangle)$;
$\mathsf{addNode}(\langle\langle \mathsf{IC\_VP.a}\rangle\rangle)$; $\mathsf{addCons}(\langle\langle \not\ni, \mathsf{IC\_VP.a, UNVP.a\_NC}\rangle\rangle)$;
$\mathsf{addCons}(\langle\langle \cup, \mathsf{owl{:}Thing, IC\_VP.a, UNVP.a\_NC}\rangle\rangle)$

The algorithm loops all properties in the schema and for each property queries the schema to find the *y* values of members of the parent class (*i.e.*, $\mathsf{node}{:}\langle\langle \mathsf{C}\rangle\rangle$) which participate in the property denoted as *PValue*. Then, it queries the schema again to find the *y* values of members of the child class (*i.e.*, $\mathsf{node}{:}\langle\langle \mathsf{D}\rangle\rangle$) which participate in the same property denoted as *CValue*. If there is only one *CValue* and more than one *PValue*,

Algorithm 1. checkIntersectionOfHasValue

procedure CHECKINTERSECTION
  inputs: *Parent*, *Child*
  $Axioms \leftarrow \emptyset$
  for $P \in SchemaProperties$ **do**
    $PMember \leftarrow \{\langle x,y\rangle \mid \langle x,y\rangle \in edge{:}\langle\langle P\rangle\rangle; \langle x\rangle \in node{:}\langle\langle Parent\rangle\rangle\}$
    $PValue \leftarrow \{y \mid \langle x,y\rangle \in PMember\}$
    $CMember \leftarrow \{\langle x,y\rangle \mid \langle x,y\rangle \in edge{:}\langle\langle P\rangle\rangle; \langle x\rangle \in node{:}\langle\langle Child\rangle\rangle\}$
    $CValue \leftarrow \{y \mid \langle x,y\rangle \in CMember\}$
    if $|CValue| = 1 \wedge |PValue| > 1$ **then**
      $Axioms \leftarrow Axioms \cup (Child \equiv Parent \sqcap P.CValue)$
    end if
  end for
  return *Axioms*
end procedure

the algorithm returns an axiom. For instance, if we envisage that the schema in Figure 1 also includes a table Mgr(MID) with Mgr(MID) $\overset{fk}{\Rightarrow}$ Emp(EID), and that the table has only the data (6,7), then using this heuristic we could detect the following axiom: $\mathsf{Mgr} \equiv \mathsf{Emp} \sqcap \mathsf{RTo}.\{5\}$.

5. Applying Machine Learning

This section explains how the machine learning module has been implemented and integrated with the OWLRel tool. It is structured in such a way that it follows a data pipeline, from the way data is collected and processed to how it is used in forecasting. This encompasses the data collection step, the reconstruction of the HDM graph in an in memory representation which facilitates the feature extraction process, and finally the training of the prediction model.

The general assumption behind our approach is to find patterns that transcend the particularities of individual users, which then can be used to produce a universal score of relevance for any axiom. In order to find such patterns, the users' interaction with the OWLRel tool is recorded in log files. During the execution, in each iteration when axioms are being suggested, they are first processed through the aforementioned pipeline. Then, the models are used to predict whether an axiom will be selected as interesting or not, and this prediction takes the form of a probability that will be displayed to the user through the GUI shown in Figure 5.

5.1. Data Collection

As the tool suggests axioms to its users, their answers are stored in log files. These log files contain information such as the suggested axiom name, whether it was marked as interesting or not, and the iteration number in which it has appeared and been clicked on. Any entry from a log file would therefore have the following schema: [axiom, clickedInteresting, clickedUninteresting, iterationAppeared, iterationClicked], and example for the Northwind database being [irreflexiveproperty:$\langle\langle \mathsf{RTo}, \mathsf{Emp}, \mathsf{Emp}\rangle\rangle$, 1, 0, 0, 1]. The

axiom is suggesting that the property RTo is an IrreflexiveProperty. The axiom has been clicked as interesting by the user in the second iteration, and it was suggested in the first iteration.

5.2. Internal Graph Reconstruction

After collecting the log files, the next step is to extract features that allow interesting axioms to be distinguished from uninteresting ones. As a prerequisite to compute such features, the graph representation of the ontology is reconstructed in memory. The initial state of the graph is given by the original ontology (such as the one presented in Figure 3), and with each interaction the graph is updated, enriching the ontology, and therefore changing the features for subsequent axioms. Accepting a suggested axiom from the GUI will inevitably change the properties of the underlying edges and nodes.

For example, acknowledging that RTo is an IrreflexiveProperty, will update the list of characteristics of that particular edge. At this stage, the Emp class, represented as a node, has two properties: RTo and Emp_Ter, which are in return represented as edges. If a new axiom, for example a PropertyChain, would be suggested and accepted, it would introduce a new property *i.e.*, a new edge in the graph. These structural changes will then change the features derived from the graph. In the case of the PropertyChain, it will among other things, increase the number of connections a node has. As a consequence of working with a graph, features can now be derived separately for any node and edge.

5.3. Feature Engineering

The internal graph representation allows the system to extract features that best describe an axiom. The feature set of an axiom is based on information about the classes and properties that appear in that axiom. The features collected from nodes are: (1) importance, (2) number of connections, (3–5) number of inclusions in UnionOf, IntersectionOf, DisjointWith and (6–7) to how many other nodes it is a super class or sub class of. The importance is governed by how many times the user has selected an axiom that directly affects the node, while the number of connections is the number of edges connecting to the node. With respect to the previous example of the IrreflexiveProperty, the feature vector derived for the Emp class is $Features_{class} = [0,2,0,0,0,0,0]$. The only non-zero value is the number of connections, since the class has two properties. Otherwise, the class is not part of a UnionOf, IntersectionOf, DisjointWith or SubClassOf.

The features extracted for properties are: (1) the number of characteristics it contains (SymmetricProperty, TransitiveProperty, *etc.*), (2–4) the presence of MinCardinality, MaxCardinality and HasValue, (5) number of properties it is an InverseOf, (6) its importance (as defined for nodes) and (7) the iteration it appeared in. In contrast to the nodes, new properties can dynamically appear if a PropertyChain is being chosen from the suggested axiom list. For the same example, the feature vector for the RTo property is $Features_{property} = [1,0,0,0,0,0,0]$. Here, the only non-zero value is the number of characteristics, since the property is a FunctionalProperty.

When a user selects an axiom, it enriches the ontology such that the features for all other axioms referring to nodes or edges in the neighborhood are being affected. With the addition of a new property the number of edges change, and some nodes gain in importance. Therefore, the relevance is being recomputed after each enrichment and all the suggested axioms will have different scores.

As we now have the features for classes and properties, it is possible to derive the complete feature vectors for axioms. Axioms such as cardinality, property characteristics, InverseOf and HasValue always affect properties between two classes. Therefore, they can be represented by the information of the affected nodes and edges. A PropertyChain axiom, on the other hand, affect two sets of nodes and edges. This can be converted to the previously mentioned feature space by taking the average of the edge and node features. UnionOf, IntersectionOf and DisjointWith axioms do not refer to an existing property, and thus cannot be modelled together with the previous axioms. Because of this, two machine learning models have been used: one that focuses on property characteristics, PropertyChain, InverseOf, HasValue and cardinality, and another model that focuses only on the UnionOf, IntersectionOf and DisjointWith axioms. The first machine learning model receives as input the averaged features of all the nodes and properties it affects. The second model acts only on nodes, so it receives solely the averaged features of these. Additionally, each observation receives a categorical variable describing the type of axiom, along with two variables quantifying in which iteration it has appeared and when has it been clicked as interesting. Each observation gathered from the log file will therefore be processed to contain 10 features in the case of UnionOf, IntersectionOf and DisjointWith axioms, while 17 for the other axiom types. For an axiom affecting classes and properties the feature vector would be $Features_{axiom} = [Avg(Features_{class}), Avg(Features_{property}), itApp, itClick, axiomType]$. Finally, the feature vector given to the machine learning algorithms for the RTo IrreflixeProperty has the following value: $[0,2,0,0,0,0,0,1,0,0,0,0,0,0,0,1,\text{Irreflexive}]$.

5.4. Training and Validation

The nature of the collected data allows for supervised learning methods to be deployed, as the 'correct' label for each suggested axiom has been recorded. The machine learning models taken into consideration for this task were the Naive Bayes classifier and the Logistic Regression due to their simplicity, and the fact that they output probabilities rather than just class labels. The Logistic Regression model is defined as in equation (29) where the linear model described by the coefficients beta is applied onto the input variables x_i (*e.g.*, the feature vector) after which it is being squashed by the logistic function to output a probability.

$$y = \frac{1}{1 + e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p}} \tag{29}$$

Data has been collected from three users enriching the Wine Ontology[4] with axioms proposed from the schema enhancement phase described in Section 4. The total amount of axioms interacted with, amounts to roughly 300. Using the Weka toolkit[5], the machine learning algorithms were run with the default model and regularization parameters provided. The performance of the models was assessed using a standard 10-Fold Cross Validation leading to the conclusion that the Logistic Regression is the better choice, since it obtained an average accuracy of 86%. Besides of the standard, numerical evaluation method, the applicability of the models has been assessed in practice by manual

[4] http://www.w3.org/TR/owl-guide/wine.rdf
[5] http://www.cs.waikato.ac.nz/ml/weka/

inspection, showing no significant discrepancy between the predicted relevance and the users' actual preference.

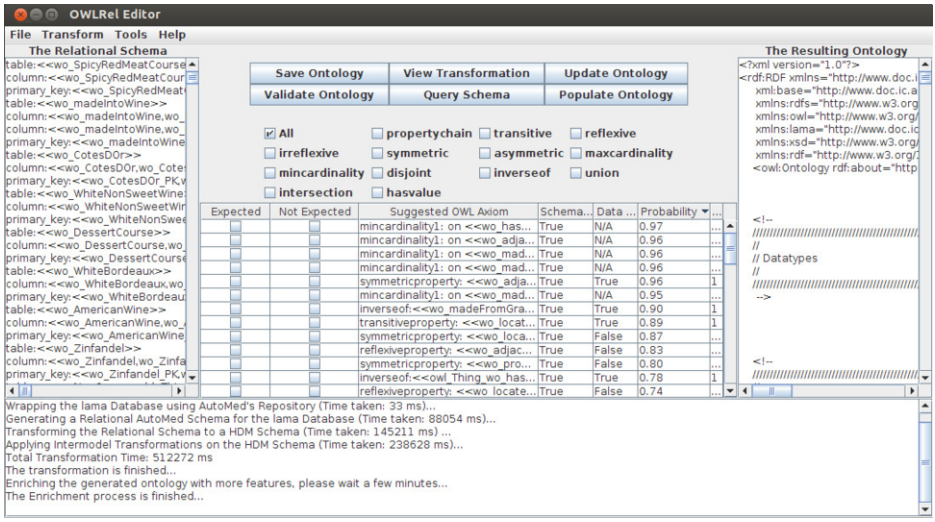

Figure 5. Ontology Enrichment using OWLRel Editor

6. Related Work

Most of the approaches that extract a domain-specific ontology from a relational database, such as: [13], [14] and [5], suffer from one or more of the following problems [15]: 1) Do not map highly expressive OWL features. 2) Miss interpret primary keys. 3) Do not properly identify class hierarchies.

The survey [15] 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 and 3. Problem 3 was addressed by the work of [14] and [16] 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 FunctionalProperty and Cardinality 1. Only a few, like us, have considered using the HasKey construct such as [13] and [17]. Furthermore, eliciting the highly expressive constructs of OWL, such as HasValue, TransitiveProperty and SymmetricProperty, was only considered by [6]. However, their rules for generating these constructs are ambiguous and do not rely on the analysis of the data. 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 relational database and represented primary keys using the HasKey construct, thus providing solutions for problems 1 and 2 above. Moreover, our approach applies machine learning to the extraction of OWL ontologies from relational databases, something that was not considered by any of the existing state-of-the-art approaches.

7. Conclusions and Future Work

This paper has presented an approach that extracts OWL ontologies from relational databases using a combination of data analysis and machine learning. The data analysis technique used helped in producing rich ontologies that contain more semantic information than found in the relational databases. Applying machine learning algorithms to the extraction of ontologies helped in ranking the large number of suggestions produced by the data analysis technique. Testing our approach showed that it is feasible and promising. Future work will be directed towards enhancing our heuristics and the machine learning algorithms used.

References

[1] X. D. M. Li and S. Wang, "Learning Ontology from Relational Database," in *In proc. of Inter. Conf. on Machine Learning*, vol. 6, pp. 3410–3415, 2005.

[2] L. Lin, Z. Xu and Y. Ding, "OWL Ontology Extraction from Relational Databases via Database Reverse Engineering," *Journal of Software*, vol. 8, no. 11, pp. 2749–2760, 2013.

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

[4] W3C, "OWL 2 Web Ontology Language New Features and Rationale," June 2009. http://www.w3.org/TR/2009/WD-owl2-new-features-20090611/.

[5] S. H. Tirmizi, J. Sequeda, and D. Miranker, "Translating SQL Applications to the Semantic Web," in *DEXA*, pp. 450–464, Springer, 2008.

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

[7] L. Al Khuzayem and P. McBrien, "OWLRel: Learning Rich Ontologies from Relational Databases," *Baltic Journal of Modern Computing*, vol. 4, no. 3, p. 466, 2016.

[8] A. Poulovassilis and P. McBrien, "A General Formal Framework for Schema Transformation," *DKE*, vol. 28, no. 1, pp. 47–71, 1998.

[9] M. Boyd and P. McBrien, "Comparing and Transforming between Data Models via an Intermediate Hypergraph Data Model," *DS*, vol. IV, pp. 69–109, 2005.

[10] A. C. Smith and P. McBrien, "Inter Model Data Exchange of Type Information via a Common Type Hierarchy," in *DISWEB*, 2006.

[11] P. McBrien and A. Poulovassilis, "Data Integration by Bi-Directional Schema Transformation Rules," in *Proc. of ICDE*, pp. 227–238, IEEE, 2003.

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

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

[14] I. Astrova, "Reverse Engineering of Relational Databases to Ontologies," in *The Semantic Web: Research and Applications*, pp. 327–341, Springer, 2004.

[15] D.-E. Spanos, P. Stavrou, and N. Mitrou, "Bringing Relational Databases into the Semantic Web: A Survey," *Semantic Web*, vol. 3, no. 2, pp. 169–209, 2012.

[16] F. Cerbah, "Mining the Content of Relational Databases to Learn Ontologies with Deeper Taxonomies," in *Inter. Conf. on WI-IAT.*, vol. 1, pp. 553–557, IEEE, 2008.

[17] E. Vysniauskas, L. Nemuraite, R. Butleris, and B. Paradauskas, "Reversible Lossless Transformation From OWL 2 Ontologies into Relational Databases," *IJITCA*, vol. 40, no. 4, pp. 293–306, 2011.