Exploring Logical and Hierarchical Information to map Relational Databases into Ontologies

Hegler Tissot

Federal University of Parana, CS Department, C3SL Labs, Curitiba, Brazil.
E-mail: hegler@gmail.com

Cristiane Aparecida Gonçalves Huve

Federal University of Parana, CS Department, C3SL Labs, Curitiba, Brazil.
E-mail: caghuve@inf.ufpr.br

Leticia Mara Peres

Federal University of Parana, CS Department, C3SL Labs, Curitiba, Brazil
E-mail: lmperes@inf.ufpr.br

Marcos Didonet Del Fabro

Federal University of Parana, CS Department, C3SL Labs, Curitiba, Brazil.
E-mail: marcos.ddf@inf.ufpr.br

Abstract Ontologies are formal specifications of conceptualizations. Their designs require to understand the concepts involved in the domain to be mapped. One well-known method to produce ontologies is to extract their concepts from relational databases. We conducted a practical study over a real-world scenario on applying existing rules and we identified open issues to be addressed, such as the utilization of logical metadata as a proper vocabulary, the implementation targeted to specific domains and mappings of hierarchical and self-hierarchical structures. In this paper, we present a novel approach that overcomes these issues. Our solution uses physical and logical models to enrich the terminology produced in the target ontology. It also contains a more comprehensive set of rules, taking into account instances and (self)hierarchies. We validate our approach with 2 experiments from the healthcare domain as input.

Keywords: Ontologies, rule based integration, logical model, hierarchical structures

Biographical notes: Hegler Tissot is a Senior Research Fellow in Health Informatics. He received his Ph.D. degree in Computer Science from the Universidade Federal do Paraná (UFPR) in 2016, as a member of the C3SL Labs, and was a postdoctoral researcher at Macquarie University, in 2017. He works with knowledge engineering and his main research interest is about improving public health services by applying natural language processing to extract structured information from electronic health records and using machine learning to design prediction models within the clinical domain.

Cristiane Aparecida Gonçalves Huve is a PhD student at Federal University of Parana. She has been working on the development and research of complete ontologies from relational databases with logical information.

Leticia Mara Peres is an Adjunct Professor at Federal University of Paraná, Curitiba, Brazil. She has a Ph.D. degree in Computer Science from Federal University of Paraná, 2010, and carried out her post-doctoral studies at the Department of Services and Information Systems Engineering at the Polytechnic University of Catalonia, 2016. Her main research interests are on applications and fundamentals of systems and software engineering, with emphasis on verification and validation of systems and software.

Marcos Didonet Del Fabro received his Ph.D. degree in computer science from the University of Nantes, in 2007. He is an Associate Professor with the Universidade Federal do Paraná, Brazil. He was a Researcher with IBM Software Group, France, on the integration of business rules and model-driven engineering. He was a Post-Doctoral Fellow with ILOG. His current research is about information extraction of imprecise data, model-driven engineering and data integration.

1 Introduction

Ontologies are explicit and formal specifications of a conceptualization. An ontology depicts a certain problem domain, and provides an abstract model, comprising unambiguous and consensual definition of concepts and relationships among them, which can be understood and processed by computers [1]. Ontologies are increasingly being used, and considerably growing in number, complexity and domains they model [2], playing an important role in resolving semantic heterogeneity. However, building ontologies from scratch or manually is a complex, mistakable and time consuming process [3]. Moreover, the domain experts need to understand the syntax and semantics of ontology development languages, often leading to poor definitions [4].

To overcome this lack, the generation of ontologies from relational databases (RDB) has been widely studied as a viable approach for ontology construction [5]. Relational databases can be used as the source of information, leading to the creation of systems for instantiating ontologies. The process of building ontologies should be preferably (semi)automated for a knowledge acquisition process that does not have the support of domain experts [6, 5]. The RDB structure is intended to be used in different ways in order to construct ontologies, such as simply building an ontology, integrating multiple databases, ontology learning, and managing the organization knowledge [7]. Moreover, ontologies can be used to unify the description and retrieval of data in relational databases, giving data integration that includes: a) to recognize data inconsistency and redundancy, b) to support shared common data understanding and mapping, c) to provide a vocabulary of terms which is independent of relational database schemata, and d) to express the equivalence of terms used in relational databases [8].

The works from [9, 10, 11] provided a classification covering aspects such as the existence of an initial ontology, its application in reverse engineering, automation level, and others. Notably, the approaches [12, 13, 3, 14, 15, 16, 6, 17, 18, 19, 5, 20] aim to convert relational data models to ontologies, by providing mapping rules that translate the elements of a database schema into concepts and relationships of an ontology. The W3C RDB2RDF Working Group provides as well a survey [21] of different approaches of mapping relational databases to ontologies, including the definition of a reference architecture to compare them. It focuses on the aspects to be addressed to move towards standardization, not detailing the kinds of mappings.

We conducted a practical study over a scenario from the medical domain on applying existing rules, aiming on the increase of the legibility of the generated ontology. We identified that although these approaches are designed with a significant number of rules, they perform mappings at the physical level and there are gaps to be addressed. Important aspects that could enrich the generated ontology are not handled, such as: a) using relationships and tuples in order to identify class (self)hierarchies, b) considering "Boolean" attributes in order to define positive or negative property assertions, and c) using logical metadata as a proper vocabulary in order to produce a target ontology that provides the common understanding of a knowledge domain.

In this paper, we present a complete relational-to-ontology mapping architecture and tool to overcome the mentioned issues. Our solution has three main contributions.

Generic architecture: it covers the full chain of definitions and operations. We notably define metadata scripts and mappings that store complete information about the mapping process, which is technology independent.

Utilization of the logical model: the architecture uses the physical and logical models to enrich the terminology produced in the target ontology, making the ontology easier to understand. Both models are also stored in the metadata mappings, being as well not dependent of any RDB or ontology format.

Mapping rules: they cover a large set of translation cases. Some part of the rules are based on existing works, while the new subset handles problems identified on our study. We provide improvements in the naming of ontology elements and instances, mapping NxN relationships, handling database constraints and producing self-hierarchical relationships. The output ontology is generated in the OWL [1] format.

We validate our approach through 2 experiments using input data from the healthcare domain. The first one focuses on the automatization of the execution of the rules. It is backed up by a complete tool available for download [2] . The second one uses a complex relational data model where all the designed rules are applied and we compare with a significant existing approach.

This paper is organized as follows: Section 2 refers to the related work, providing an extensive classification of the existing kind of rules. Section 3 presents the motivations for our work using a real-world scenario. It makes clear the necessity of improving existing solutions. Section 4 describes the RPL_2O mapping process architecture and its set of mapping rules. Section 5 depicts the experiments performed in order to validate our approach and to compare it with related work. Finally, Section 6 concludes with considerations and future work.

2 Related Work

Several approaches aim to convert relational data models into ontologies. Most part of these solutions develop set of mappings using a RDB physical model or variations/extensions of it. Michel [11] Spanos [9] and Sequeda [10] surveyed the motivations and benefits of a mapping process, analyzing related work and expose different research topics of relational databases in the context of the Semantic Web, the challenges, and their different

1 http://www.w3.org/OWL/
2 https://github.com/caghuve/rdb-to-onto

application purposes. In short, these approaches and developed tools are classified with respect to the following aspects: a) existence of ontology, b) ontology domain, c) application of database reverse engineering, d) automation level, e) data accessibility, e) mapping language, f) ontology language, g) vocabulary reuse, h) complexity of mapping definition, i)software availability, j) graphical user interface, and l) purpose.

Concerning the creation of a new ontology from scratch from a relational database, in which our approach will fit in, [9] classified the approaches in two groups: a) not performing reverse engineering, and b) performing reverse engineering, through a set of forward and/or backward mappings. In the first group, approaches such as [22, 23] use the R2RML language [23, 24, 25]. Although it is an important initiative, this language does not support to record the decisions made during the mapping process and the relationships between concepts of the relational-to-ontology mapping. For this reason, it is difficult to provide the inverse translation. Most part of the surveyed approaches support reverse engineering, even if some of them does not have a working forward/backward implementation. X3ML [26] is a recent approach that enables creating mappings using a specific mapping language. The framework is complete and other solutions could be implemented on top of it. However, it does not focus in the kinds of rules that could be implemented.

Jain and Singh [27] compare well-known frameworks used to convert relational databases to ontologies. The authors highlight the most common issues in existing tools, and they propose to extend an existing framework with new features to overcome such shortcomings. However, it provides a 1:1 mapping between the input and output elements (table x class, column x property), limiting the modeling possibilities.

Telnarova [12] produces ontologies from data extracted from RDBs, specifying rules to create classes, properties, hierarchies, cardinalities and individuals. Gherabi et al. [13] developed a tool that searches for particular cases of database tables to determine which ontology component has to be created from which database component. OGSRD [3] is a method for ontology building from relational databases that uses construction rules of ontology elements (concepts, properties, axioms, individuals), but it ignores sets of tables that express association data. Ren et al. [15] propose a rule-based transformation solution, which considers both schema transformation (relations, attributes, and constraints) and data extraction. DB2OWL [16] maps tables to classes and columns to predicates, using relational database schema characteristics' to assert sub-classes and object properties. Louhdi et al. [6] propose a set of transformation rules which analyzes stored data to detect disjointness and totalness constraints in hierarchies and it calculates the participation level of tables in n-ary relations. Ramathilagam & Valarmathi [4] propose an RDB to Ontology mapping framework that can generate an Ontology based on the proposed mapping rules for a banking domain. This framework also includes additional semantic rules to improve the expressiveness for the generated ontology regarding to the search and retrieval processes. QUALEG DB [14] is a mapping implementation that handles issues often found in existing approaches, such as: a) discovery of inheritance; b) discovery of restrictions, symmetric and transitive properties; c) consideration of constraints that capture additional semantics; d) analysis the semantic loss caused by the translation, due to the semantic mismatch between both data models. SOAM [17] is a semi-automatic ontology acquisition method based on data in RDB, which aims to acquire an OWL ontology without demanding an intermediate model and to semi-automatically refine the obtained ontology according to existing lexical knowledge repositories. Finally, a set of solutions focus on mapping the database tuples to ontology individuals using different methods, such as: a) based on a defined template (RDB2ONTO [18]); or b) based on datasets extracted from the source RDB (RDB2OWL [19] and RDOTE [5]). Similarly, D2RQ [3] offers a powerful declarative language for mapping RDB tuples to individuals of an ontology. [28] provides a set of extensive mappings covering nearly all the elements of an RDB, translating schema and instances. However, they do not focus on semantic relations handling hierarchical mapping. [29] also provide a complete mapping from RDB elements, though they use an intermediate model that it is further translated into an ontology. While it provides a generic architecture, it is necessary to manually enrich the ontology with additional information and to know another representation format.

Solutions addressing hierarchical mapping were also studied. Sequeda et al. [10] provides an overview of automatic translation of an inheritance hierarchy in an ontology. Reviewed approaches express inheritance modeling through possible foreign key patterns, wherein: a)foreign key is also the primary key, b)foreign key and primary key are disjoint, and c)foreign key is a subset of the primary key. Finally, the approach from Cerbah [30] considers additional information in the translation process. It uses a distance-based feature to select a relevant dataset based on the similarity between the feature value and a subset of the input text. The main objective of this work is to integrate to 'RDBToOnto' platforms a learning component to perform categorization patterns from which class hierarchies can be generated.

3 Motivation

We motivate our approach after attempting to apply the mapping rules described in the previous section to create an ontology using OWL 2.0 [4] , for the *InfoSaude* system. *InfoSaude* [31, 32] is a medical information system used

3 http://sourceforge.net/projects/d2rq-map/
4 https://www.w3.org/TR/owl2-overview/

Table 1 Common mapping rules to map relational database elements into ontology components.

	Elements	Rule	Description
1.	Tables to Classes	A	Each non associative table is mapped to an ontology class.
		B	Information from different tables is mapped to one single ontology class.
2.	Associative Tables	C	Many-to-many relationships are mapped to two object properties.
		D	Many-to-many relationships are mapped to two mutually inverse object properties.
		E	Domain and range are defined for object properties that represent many-to-many relationships.
		F	Each table that references more than two tables is mapped to a class and an object property for each foreign key.
3.	Foreign Keys to Object Properties	G	Each foreign key in non associative tables is mapped to an object property.
		H	Domain and range are defined for object properties that represent FKs.
		I	FKs are mapped as two mutually inverse object properties.
		J	Define "has-part" and 'is-part-of" object properties for each FK that is part of the PH in non-associative tables.
4.	Tables to Subclasses	K	Each foreign key that is equivalent to the primary key is mapped as a subclass.
		L	Each foreign key that is part of the primary key is mapped as a subclass.
5.	Columns to Data Properties	M	Each column that is not part of a foreign key is mapped to a data property.
6.	Not Null Constraints	N	Not null columns are mapped to data properties with *min* cardinality restriction equals to 1.
7.	Primary Keys to Data Properties	O	Primary key columns are mapped to data properties with *min* and *max* cardinality equals to 1.
		P	Primary key columns are mapped to inverse functional properties with *min* cardinality equals to 1.
8.	Unique Constraints to Data Properties	Q	Unique key columns are mapped to data properties with *max* cardinality equals to 1.
		R	Unique key columns are mapped to inverse functional properties.
9.	Table Data to Individuals	S	Tuples is mapped to ontology individuals of specific classes.

to meet the needs of 75 public health centers in the city of Florianópolis, Brazil. It integrates various information systems used by the Brazilian Ministry of Health, such as the Outpatient Information System (CIS) and the International Classification of Diseases (ICD) tables. The system is also used to report information about Ambulatory Care Individual (RAAI), summarizing data on the type of care, pregnancies, procedures performed on the patient, applied vaccines and drug prescriptions. This schema was chosen for different reasons: first, we could apply our solution to a real world scenario; second, it is complex enough since it comprises several existing rules; finally, it enabled us to discover new issues that are not covered by existing solutions, which are described in this section. Still, other schemas with similar constructs from different applications could also be used.

Figure 1 illustrates an excerpt of the physical and logical models [5] . This excerpt stores information about medical procedures, its related disease information and specific groups of procedures. As it can be seen, the logical model is easier to understand, since it has a more friendly terminology.

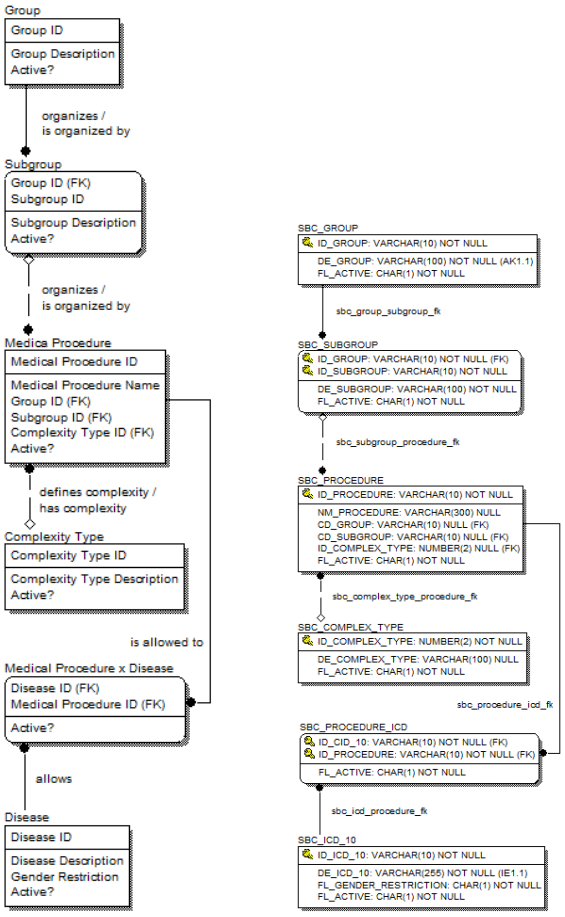
This RDB data model was used to motivate our approach and the driving example to explain the rules definitions, as long as the set of rules from the related work cannot to handle some RDB structures, or could handle only part of them, that could enrich the target ontology. These elements are presented in this section. We describe below the main issues found while trying to produce a RDB-to-Ontology mapping for the *InfoSaude* system.

Naming classes and properties

The analyzed solutions keep the original names, which makes the ontology difficult to be understood when there is some type of encoding in naming RDB elements. Although acronyms and abbreviations are widely used to create RDB elements, it is desirable to have more meaningful names for the ontology elements. Furthermore, when using FK names to name object properties, the ontology may not be able to express concepts with similar relations (in terms of meaning) modeled between different entities. This can lead to generate ontologies where each source FK relation is a conceptually different one.

RDB logical metadata provides conceptual information of a specific problem domain expressed independently of a particular database management. The vocabulary used in logical data models are more closely related to the real world names than the terminology used to create the physical data model.

5 The original model is in Portuguese. It has been translated to English for illustration.



(a) Logical model (b) Physical model

Figure 1 *InfoSaude* data model excerpt: Medical Procedures.

Additionally, there may also exists conflicts when setting data property names for columns that have the same physical (e.g. `active_flag`) or logical names (e.g. *"Full Description"*) in different RDB entities and tables. One single data property could be created to refer to all database columns that share the same semantic definition, rather than creating one single data property corresponding to each database column.

Primary and unique keys

Primary and unique keys are handled by setting *min* and *max* cardinalities and by defining such keys as inverse functional properties, which are not defined in existing specifications. In addition, creating inverse functional object properties to simulate uniqueness would be semantically inappropriate.

Moreover, it is not possible to define a set of columns as inverse functional properties for unique keys composed of two or more columns (composite unique constraints), otherwise we would have two different unique keys. Composite keys should be mapped to a single data property in order to keep the RDB key consistence within the ontology individuals.

Check constraints and ranged data types

Although in [14] check constraints with specific enumerated values are mapped to data range definitions, no specific class is created to map valid values as individuals. Columns associated to a specific check validation based on a range of values can be mapped as object properties instead of data properties. Instead of just defining a range of possible values for a data property, the validation set could be easily managed by mapping such columns as object properties with *range*, which is defined as a specific class to represent the check domain and the corresponding individuals that define the possible values.

Naming individuals

The analyzed solutions do not describe how to name the ontology individuals, except for [15], which translates primary key values into URIs' (Universal Resource Identifiers) postfix of the individual names. When a column that describes a table tuple is not used as part of the instance name, it becomes necessary to analyze the instance property values to recognize such instance, instead of identifying it only by its name.

None of the solutions define criteria to select which input data records should be mapped into ontological instances. Although the entire RDB metadata can be used in order to create a target ontology representing and describing the domain, record data can be partially mapped to ontology individuals depending on the kind of table they are associated with. According to [15], RDB data is divided into two parts: a) master data, representing the core objects in business behaviors, such as customers and products, as well as association data used to explain the relations between them; and b) transactional data, representing the historical occurrences of events in business behaviors that describe business actions. Transactional data can be disregarded when mapping a RDB to an ontology, however, unless the RDB logical model stores some information about each entity type, it can be a difficult task to identify whether a given table stores master or transactional data.

Hierarchies

Some of the analyzed solutions handles inheritance hierarchies from SQL schemas. They create a subclass hierarchy based on FKs and PKs [12, 14, 6, 15, 3, 16, 17] specifications. However, none of them support the creation of a class hierarchy based on record data extracted from unique columns that have self-hierarchical structures. This would help to adjust database models that have hierarchical models only modeled as instances.

In the *InfoSaude* model, the table `SBC_ICD_10` (International Classification of Diseases provided by the World Health Organization [6]) is an example in which the PK column `ID_ICD_10` self-connects tuples without a FK definition (e.g. a tuple with `ID_ICD_10` = "S08" represents

6 http://www.who.int/classifications/icd/

a group of diseases that comprises tuples with column `ID_ICD_10` in {"S081","S082","S088","S089"}).

Many-to-many relationships

There are tables with many-to-many relationships but that do not have any other attribute beyond those used as FKs. We aim to extend these mapping rules to cover those associative tables that have up to one Boolean attribute used to identify whether or not the relation is active.

Boolean attributes can be mapped in different ways in a RDB schema (e.g. "Boolean" with values "True" or "False", "Numeric" with values 0 or 1, "Char" with values "Y" or "N"). Unless the RDB logical model stores any semantic information about each attribute, it can be difficult to identify whether or not an additional column found in an associative table cover such semantic use.

Although table *SBC_PROCEDURE_ICD* in Figure 1 is an associative table, it should not be consider as such because it has an additional column (*FL_ACTIVE*). We drive our approach to consider that if this additional column is defined as a "Boolean" datatype we can still map such table into two ontological object properties, using the columns to define object property assertions.

These additional mapping rules are not taken into account in the analyzed approaches and they are important in order to improve the conceptual description provided by the target ontology. In this paper we present a set of rules that overcomes the issues described above.

A mapping process considering only the use of RDB physical model accounts for a plain syntactical translation of relational databases to ontologies. Combining RDB physical and logical models is our main motivation to implement a new architecture, in a way to support more expressive mapping, differentiating from others proposals for presenting new rules of identifying hierarchies and by worrying about the legibility of the generated ontology.

4 RPL$_2$O Architecture

In this section we present our architecture to perform relational-to-ontology mappings (RPL$_2$O). First, we describe the main components and their flexible pipeline, taking into account the logical and physical data models as input. Second, we propose a set of mapping rules, which are a composition of already known rules from the literature and also new rules handling the issues presented in the motivation section.

4.1 Context

This section present the definitions of ontology models and database schemas, which are used in our approach.

4.1.1 Relational databases schemas

Modeling is part of database design which defines a capable structure for data manipulation. Database modeling is divided into three levels: conceptual, logical and physical. At a high-level, data requirements are mapped into a conceptual data model, sufficiently detailed to describe its scope [33]. Next level matches the logical specification of a data model, which groups the information into structures (e.g. entities, attributes) and describes how the information will be structured (e.g. relationships, integrity rules). Last, physical data model corresponds to the internal organization of data storage. [33]. In our approach, we focus on the logical and physical models.

The database definitions are descriptive information stored by a DBMS in a catalog or dictionary [34]. The data specification from a database is called relational (or data) schema. A database schema reflects the design and the data specification of the database.

A relational schema R, denoted by $R(A_1, A_2, ..., A_x)$, is composed by a relation name R and a list of attributes $A_1, A_2, ..., A_x$. Each attribute A_i, also named a column, is a name of a role played by some domain D in the relational schema R. D is called domain of A_i and is denoted by $dom(A_i)$. A relation state r of a relational schema R, also denoted by $r(R)$, is a set of n-tuples $r = \{t_1, t_2, ..., t_n\}$ (tuples are also referred to as records). Each n-tuple t is an ordered list of m values, $t = (v_1, v_2, ..., v_m)$, where each value v_p, $1 \leq p \leq n$, is an element of $dom(A_i)$ or is a special NULL value [34].

The possible tuple values can be categorized differently. A key value (K) allows identifying stored data referring it by a unique identifier [35]. A unique key (UK) is composed of one or more columns, and it forces a unique information in each table tuple. A primary key (PK) is composed of one or more columns which, taken collectively, allows us to identify uniquely a table tuple of the database [33]. In order to relate two or more tables, they must have a common relationship. A foreign key (FK) denotes a relationship of its corresponding tuple with a primary (PK) of another tuple. Finally, integrity constraints, or simply constraints, are conditions used to prevent the entry of incorrect information [34]. An integrity constraint involving two relations is called foreign key (FK) constraint. The constraints may be written over any kind of tuple value. A database element can be a table, a column, a tuple, a value, a constraint, or a key.

4.1.2 Ontologies

Ontology comes from two Greek words *ontos* (being) + *logos* (science, study) [36]. The use of the term ontology in computer science is related to building knowledge bases using automatic computational reasoning, with interoperable structures that describe concepts and relations among them [37, 38].

Ontologies standardize meanings through semantic identifiers, which can represent the real and conceptual

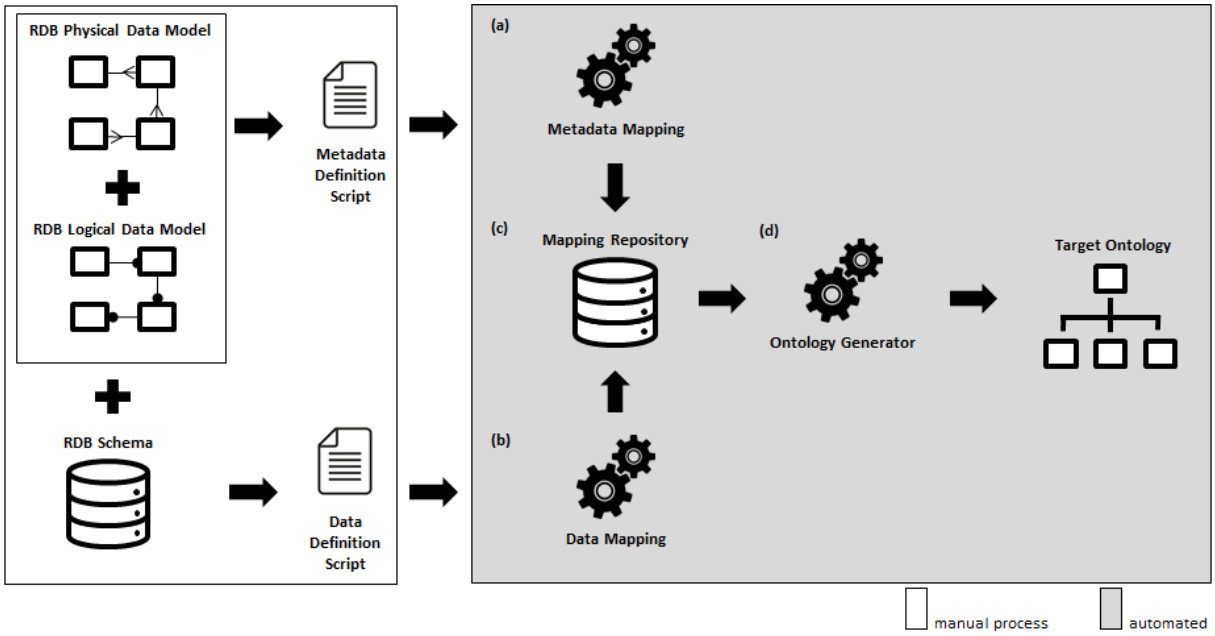


Figure 2 RPL$_2$O Architecture and Components.

world [37, 39], using a specific and shared vocabulary to describe a domain, capturing concepts and relations, and axioms to restrict its interpretation [40].

An ontology O can be represented is denoted by five kinds of elements $O = <C, P, I, V, A>$, denoting, respectively, the sets of classes $C = \{c_1, c_2, ..., c_i\}$, properties $P = \{p_1, p_2, ..., c_j\}$, instances $I = \{i_1, i_2, ..., i_k\}$, property values $V = \{v_1, v_2, ..., v_l\}$ and axioms $A = \{a_1, a_2, ..., a_m\}$.

Informally, classes are used to represent a group of elements, which have similar characteristics and form a capable concept to represent an object belong to a domain. Properties describe characteristics and interaction types among concepts of a domain. The interactions (or associations) are denoted by binary properties. Axioms represent conditions (or constraints) which must be strictly adhered. Instances represent individuals which share class properties, and property values is a set of instances for each of these properties [41].

In our work, we use OWL (Web Ontology Language) ontologies [42, 43] as basis for representation.

4.2 RPL$_2$O Components

The RPL$_2$O process developed in our architecture has three main steps (see Figure 2), which follows the pipeline described below. The approach is semi-automatic. The metadata are partially extracted from the database models. Additional information is manually given, as explained below. After the definition files are set, the execution of the mappings are fully automatic.

(Meta)data Definition Scripts: the *metadata definition script* extracts and insert into a single CSV (Comma-Separated-Value) metadata definition file the RDB physical and logical data models, so it can be further read in a single step. The (Meta)data Definition Scripts follow the same template. The CSV is composed of 18 columns. The first one receives an identification of the *database element type*. It has three possible values: T(for tables); C(for columns); and R (for tuples). The second CSV column receives the *element physical name* and the logical database name is filled in the third column. The physical and logical names of a database element are related to the tables or columns names. The next column receives the table characteristics being identified by associative and non-associative table. This need to be manually filled. The next CSV column stores the columns constraints and additional information that can be used to detail the database business rules. The last two columns can receive record values of database tuples. The produced file is independent of the underlying RDB technology, it can be created by importing the data from the database schema, not being mandatory to inform the RDB data or the Logical Data Model. The files generated from the metadata and data definition scripts are first extracted then completed manually. The physical metadata is acquired from the database schema. The input logical may be available in specific RDB tuples or in specific files according to a given modeling format. The metadata definition file is filled with the physical metadata, logical metadata plus additional information that can be used to drive the process, such as the range and the domain of a check constraint. In Figure 2, physical and logical metadata are merged by the *Metadata Definition Script* (illustrated in the top-left rectangle). All filled metadata can be automatic read in the *Metadata Mapping*(a). Similarly, the *Data Definition Script* is filled with extracted data from the RDB Schema into a specific data definition file. This file is used when a subset of database instances are used in the *Data Mapping*(b), such as hierarchical information.

(Meta)data mappings: the *Metadata Mapping*(a) and the *Data Mapping*(b) processes use the input scripts filled with RDB *Metadata* and *Data*. The *Metadata Mapping* component uses logical and physical metadata obtained from the corresponding RDB data models to map the RDB components to the ontology elements. When the logical name is not inserted, the mapping occurs considering the physical name of the database elements. The *Data Mapping* component uses data gathered from the RDB records to create ontology individuals or subclasses. This mapping allows to classify the future ontology elements with elements that already exist. It also enables to create new elements hierarchically. Once the files are created, the *Metadata Mapping* (a) and the *Data Mapping* (b) processes are conducted automatically from the input scripts, where it applies a set of mapping rules, detailed in section 4.3. The results of the mappings are automatically generated and stored in the *Mapping Repository* (c). The mappings stored in the mapping repository are initially used for forward transformations, but the same architectural model could be used to be extended and to implement inverse mappings. In our solution, we have implemented only the forward transformation. The term "mapping" is often used to refer the connection between database elements and ontology elements. For this, we designed a entity–relationship model wherein the created entities to represent database and ontology elements and its correspondences were represented by an entity relationship model. In RPL_2O architecture, this model is a metamodel which we elaborated to create a schema and maintain the relations between database elements and ontology elements. The schema is represented by *Mapping Repository* (c) component. During *(Meta)data mappings* the database elements (from scripts) are stored as a record into tables defined for the database context. Once the database elements are stored and the *RPL_2O Mapping Rules* (4.3) are applied, the ontology elements are produced and inserted into tables defined for ontology context. The relations between tables from database context and ontology context in this metamodel enables source-to-target traceability. Each record is associated to a database mapping process, being possible to keep stored the mapping of distinguished databases without processing the entire mapping routine when is necessary only perform the *Ontology generation.* The *(Meta)data mappings* perform different types of mappings, which occur according to the database elements inputs, being the mapping processed in the following sequence: database tables, database columns and database tuples.

Ontology generation: the *Ontology Generator*(d) process uses the data previously stored in the *Mapping Repository*(c) to create the target ontology, following a series of naming and hierarchical organization rules in order to avoid duplicated names within the ontology elements. As a result of RPL_2O architecture semi-automated process, an OWL file is generated with the results of this mapping process, keeping the mapping relations stored in the *Mapping Repository* (c). RDB definitions that cannot be directly captured or inferred from the RDB data and metadata produce inconsistent ontologies. Such inconsistencies can be used as feedback and new definitions can be directly added to the repository. Once the *Mapping Repository* is changed, a new version of the target ontology can be generated.

4.3 RPL_2O Mapping Rules

In this section we present an extended set of mapping rules between RDB elements and ontology elements. The rules are organized in two groups: *RDB Mapping Rules* and *Ontology Validation Rules* For each rule, we provide its definition, detailing the RDB data/metadata required, depicting specific details of the process and an illustrative example extracted from the generated ontology. As already stated, Figure 1 shows an excerpt of the *InfoSaude* system data model used in the examples (for managing a list of medical procedures), which will be used as driving example.

4.3.1 RDB Mapping Rules

These rules focus in the RDB elements and handle how to map each RDB component to the corresponding ontology elements (more than one rule can be defined to map each RDB component). The approach is intended for generating an ontology from scratch after giving an input database and its models (logical and physical), at least after its first execution. This means it won't be necessary to create more specialized mappings with intermediate nodes. Still, the approach handles N-N mappings from the input database and translate them, corresponding to the associative tables mapping. We do not handle event-centric approaches, though we think it could be adapted if added consistency maintenance. The mapping rules handle and extend the five main kinds of mappings identified in the related work: 1) RDB entities/tables and ontology classes, 2) RDB attributes/columns and ontology data properties, 3) RDB relationships/foreign key constraints and ontology object properties; 4) RDB tuples and ontology instances and 5) hierarchical relations between classes, including check constraints and self-hierarchies.

Database Tables

A database table represent a collection of data for a given domain concept.

Rule 1. *Non-associative Tables to Classes.*

Non-associative tables store information to be used as reference to other tables. Each non-associative table is mapped to an ontology class; the class name will be assigned during the *Ontology Generator* process by using the logical entity name.

This rule is equivalent to Rule A in Table 1. See Section 4.3.2 for further details about naming ontology elements and how we suggest guaranteeing unique element names within the entire ontology.

Example: tables SBC_PROCEDURE, SBC_COMPLEX_TYPE and SBC_ICD_10 are non-associative tables, mapped to the corresponding OWL classes, as shown below:

```
<Declaration>
    <Class IRI="#MedicalProcedure"/>
</Declaration>

<Declaration>
    <Class IRI="#ComplexityType"/>
</Declaration>

<Declaration>
    <Class IRI="#Disease"/>
</Declaration>
```

Ontology Properties

Properties in are binary associations among individuals that represent general facts about class members and specific facts about individuals [42, 1]. Properties are characterized by their domain, range or algebraic characteristics [44]. More than one domain or range may be declared to restrict the relation. There are two kinds of properties: *owl:ObjectProperty*, which relates classes and individuals and *owl:DatatypeProperty*, which relates a class to a data type value.

Rule 2. *Associative Tables to Object Properties*

Associative tables store relationships between two or more tables. We identify associative tables as those that have two FKs used to associate two non-associative tables. However, in order to map RDB elements to ontology elements, we consider only the associative tables that represent a relationship between exactly two tables. We also take into account whether the table contains columns about the activation of the relation (often found in RDB models). The PK and FK columns and the database column used as "Active Flag" has to be explicitly configured in the *Metadata Definition Script*. Nevertheless, an associative candidate table can be forced to be mapped as a non-associative one if specified in the *Metadata Definition Script*.

Each associative table is mapped into two mutually inverse object properties. When available, logical relationship names are preferably used to name the object properties. Optionally, logical entity names that are part of the associative relation can be used. For each object property we inversely define *Domain* and *Range* according to the classes mapped from the two non-associative tables that are being associated. This rule is equivalent to Rules D, E and F in Table 1.

Example: in Figure 1, table SBC_PROCEDURE_ICD is an associative table between SBC_PROCEDURE and SBC_CID_10. Additionally, column FL_ACTIVE defines whether or not the associative relation is active. The logical associative relationship is named *"allows"* and *"is allowed to"*. The associative table is mapped to the corresponding object properties:

```
<Declaration>
  <ObjectProperty IRI="#DiseaseAllowsMedicalProcedure"/>
</Declaration>

<Declaration>
  <ObjectProperty IRI="#MedicalProcedureIsAllowedToDisease"/>
</Declaration>

<InverseObjectProperties>
  <ObjectProperty IRI="#MedicalProcedureIsAllowedToDisease"/>
  <ObjectProperty IRI="#DiseaseAllowsMedicalProcedure"/>
</InverseObjectProperties>

<ObjectPropertyDomain>
  <ObjectProperty IRI="#DiseaseAllowsMedicalProcedure"/>
  <Class IRI="#Disease"/>
</ObjectPropertyDomain>

<ObjectPropertyDomain>
  <ObjectProperty IRI="#MedicalProcedureIsAllowedToDisease"/>
  <Class IRI="#MedicalProcedure"/>
</ObjectPropertyDomain>

<ObjectPropertyRange>
  <ObjectProperty IRI="#DiseaseAllowsMedicalProcedure"/>
  <Class IRI="#MedicalProcedure"/>
</ObjectPropertyRange>

<ObjectPropertyRange>
  <ObjectProperty IRI="#MedicalProcedureIsAllowedToDisease"/>
  <Class IRI="#Disease"/>
</ObjectPropertyRange>
```

Database Columns

A database column represents a set of data values of a particular type, one value for each row of the table. We define a data property for the columns defined in the source RDB.

Rule 3. *Database Columns to Data Properties*

Each column from non-associative tables that is not part of a FK and that the content is not restricted by a check constraint with a list of possible values is mapped to data properties in the target ontology. We do not consider FK and check constraints, avoiding mapping mistakes with other rules, such as in rule 5. This rule is equivalent to Rule M in Table 1. For each data property, the domain of each property is specified using the corresponding mapped class from the table columun; the range is specified using the column data type.

Example: table SIP_PROCEDURE has a column named NM_PROCEDURE (*Medical Procedure Name*) with datatype VARCHAR(300), which is mapped to a data property with the corresponding *Domain* and *Range*, resulting in the OWL excerpt below:

```
<Declaration>
  <DataProperty IRI="#MedicalProcedureName"/>
</Declaration>

<DataPropertyDomain>
  <DataProperty IRI="#MedicalProcedureName"/>
  <Class IRI="#MedicalProcedure"/>
</DataPropertyDomain>

<DataPropertyRange>
  <DataProperty IRI="#MedicalProcedureName"/>
  <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
```

Database constraints

Database check constraints are a type of integrity constraint which can ensure that only specific values are allowed for a given column.

Rule 4. *Check Constraints to Object Properties*

This rule maps the columns not mapped by Rule 3, i.e., which contents are restricted by a list of possible values. Each constraint is mapped to an object property. It also creates a ontology class to represent the check constraint domain and the possible values are mapped as instances of the check domain class. The domain class is then used to set the object property range. There is no equivalent rule in Table 1.

Example: in Figure 1, column `FL_GENDER_RESTRICTION` in table `SBC_ICD_10` is associated with a check constraint that limits the possible column values to "M", "F" or "A" (*Male*, *Female*, or *All*), resulting in the OWL excerpt below:

```
<Declaration>
  <Class IRI="#GenderCheckDomain"/>
</Declaration>

<Declaration>
  <NamedIndividual IRI="#Female"/>
</Declaration>

<Declaration>
  <NamedIndividual IRI="#Male"/>
</Declaration>

<Declaration>
  <NamedIndividual IRI="#All"/>
</Declaration>

<ClassAssertion>
  <Class IRI="#GenderCheckDomain"/>
  <NamedIndividual IRI="#Female"/>
</ClassAssertion>

<ClassAssertion>
  <Class IRI="#GenderCheckDomain"/>
  <NamedIndividual IRI="#Male"/>
</ClassAssertion>

<ClassAssertion>
  <Class IRI="#GenderCheckDomain"/>
  <NamedIndividual IRI="#All"/>
</ClassAssertion>

<Declaration>
  <ObjectProperty IRI="#GenderRestriction"/>
</Declaration>

<ObjectPropertyDomain>
  <ObjectProperty IRI="#GenderRestriction"/>
  <Class IRI="#Disease"/>
</ObjectPropertyDomain>

<ObjectPropertyRange>
  <ObjectProperty IRI="#GenderRestriction"/>
  <Class IRI="#GenderCheckDomain"/>
</ObjectPropertyRange>
```

Rule 5. *Foreign Keys to Object Properties*

Columns that compose FKs between non-associative tables are not mapped to data properties. Instead, these columns are not mapped to data properties, but there will be two mutually inverse object properties for each FK. Although a FK can be a composite constraint (i.e. defined with two or more table columns), we consider the FK constraint as an entire object. For each object property we define the domain and range according to the classes mapped from the two non-associative tables that are part of the relationship. This rule is equivalent to Rules H and I in Table 1

Example: the `SBC_PROCEDURE` table (*Medical Procedure*) has a column `ID_COMPLEX_TYPE` which is defined as a *NOT NULL FK* constraint referencing the table `SBC_COMPLEX_TYPE` (*Complexity Type*) where the same column name is defined as the table PK. Column `ID_COMPLEX_TYPE` is mapped to a data property for the :1 table (`SBC_COMPLEX_TYPE`) but it is not for the :N table (`SBC_PROCEDURE`):

```
<Declaration>
    <ObjectProperty IRI="#isComplexityTypeOf"/>
</Declaration>

<Declaration>
    <ObjectProperty IRI="#hasComplexityType"/>
</Declaration>

<InverseObjectProperties>
  <ObjectProperty IRI="#hasComplexityType"/>
  <ObjectProperty IRI="#isComplexityTypeOf"/>
</InverseObjectProperties>

<ObjectPropertyDomain>
  <ObjectProperty IRI="#isComplexityTypeOf"/>
  <Class IRI="#ComplexityType"/>
</ObjectPropertyDomain>

<ObjectPropertyDomain>
  <ObjectProperty IRI="#hasComplexityType"/>
  <Class IRI="#MedicalProcedure"/>
</ObjectPropertyDomain>

<ObjectPropertyRange>
  <ObjectProperty IRI="#isComplexityTypeOf"/>
  <Class IRI="#MedicalProcedure"/>
</ObjectPropertyRange>

<ObjectPropertyRange>
  <ObjectProperty IRI="#hasComplexityType"/>
  <Class IRI="#ComplexityType"/>
</ObjectPropertyRange>
```

Rule 6. *Inheritance Relationships*

For each FK which is equivalent to a FK in non-associative tables, the class corresponding to the referenced table is defined as a superclass of the one that represents the non-associative table (this is usually known in RDB data models as a 1:1 relationship). We do not map the FK constraint to the corresponding object property, ignoring Rule 5 and we do not map the PK columns as data properties in the subclass. This rule is equivalent to Rule K in Table 1.

Example: *InfoSaude* has a generic table to store information about all the *Employees* (`SHC_EMPLOYEE`). However, for those who are health professionals, there is another table (`SIP_PROFESSIONAL`) which the PK is defined as a FK that references `SHC_EMPLOYEE`, forming an inheritance relationship that should be handled. This two tables are not shown in Figure 1.

```
<Declaration>
    <Class IRI="#Employee"/>
</Declaration>

<Declaration>
    <Class IRI="#Professional"/>
</Declaration>

<SubClassOf>
  <Class IRI="#Professional"/>
  <Class IRI="#Employee"/>
</SubClassOf>
```

Rule 7. *Not Null Constraints*

For each not null constraint we map the corresponding ontology property with *minCardinality* equals to

1. This rule is applied for database columns in non-associative tables, columns associated to a check constraint, and FK constraints.

The only way to find ontology individuals that break the *minCardinality* though the definition of ontology assertions for each individual that has a property P associated with a RDB NOT NULL constraint, is by setting such individual to a class *P max 0*, where P corresponds to the data or object property mapped from the database. We consider this rule as non-mandatory, and it should only be used when the target ontology will be used to check for missing individuals' properties. This rule is equivalent to Rule N in Table 1, not being restricted to data properties, but rather extended to object properties.

Example: the not null FK in the table `SBC_PROCEDURE` that references `SBC_COMPLEX_TYPE` is defined as NOT NULL in the RDB data model, resulting the following OWL tags for the corresponding object property in the target ontology.

```
<SubClassOf>
  <Class IRI="#MedicalProcedure"/>
  <ObjectMinCardinality cardinality="1">
    <ObjectProperty IRI="#hasComplexityType"/>
  </ObjectMinCardinality>
</SubClassOf>
```

Rule 8. *Unique Constraints*

We do not map unique constraints (including the PK constraint) to ontology elements. There are two reasons to it: a) single column with unique constraints should be mapped as inverse functional data properties, but inverse functional data properties are not defined in the OWL 2.0 specification; b) composite unique constraints cannot be mapped to ontology assertions using inverse functional data properties, unless a composite data property is defined to represent the composite value.

Object inverse functional properties are defined in the OWL 2.0 specification, and could be further used to represent NOT NULL FK constraints in the target ontology. However, this would require that the ontology individuals mapped from the table tuples should also be defined in a disjoint group of individuals, to explicitly assert that they are different to each other.

Database Tuples

A database tuple represents a tuple stored in one of the RDB tables. The analyzed solutions map RDB tuples to ontology instances. However, according to the table characteristics, we propose to map the database tuples to an instance or a class, depending if the table is set as part of an existing hierarchical structure.

Rule 9. *Tuples to Instances*

We map each database tuple that is not in transactional tables to an ontology instance. The columns are mapped according to its specific mapping. The column values become data properties and FK and check constraint values become object properties. This rule is equivalent to Rule S in Table 1.

Example: table `SBC_COMPLEX_TYPE` is not a transactional table. It has three tuples stored in the source RDB that are mapped to the corresponding ontology instances using the `DE_COMPLEX_TYPE` (*Complexity Type Description*) column, which is used to set the instance names, resulting the following OWL tags:

```
<Declaration>
    <NamedIndividual IRI="#HighComplexity"/>
</Declaration>

<ClassAssertion>
    <Class IRI="#ComplexityType"/>
    <NamedIndividual IRI="#HighComplexity"/>
</ClassAssertion>

<DataPropertyAssertion>
    <DataProperty IRI="#hasComplexityTypeId"/>
    <NamedIndividual IRI="#HighComplexity"/>
    <Literal datatypeIRI="&xsd;integer">3</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
    <DataProperty IRI="#hasComplexTypeDescription"/>
    <NamedIndividual IRI="#HighComplexity"/>
    <Literal datatypeIRI="&xsd;string">High Complexity</Literal>
</DataPropertyAssertion>

<ObjectPropertyAssertion>
    <ObjectProperty IRI="#hasStatus"/>
    <NamedIndividual IRI="#HighComplexity"/>
    <NamedIndividual IRI="#True"/>
</ObjectPropertyAssertion>
```

Ontology Subclasses

One class can inherit characteristics of another class by establishing a hierarchically schematized relationship. The class that inherits characteristics of the other class is a subclass and all its instances belong to the base class.

Rule 10. *Tuples to Subclasses*

Tuples that are exclusively used to hierarchically organize the content of other tables can be used as input to reproduce a similar hierarchical organization of subclasses. A table must have only one FK reference to its PK to be considered as a hierarchical table. We create a hierarchical class structure under the class corresponding to the table that owns the FK to the hierarchical table. This rule extends the idea behind Rule L in Table 1, using the content (tuples) of the referenced table to create the hierarchical structure, rather than just mapping the referenced table as one single superclass.

Example: tables `SBC_GROUP` and `SBC_SUBGROUP` are used to organize the *Medical Procedures* stored in the table `SBC_PROCEDURE`. The two FKs amongst those three tables are set as "hierarchical" in the *Metadata Definition Script*, making the tuples stored in `SBC_GROUP` and `SBC_SUBGROUP` to be mapped to subclasses under the *MedicalProcedure* class. Thus, instead of instantiating all the medical procedure tuples directly connected to its corresponding class, such instances can be organized in a two-level subclass hierarchy, according to the group and subgroup each medical procedure is associated. *Medical Procedure* instances are directly connected to the subclass corresponding to their subgroup.

In Figure 3, the instance corresponding to the "Cerebral Angio-Ressonance" procedure is not directly associated to the *MedicalProcedure* class, but to its corresponding subgroup (*MRI*).

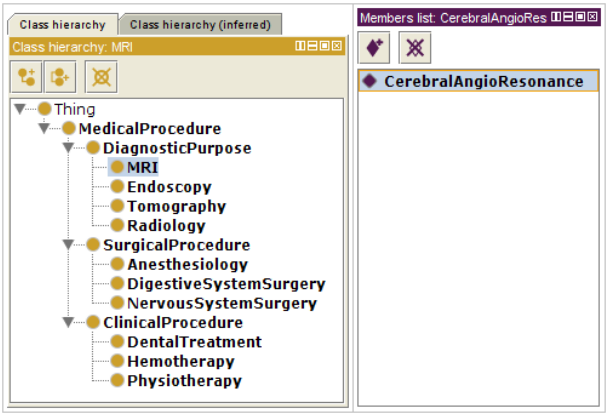


Figure 3 Hierarchical structure of *Medical Procedures*.

Database Columns with Self-hierarchical Structures

Hierarchies between database elements elements are defined by self-relationship within a single table. This is achieved by creating a new column with an integrity constraint (FK) that references the own table primary key (PK). When this occurs, this table represents a self-hierarchical structure.

Rule 11. *Self-hierarchical Columns*

When the columns represent a self-hierarchical structure, we create a similar hierarchical structure of subclasses. This rule is not present in existing related work.

This rule is used to map self-hierarchies that are not explicit in a "self" FK relationship and the hierarchies are encoded in the instances, not in the schema, using some specific encoding. For example, the "S08" element has sub-elements "S081", "S082", "S083"; "S081" could have elements "S0811", "S0812", etc. This means the hierarchical information is domain dependent and need to be interpreted by a specific function that is implemented.

Example: in table `SBC_ICD_10` (*Disease*), the PK column `ID_ICD_10` self-connects tuples without a FK definition (e.g. a tuple with `ID_ICD_10` = "S08" represents a group of diseases that comprises tuples with column `ID_ICD_10` in {"S081","S082","S088","S089"}). Thus, tuple "S08" is used to create a subclass of *Disease* instead of an ontology individual, whilst tuples comprising PK values in {"S081","S082","S088","S089"} will follow the Rule 9. However, they are associated with the corresponding subclass "S08" instead of the superclass *Disease*.

4.3.2 Naming ontology elements

Assigning name to ontology elements if often a difficult task, specially when dealing with data properties. For instance, we frequently find columns name `DESCRIPTION` in RDB models, but they could not be directly translated into several data properties. Acronyms, prefixes and suffixes are often adopted to name physical RDB elements. However, they are unlike to have any logical meaning, as much as they are used as a system encoding in order to physically arrange the elements. The adoption of the logical metadata information helps on handling this issue.

During the *Metadata* and *Data mapping* process, names of ontology elements are independently defined from each other.

In the *Ontology Generator* process, ontology elements can be renamed and hierarchically reorganized to avoid conflicts due to duplicated entities or attribute names. In this step, elements are renamed to include additional information to distinguish them (e.g. columns sharing the same logical name have the entity name concatenated to its name as a suffix), and a "super" element is created to handle the renamed ones in order to provide the common vocabulary of such knowledge domain represented by the RDB.

Example: in Figure 1, tables `SBC_PROCEDURE`, `SBC_COMPLEX_TYPE`, and `SBC_CID_10` have all a column named `FL_ACTIVE`. Although they have the same name, each column could be differently defined regarding its datatype, or check and NOT NULL constraints, leading to different ontology definitions for *Domain*, *Range* and *minCardinality*. Thus, the *Ontology Generator* process renames corresponding data properties originally defined as *Active* to three different names, including the name of the corresponding class as a prefix, and hierarchically organizing these data properties in a super data property with the original shared name. Figure 4 illustrates the hierarchical structure of all the `FL_ACTIVE` columns mapped to data properties in the target ontology.

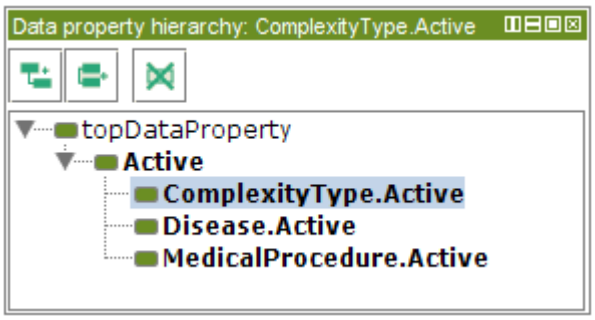


Figure 4 Hierarchical structure created for the columns named as `FL_ACTIVE` in the RDB.

The approaches used to rename the ontology elements can be slightly different according to each element type. Object and data properties can use the corresponding class name as a suffix, as well as the schema name when multiple schemata are being used to create a target integrated ontology. Moreover, ontology individuals can use the class name or the content of PK columns as prefixes to the content of the main column used to textually describe a table tuple.

Independently of the method used to rename ontology elements, we strongly suggest to reorganized those elements hierarchically in order to keep a common vocabulary free of encodes.

4.4 Rule execution flow

We describe the rule execution flow in Algorithm .1. The rules are executed following a specific order, constrained by the types of the input elements and additional conditions. The automated process of our Architecture receives as input the *Metadata* and *Data* definitions, which are previously generated and/or manually completed and it iterates through the input file lines. *MetadataMapping()* is a procedure that gets all RDB elements and checks the type being processed, following the sequence: *Tables*, *Columns* and *Tuples*. For each of them, there is a specific method (*getTableType*, *getTupleType* and *getTupleType*) that gets the element in the specific position in the definition file. After the RDB element is matched, the procedure identifies the appropriated rule to be executed. Each rule transforms ALL elements of a given type to the target ontology. The RDB elements mapped are stored in the *Mapping Repository* so that at the end of the process it is possible to generate an OWL File as output.

Algorithm .1: Pseudocode of Automated Process

```
1  Input: Metadata and Data Definitions;
2  Output: OWL File;
3  Procedure: MetadataMapping()
4  for Line ln : {Metadata ∪Data}.getLine() do
5      type = ln.getElementType()
6      if type is Table then
7          if ln.getTableType() is non-associative then
8              Rule_1 (ln);
9          else
10             Rule_2 (ln);
11         end
12     else
13         if type is Column then
14             if ln.getColumnType() is not FK AND
                ln.getColumnType() is not Check
                Constraint then
15                 Rule_3 (ln);
16             else
17                 if ln.getColumnType() is Check
                    Constraint then
18                     Rule_4 (ln);
19                 else
20                     Rule_5 (ln);
21                 end
22                 Rule_7 (ln);
23             end
24         else
25             if type is Tuple AND ln.getTupleType() is
                from TableConcepts then
26                 Rule_10 (ln);
27             else
28                 if ln.getTupleType() is Self_hierarchical
                    then
29                     Rule_11 (ln);
30                 else
31                     Rule_9 (ln);
32                 end
33             end
34         end
35     end
36 end
37 TargetOntology ←MapRepository;
```

5 Experiments

The evaluation of the generated ontology is done in two steps. First, we check if the ontology is correctly built. Second, we verify if the output data complied after the execution of the mapping rules, through the inference of logical consequences using a reasoner. We do not use a gold standard ontology for comparing the result, as proposed by the method of [45], because the application domain of this work has not been mapped to a previous ontology, making such comparison unfeasible.

We performed two different experiments to evaluate our approach. First, we present the obtained results by executing a full mapping process, to show the feasibility of rules and tool [7] . We use a RDB physical and logical model of a dental care system. Second, we manually applied the rules to the *InfoSaude* system. The objective was to compare the resulting ontologies with one important previous work.

5.1 Experiment 1: RPL$_2$O architecture implementation

In this experiment we developed a tool handling all the RPL$_2$O components. We chosen a RDB model from a dental care system, containing 25 tables, 45 primary keys, 1 unique key, 42 foreign keys, 27 check constraints, 214 columns with no constraints and 696 tuples. The RDB logical model of this subset has 229 columns with logical name.

7 Available at https://github.com/caghuve/rdb-to-onto

We applied our tool in three scenarios. In scenario 1, only the RDB physical model elements were used; in scenario 2 we used the RDB physical and logical elements; in scenario 3, we manually removed the RDB elements that were not part of the application domain, such as information about system configuration, logs and others. This yield a difference of 143 columns. Consequently, each scenario has a different *Metadata Definition Script*.

Table 2 presents the number of generated ontology elements for each scenario. We classified the results of different ontology elements by scenario. Scenarios 1 and 2 resulted in an ontology with the same number of elements. The difference is on the name on the output elements for scenario 2, since we used the RDB logical model as additional input. Figure 5 shows an excerpt of the resulting ontologies for scenarios 1 (left side) and

Table 2 Number of ontology elements

Elements	Scen. 1	Scen. 2	Scen. 3
Class	31	31	31
Sub class of	344	344	203
Object property	73	73	73
Inverse	23	23	23
Domain	73	73	73
Range	73	73	73
Data property	236	236	95
Domain	236	236	95
Range	236	236	95
Instances	744	744	744
Object property assertion	0	0	0
Data property assertion	8	8	8
Class assertion	744	744	744

2 (right side). In scenario 3 we observed a difference on generated elements. The number of RDB columns with no constraints in scenario 3 is 73, thus resulting in a smaller amount of data properties. The number of subclasses was also small, according to the definitions of *Rule 7*. The *Object property* elements were generated from Rules 2, 4 and 5. The number of object property is the same in the 3 scenarios, since the number of elements in the scenarios does not change. In scenario 3, The *Instances* and *Object property assertion, Data property assertion*, and *Class assertion* were generated from Rule 4 and 9. The tool did not generated *Object property assertion* because the column values were mapped only to *Data property*.

From RDB model with 27 check constraints, only 8 contains different values and 19 of them have values that repeat in different tables e.g. *gender: male / female*, so they were used to generated 27 object properties, 8 classes from a total of 31 to represent the check constraint domain and 48 instances, referring to possible columns values. Considering the 48 generated instances (from a total of 744) related to possible values of check constraint and the instances generated from the 696 tuples, 744 instances were created at the end of the mapping.

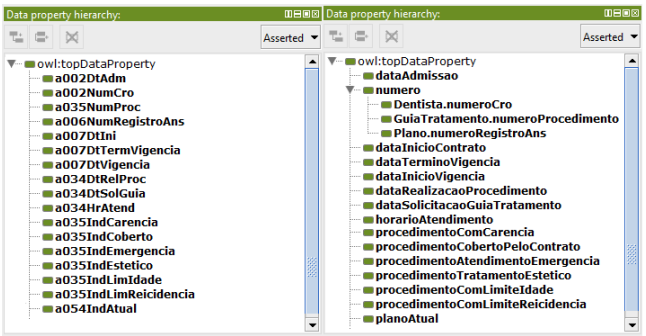


Figure 5 Sample target ontology for scenarios 1 and 2.

For each target ontology, we individually checked the fulfillment of the definitions established in each one of the Mapping rules; the naming and correct application of specific characteristics of each element; the correct grouping of elements which could generate duplicity; and the establishment of relationship among the database elements in the Mapping Schema. The architecture validation was performed in each of the architecture components. We observed its behavior and the ontology representativenes in each of the scenarios. It was a hard and manual validation task, but important to analyze the coherence of the rule definitions, and regarding the well functioning of *Mapping Repository*. Therefore, we observed a greater clarity in the ontology element names of the second scenario, compared to the first scenario, highlighting one of the benefits gained through the use of the *Logic Data Model*, which increases the ontology legibility.

5.2 Experiment 2: comparing with other solutions

To show how our approach differs from some of the previous works [15, 6, 4], we compare the resulting ontologies performing a manual mapping using a subset of mapping rules that affects the ontology hierarchical structure (classes and properties). We select these works because they clearly presents the definition and application of their mapping rules, unambiguously. In this way it was possible to calculate the number of generated rules and to compare to our approach. However, it would be more adequate if we had access to the implemented code, since it could have bugs or different interpretations. It is subject of a future work to use the same strategy and to do an exhaustive comparison with other existing approaches.

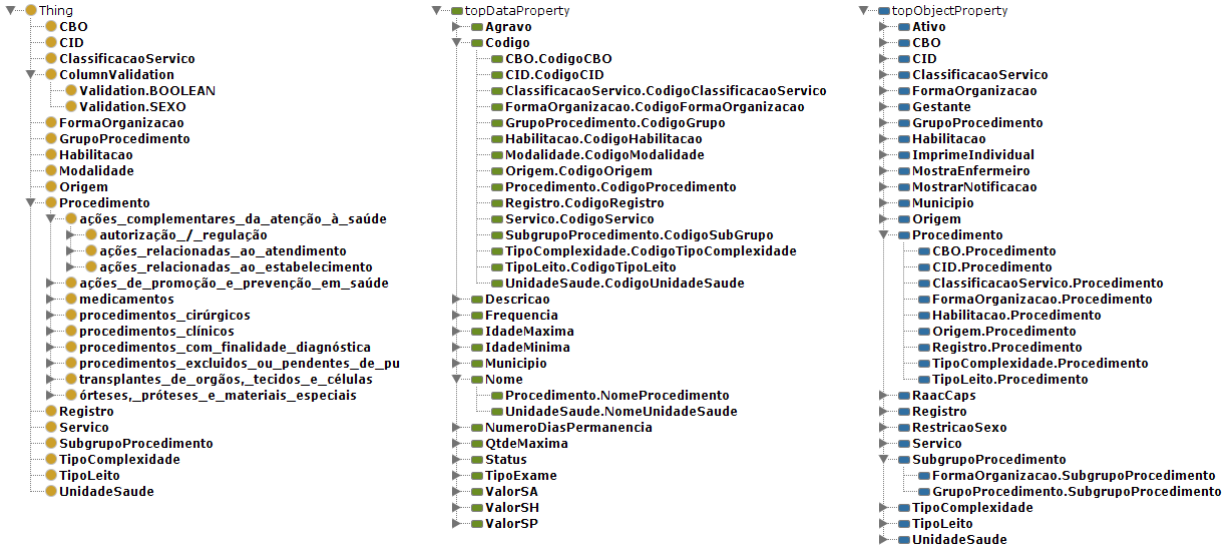
We used a subset of the *InfoSaude* system, comprising 24 tables, 103 columns, and 24 foreign keys. We present a set of illustrations to depict the differences. We use a similar setting of scenario 3, considering only the domain elements, and using the logical model in our approach. Table 3 depicts the number of ontology elements created in each approach.

Lines 1 and 2 in Table 3 refer to the classes mapped from non-associative tables and object properties mapped from associative tables. Our approach differs from others by considering as associative tables even those with additional boolean attributes that are used to set whether the relation is positive or negative. All associative tables in the *InfoSaude* system have an additional boolean attribute and, thus, they are not recognized as associative tables in the other approaches. Additionally, other approaches are focused in the physical data model. Instead, we use the logical nomenclature in order to provide a proper vocabulary.

Table 3 Target ontology elements in different approaches.

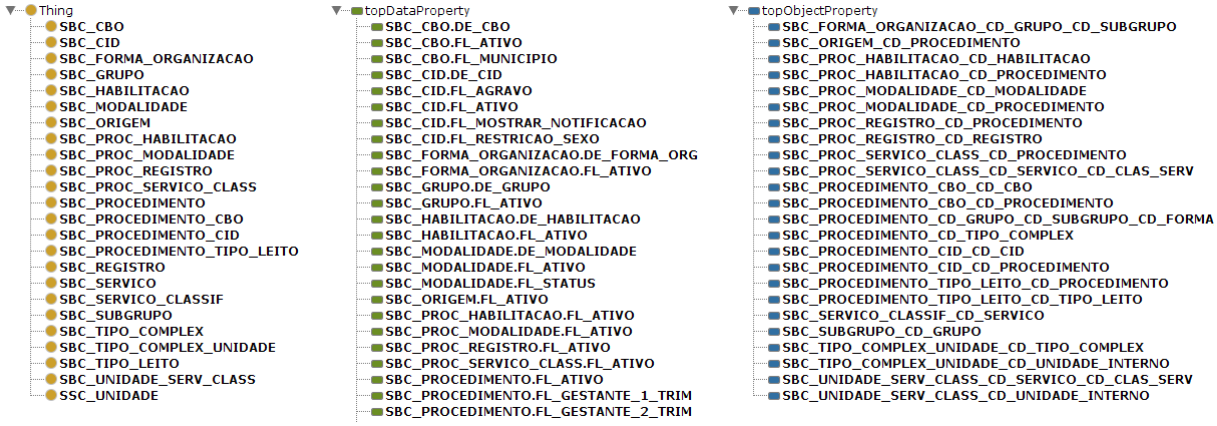
	[6]	[4]	[15]	RPL_2O
1. Main Classes	24	24	24	15
2. Associative Obj Prop	-	-	-	16
3. FK Obj Prop	48	24	24	12
4. Super Obj Prop	-	-	-	22
5. Data Properties	62	103	58	41
6. Super Data Prop	-	-	-	14
7. Validation Classes	-	-	-	2
8. Validation Individuals	-	-	-	2
9. Validation Obj Prop	-	-	-	25
10. Hierarchical Classes	-	-	-	480

Figures 6 and 7 contrast the resulting ontology based on our proposed set of rules against the ontology produced by [15]. Figure 6(a) shows some of resulting



(a) Classes (b) Data Properties (c) Object Properties

Figure 6 Sample of ontology elements resulting by applying RPL_2O mapping rules.



(a) Classes (b) Data Properties (c) Object Properties

Figure 7 Sample of ontology elements resulting by applying [15].

classes following our approach. Subclasses under class *ColumValidation* result from Rule 4 (Check constraints to Object Properties) and all subclasses levels under class *Procedimento* result from Rule 10 (Tuples to Subclasses). Although disregarding the fact of missing logical names, the amount of classes resulting by [15] is lower, and no class hierarchy is produced (Figure 7(a)). Despite being in Portuguese, both figures enable to understand the differences in the elements hierarchy and naming.

Figures 6(b) and 6(c) show a sample of the object and data properties resulting from our proposed mapping rules. The super-property *Procedimento* groups all the object properties mapped from those foreign keys that reference the table `SBC_PROCEDURE`. Similarly, the super-property *Nome* groups all the data properties referring attributes that correspond to names. In contrast, Figures 7(b) and 7(c) shows how the set of properties is usually mapped following previous works, in which no property hierarchy is produced.

6 Conclusions

In this work we present an approach for mapping relational databases to ontologies using logical and hierarchical information. Our solution takes stock of previous works and provides a more comprehensive set of rules, handling cases uncovered by the literature. We were motivated by the implementation of a real-world scenario, which lead us to identify different open issues. Since it is hard to state that one set of rules for a given solution is more appropriate than another, we provide a classification of existing solutions with respect to the kind of rules and we have identified several open issues yet to be handled, finding as the main drawbacks the utilization of the RDB schema with no logical information and little support to hierarchical data.

The main contributions of our approach are the following. First, we use the RDB logical information for generating the target ontology. This enables having more

understandable names for the ontology elements, which can be crucial when inferring complex relations. Second, we go further than existing approaches by designing and developing rules to handle hierarchical and self-hierarchical structures by using domain information from the RDB instances, such as the FK constraints. This means our approach does RDB schema migration, but we also use tuples from a subset of the RDB tables to enrich the output ontology. We also present a novel implementation for mapping N-to-N relationships and we take database constraints into account. Finally, we present a set of possibilities for handling ontology names, and we also present directions towards alternative solutions. Our architecture is independent from the input database schema, since we use standard SQL code to access the database elements. Once extracted, the logical and physical models together with the metadata and data information are stored in to separate mapping files – data and metadata files respectively. By ssing distinct metadata reading components ii is possible to support different database vendors.

We validate our approach with two real-world scenarios in the healthcare domain: first we apply our semi-automatic approach in a dental care system RDB model in order to empirically analyze the generated ontology. The tool is available for download. Second, we use the RDB model from the *InfoSaude* system to validate all the designed mapping rules and compare our approach against existing solutions.

As future work, we aim to track the mapping process for verification of completeness. The implemented experiments are forward migration, but the script definitions are generic enough and could be applied in the backward scenario, using the mapping for bidirectional transformation and traceability on querying the relational database through the ontology concepts.

References

[1] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[2] Timo Weithöner, Thorsten Liebig, and Günther Specht. Efficient processing of huge ontologies in logic and relational databases. In *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops*, pages 28–29. Springer, 2004.

[3] Lei Zhang and Jing Li. Automatic generation of ontology based on database. *Journal of Computational Information Systems*, 7:4:1148–1154, 2011.

[4] C Ramathilagam and ML Valarmathi. A framework for owl dl based ontology construction from relational database using mapping and semantic rules. *International Journal of Computer Applications*, 76(17):31–37, 2013.

[5] Konstantinos N. Vavliakis, Theofanis K. Grollios, and Pericles A. Mitkas. Rdote - transforming relational databases into semantic web data. In Axel Polleres and Huajun Chen, editors, *ISWC Posters&Demos*, volume 658 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[6] Mohammed Reda Chbihi Louhdi, Hicham Behja, and Said Ouatik El Alaoui. Transformation rules for building owl ontologies from relational databases. *Computer Science & Information Technology (CS & IT)*, 3:271–283, 2013.

[7] Mona Dadjoo and Esmaeil Kheirkhah. An approach for transforming of relational databases to owl ontology. *arXiv preprint arXiv:1502.05844*, 2015.

[8] Irina Astrova, Nahum Korda, and Ahto Kalja. Rule-based transformation of sql relational databases to owl ontologies. In *In Proceedings of the 2nd International Conference on Metadata and Semantics Research*, 2007.

[9] Dimitrios-Emmanuel Spanos, Periklis Stavrou, and Nikolas Mitrou. Bringing relational databases into the semantic web: A survey. *Semantic Web*, 3(2):169–209, 2012.

[10] Juan F Sequeda, Syed Hamid Tirmizi, Oscar Corcho, and Daniel P Miranker. Survey of directly mapping sql databases to the semantic web. *The Knowledge Engineering Review*, 26(4):445–486, 2011.

[11] Franck Michel, Johan Montagnat, and Catherine Faron-Zucker. *A survey of RDB to RDF translation approaches and tools*. PhD thesis, I3S, 2014.

[12] Zdenka Telnarova. Relational database as a source of ontology creation. In *IMCSIT*, pages 135–139, 2010.

[13] Noreddine Gherabi, Khaoula Addakiri, and Mohamed Bahaj. Mapping relational database into owl structure with data semantic preservation. *CoRR*, abs/1205.5922, 2012.

[14] Irina Astrova. Rules for mapping sql relational databases to owl ontologies. In *Metadata and Semantics*, pages 415–424. Springer, 2009.

[15] Yutao Ren, Lihong Jiang, Fenglin Bu, and Hongming Cai. Rules and implementation for generating ontology from relational database. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 237–244, 2012.

[16] Nadine Cullot, Raji Ghawi, and Kokou Yétongnon. Db2owl: A tool for automatic database-to-ontology mapping. In *SEBD*, pages 491–494, 2007.

[17] Man Li, Xiaoyong Du, and Shan Wang. A semi-automatic ontology acquisition method for the semantic web. In *Advances in Web-Age Information Management*, pages 209–220. Springer, 2005.

[18] Michal Laclavik. Rdb2onto: Relational database data to ontology individuals mapping. In *Tools for Acquisition, Organisation and Presenting of Information and Knowledge*, pages 86–89, 2006.

[19] Guntars Būmans and Kārlis Čerāns. Rdb2owl: A practical approach for transforming rdb data into rdf/owl. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, pages 25:1–25:3, New York, NY, USA, 2010. ACM.

[20] Humaira Farid, Sharifullah Khan, and Muhammad Younus Javed. Dsont: Dspace to ontology transformation. *J. Information Science*, 42(2):179–199, 2016.

[21] Satya S Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda, and Ahmed Ezzat. A survey of current approaches for mapping of relational databases to rdf. *W3C RDB2RDF Incubator Group Report*, pages 113–130, 2009.

[22] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. A direct mapping of relational data to rdf. w3c recommendation 27 september 2012, 2012.

[23] Juan F Sequeda, Marcelo Arenas, and Daniel P Miranker. On directly mapping relational databases to rdf and owl. In *Proceedings of the 21st international conference on World Wide Web*, pages 649–658. ACM, 2012.

[24] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2rml: Rdb to rdf mapping language, w3c recommendation 27 september 2012. *Cambridge, MA: World Wide Web Consortium (W3C)(www. w3. org/TR/r2rml)*, 2012.

[25] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. A direct mapping of relational data to rdf (2012). *URL http://www. w3. org/TR/2012/REC-rdb-direct-mapping-20120927*, 2012.

[26] Yannis Marketakis, Nikos Minadakis, Haridimos Kondylakis, Konstantina Konsolaki, Georgios Samaritakis, Maria Theodoridou, Giorgos Flouris, and Martin Doerr. X3ml mapping framework for information integration in cultural heritage and beyond. *Int. J. Digit. Libr.*, 18(4):301–319, November 2017.

[27] Vishal Jain and Mayank Singh. A framework to convert relational database to ontology for knowledge database in semantic web. *International Journal of Scientific & Technology Research*, 2:9–12, 2013.

[28] Mohamed A. G. Hazber, Ruixuan Li, and Xiwu Gu. Integration mapping rules: Transforming relational database to semantic web ontology. *Applied Mathematics Information Sciences*, 10(3):881–901, 2016.

[29] Lama Al Khuzayem and Peter McBrien. Owlrel : Learning rich ontologies from relational databases. 2016.

[30] Farid Cerbah. Mining the content of relational databases to learn ontologies with deeper taxonomies. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 553–557. IEEE, 2008.

[31] Hegler Tissot. HEXTRATO: using ontology-based constraints to improve accuracy on learning domain-specific entity and relationship embedding representation for knowledge resolution. In *IC3K 2018 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 1, pages 72–81. SciTePress, 2018.

[32] Hegler Tissot and Richard Dobson. Identifying misspelt names of drugs in medical records written in portuguese. *HealTAC-2018: Unlocking Evidence Contained in Healthcare Free-text*, 2018.

[33] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems*. McGraw Hill, 2003.

[34] Ramez Elmasri and Shamkant Navathe. Fundamentals of database systems sixth edition pearson education. *Reproduced with permission of the copyright owner. Further reproduction prohibited without permission*, 2011.

[35] Abraham Silberschatz, Henry F Korth, S Sudarshan, and Daniel Vieira. *Database System Concepts*. Elsevier, 2006.

[36] Tony Lawson. A conception of ontology. *Unpublished manuscript, University of Cambridge*, 2004.

[37] Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5):907–928, 1995.

[38] Denny Vrandečić. Ontology evaluation. In *Handbook on Ontologies*, pages 293–313. Springer, 2009.

[39] Nicola Guarino. Formal ontology, conceptual analysis and knowledge representation. *International journal of human-computer studies*, 43(5):625–640, 1995.

[40] John Davies, Dieter Fensel, and Frank Van Harmelen. *Towards the semantic web: ontology-driven knowledge management*. John Wiley & Sons, 2003.

[41] Mike Dean, Guus Schreiber, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language reference. *W3C Recommendation February*, 10, 2004.

[42] Owl web ontology language guide. http://www.w3.org/TR/owl-guide/. Accessed: Jun, 2015.

[43] Owl web ontology language overview. http://www.w3.org/TR/owl-features/. Accessed: Jun, 2015.

[44] Teresa Podsiadły-Marczykowska, Tomasz Gambin, and Rafał Zawiślak. Rule-based algorithm transforming owl ontology into relational database. In *Beyond Databases, Architectures, and Structures*, pages 148–159. Springer, 2014.

[45] Nicola Guarino. Toward a formal evaluation of ontology quality. *IEEE intelligent Systems*, 19(4):78–79, 2004.