DE GRUYTER OPEN
RIGA TECHNICAL UNIVERSITY
Information Technology and Management Science
ISSN 2255-9094 (online)
ISSN 2255-9086 (print)
December 2016, vol. 19, pp. 85–91
doi: 10.1515/itms-2016-0016
https://www.degruyter.com/view/j/itms

Improved Database Schema Development for OWL2

Henrihs Gorskis
Riga Technical University

***Abstract* – This paper proposes a novel approach to storing ontologies in relational databases. The approach consists of a database schema, which was created to be capable of storing ontology information defined in OWL2 (Web Ontology Language 2) functional syntax. The paper explains how the schema has been designed and what advantages it offers to any user. The described schema is part of a larger system. This paper also discusses how the schema cooperates with the external system, which, however, is outside the scope of this paper, to successfully create, store and retrieve ontology knowledge from the functionality offered by the relational database. Further, this paper describes the implementation of the proposed method in prototype software. The described schema shall be the first step of the creation of an ontology-based database access system.**

***Keywords* – Database, intelligent system, ontology, semantic knowledge.**

I. Introduction

Ontology describes entities using concepts, individuals and properties. The entities described by the ontology are important in the context of some domain. Therefore, the ontology also describes the domain. Usually the knowledge contained in the ontology is semantic by nature. This means that the ontology describes similar or related entities in a way analogous to the human language. In information technology, ontology is related to artificial intelligence and systems capable of some reasoning. It is concerned with the definition and relation of terms. Semantic knowledge stored in an ontology model is a powerful tool to describe and define conceptual information in addition to existing data. This allows for the classification of data with additional concepts. By examining the properties of some units of data and comparing these properties to descriptions within the ontology, it is possible to conclude associations to named and unnamed concepts. It, therefore, provides additional descriptive information to the user of the data. This makes ontology perfect for an additional meta layer on top of classical data structures to describe knowledge about the data in a way that the data themselves do not provide.

In whatever way the system uses its knowledge, it first has to store it somewhere. There are already many storage solutions for ontology knowledge. However, these solutions are not always the best approach in all situations. This paper presents a database schema for fast and simple storage of ontology knowledge in a relational database. The proposed schema is tuned for systematic extraction of information on an entity-to-entity basis. This means that the stored information about ontology entities can be extracted step-by-step, without the need for obtaining all ontology information at the same time.

The schema described in this paper is part of a larger system using ontology definitions for accessing data in a typical relational database. One of the abilities of this related system, which requires ontology, is to access related data named by an ontology concept. This is part of a larger project still, which aims to integrate ontology capabilities into existing solutions. There are already many systems that use databases for their needs. In order to integrate ontology functionality into those systems, an additional, low-impact module is desired. Figure 1 shows how an ontology model could be added to an existing system. The requirements of the system are the possibility of accessing the ontology from the already existing database, without drastically changing its structure or having a large impact on it. The primary purpose of the database schema is to work with this system. However, due to the schema general design and ease of use, it can be applied in other situations as well.

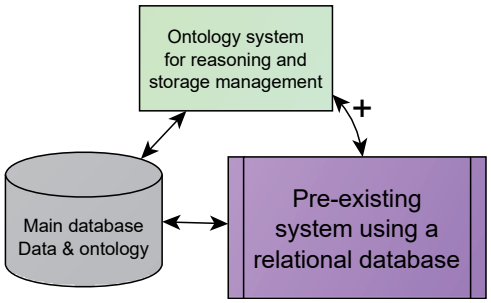

Fig. 1. System integration scheme.

II. Current Ontology Storage Solutions

The existing solutions for ontology storage range from very complex to fairly simple. However, none of the solutions found in other papers are completely satisfying the intended needs of the previously mentioned system. As a previous analysis of existing solutions has shown [1], many of them have a large impact on the database. For example, the solutions provided by the "Apache Jena" project [2] require additional databases, specifically formatted for their intended use. Their first solution for ontology persistence is called TDB and is used to store triples. The second solution proposed by Jena uses a separate database service, altogether, called "Fuseki". This would require the second database management system purely for ontology storage. Both solutions would have a considerable impact on any existing system.

An additional obstacle to already existing solutions is that many technologies described in literature have already aged and are not usable today. Even popular tools, such as Protégé, have the problem that some of the technologies developed for a previous version are not available anymore in the newest version. All these factors have led to the need to develop a simpler solution.

There are also approaches that use common relational databases to store ontology information [3].

Conversions from OWL to database schemas can be performed by using RDF information provided in XML [4], storing a very raw representation of the XML data in the ontology. More functional methods generate relational database schemas [5], [6] or object-oriented database types and schemas [7], [8] from ontology knowledge. The creation of database schema from an RDF centric approach is also possible [9]. These approaches create a database schema based directly on the knowledge contained in the ontology. As a result, this will yield different ontological domain descriptions in different schemas. Such an approach has the advantage of never having unused tables or columns, since the entire structure is based on the existing knowledge. Another advantage is that any type and data are represented in the database using primitive types of the database and ontology related tables can be directly unified with other data in the database. Moreover, some restrictions are implemented using database functionality. The disadvantage is that the generated schemas are not universal. Some of these approaches are not capable of translating all the information contained in the OWL description. Furthermore, the sizes of the schemas are dependent on the amount of knowledge in the ontology. A larger description will result in a larger schema. Some of the methods for creating database schemas from ontology create some generalised tables [10]. The considered methods sometimes use concept names directly as references to concepts. This differs from the proposed schema, which uses identification numbers.

Another approach is to use the database to store mapping rules to ontology concepts [11] described in RDF triplets. Some approaches work directly with OWL solutions and files. It often seems to be desired to add functionality to OWL and ontology in general, which is not typically present. For example, [12] presents an approach for adding constraints to OWL, so that it is better suited for data input.

Some approaches store ontology information in files; others store them in special databases. In the case when the ontology-based system uses a specialised database, it leads to two possibilities: (1) the specialised database is a separate service requiring the maintenance of a separate database solution or (2) the system uses an existing relational database solution, but transforms or creates a database schema to a specific structure based on its needs. This usually makes it very difficult to store additional information alongside.

III. Design of the Independent Simple Schema

The method described in this paper is based on the popular web ontology language OWL2 and, specifically, its functional syntax. The reason why the functional syntax has been chosen is that it closely follows the structural specification of OWL2 and most directly exposes the purpose of OWL2 [13], [14] without lengthy definitions of basic parts of the ontology, unlike the XML/RDF (Resource Description Framework) syntax. By basing the database scheme design on the needs of OWL2, it is anticipated that the information stored in such a schema will be capable of describing all the important aspects of any ontology knowledge. The present research is a continuation of a previous database scheme proposed in [1]. Some important changes are the considerable reduction of the table count. This has been done to simplify the approach and minimise the impact on the database in cases when the ontology is stored alongside other pre-existing information. In addition, many tables in the original design did not hold unique data, which would justify them to be in separate tables, because the difference could be inferred. The final database schema consists of only six tables. Figure 2 shows the final structure of the database, its tables, columns and type values.

A. Types and References

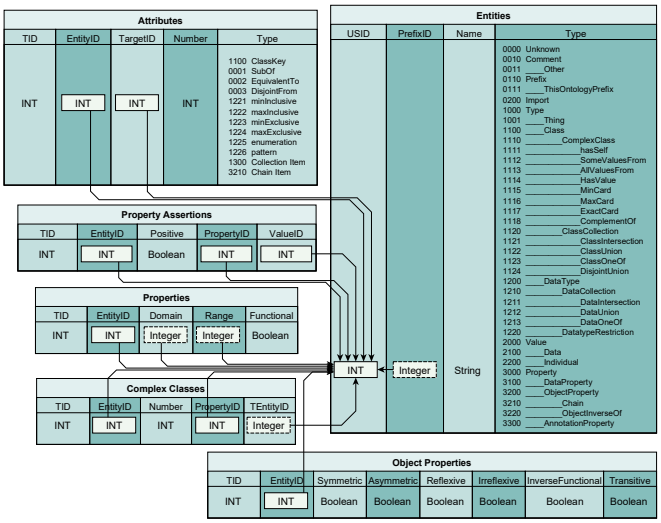

Fig. 2. The database schema.

The “entities” table has a column describing the type of the entity and the “attributes” table has a column describing the type of the relation or attribute relation between two entities. Usually in cases when there is a type in databases, there are several approaches for implementing them. The first approach is to create an enumerator if the database offers such capabilities. The second approach is to create a separate table, holding the different values and referencing these values by ID. There is the third, less elegant approach of simply using an integer value for the type, and pushing the burden of understanding of the numeric value to the user or related system. Since the proposed schema is more concerned with the storage of ontology information and it is desired to do so with the smallest impact, exactly this approach has been chosen. Since enumerators are not supported by all databases, it was decided not to use them. Enumerators are also difficult to maintain because changing them leads to the corruption of the database. It was also decided not to use an additional table, exclusively for storing unique values, to keep the impact of the ontology storage low. The related ontology-extraction system, for managing ontology information in the database, has to have the implementation of the types used in the tables. Another reason, why an external solution to type values has been chosen, is the hierarchical relations between the types. The type “individual” is an extension of the type “value”. “SomeValuesFrom” is an

extension of "Complex class", which, in turn, is an extension of "Class" and so on. This hierarchical approach allows multiple related types of entities to be obtained at the same time. It was chosen to implement this functionality using external solutions.

As can be seen from Fig. 2, all references in the schema are connected exclusively to the "entities" table. This means that all references are external keys to the unique primary key of the entity. Although only entities are referenced, all tables have been given identifiers. This is done for simplicity of maintaining the database. For example, if a record in one of the tables needs to be updated, the update function can use ID to reference the specific record.

B. Ontology Functions of the Schema Tables

At the very core of the schema, there is the entity table. It contains all unique ontology entities. An entity is a piece of named or unnamed information describing a concept, individual, property or value. The entity is uniquely identified by a prefix and name. It is the combination of prefix and name, which should be unique in the ontology. Most self-sufficient ontology descriptions, in most cases, should not have two or more entities with the same name. It is only in cases when entities from other ontology descriptions or schemas are used, which necessitates the use of prefixes to distinguish them from each other. Name and prefix are, of course, related, since they only both together provide the full URI (Uniform Resource Identifier) of the entity. The other distinguishing feature, apart from the name and prefix, is the entity type. Any entity should have only one type. Type conflicts can arise only in cases of "OWL full", when any given entity can be a concept and an individual, and maybe others, at the same time. The schema described in this paper focuses on storing "OWL DL" (Description Logic) type ontology knowledge. The type of an entity is an important part of this approach, since its secondary purpose is to provide hints to the ontology information retrieval systems about where to obtain additional information from the database. For example, if the entity has a "property" type, it means that there is a hint to search the "Properties" table for additional information about this entity. The function of all the other tables in this schema is to provide additional information about the entities from the "Entities" table.

The "Properties", "Object properties" and "Complex classes" tables are the only tables, which directly extend the entity. They are connected to the entity table in a one-to-one relationship. The columns "Entity ID" in these tables are used to trace the relationship to the core entity. In case of an entity of the type "Object property", both the tables "Properties" and "Object properties" are needed to provide all information about the object property aspects of the entity. In case of data property, only information from the "Properties" table is required. The table "Property assertions" is used for individuals and annotations to define relationships between entities using properties. The assertions are made about the relations of one individual to both, other individuals using object properties, and data using data properties. Since there can be multiple types of annotations defined by annotation properties, they also need to be described by using this table. Of course, annotation assertions can only be positive.

The "Complex classes" table provides the descriptions necessary for the definition of some, but not all complex classes. This table is used only for those complex classes that are defined by their relationship to other entities using properties. This includes cardinality classes, but excludes complex classes based on grouping. This is conceptually different from the property assertion table. Cardinality classes are the ones using the column "Number" for the restriction of minimal, maximal or the exact count of relations. Collection-based complex classes are defined by grouping not by relationships; therefore, they do not require any data from this table. Groups are defined using the "Attributes" table.

The table named "Attributes" provides all the information about how entities are directly related to each other, without the use of property relations in between. This table includes information about subtyping, classification, distinction and grouping. The relation is defined by providing references to both entities and the type of the relation. Some relations, such as the equality relation, can be viewed as directionless. Others are directed from the first entity, pointed to by the column "Entity ID", to the second one, pointed to by the column "Target ID". Collections are defined by pointing from the entity representing the collection itself to all its members. Most collections are unordered, except for the property chain, which is also viewed as a collection in the described approach. Since the order of the property entities, used to define the chain, is important, they have to be ordered. The order is achieved by providing a number, describing the position of the entity within the chain.

In response to the complexity of defining a new data type by using data type restrictions, it has been decided to exclude a table responsible for that task. Instead, all data type restrictions are defined by creating a data type entity and using attributes to describe the restrictions.

C. Translation of Statements Written in the OWL2 Functional Syntax to the Database Schema

This section describes how different statements written in the OWL2 functional syntax are translated and stored in a database using the proposed schema.

Prefixes. Prefixes are defined in the functional syntax using the operator "Pre-fix". Prefixes are added to the database using two records in the "Entities" table. Both records will have the prefix type assigned to them. First, the full prefix URI is added. The full version of a prefix, for example, "http://www.w3.org/2002/07/owl" as its name, does not have a value in its prefix columns, in the table. Once the full URI exists in the database and has an ID associated with it, the second prefix entity is created. The second entity holds the short version of the prefix in its name, for example "owl". This prefix entity also does not have a value in its prefix column. Entities of the prefix type should be the only ones without a prefix themselves. All other entities should have a prefix, usually the ontology main prefix if no other is specified. The short prefix is linked to the full URI using a record in the "attributes" table of the type "EquivalentTo".

The OWL2 functional syntax also uses the operator "Ontology". The ontology definition is used in OWL2 documents to relate all other entities and relation to the ontology they are defined in. It is the responsibility of the converter to keep track of entities belonging to the ontology. As far as the schema is concerned, this keyword defines the main ontology prefix. By creating a prefix using the "ThisOntologiesPrefix" type, a prefix for the ontology is defined. Any entities defined within the "Ontology" operator will be given the ontology prefix automatically.

Imports. Imports are defined using the operator "Import". Handling of imports is outside the scope of this paper. However, an entity of the type "Import" is created in the "entities" table. This is done so as not to lose the reference to other ontology. Depending on the implementation of the translation function, the referenced ontology can be stored in the database as well. Otherwise, the referenced ontology has to be accessed, when needed, from an external system.

Declarations. The following operators are used in OWL2 to declare the existence of named entities: "Declaration", "Class", "DataProperty", "ObjectProperty", "NamedIndividual", "Datatype" and "AnnotationProperty". All these keywords are related to the declaration of new entities and are usually found at the beginning of the ontology in the OWL2 document. The result of translating them into the database is very similar. For each type, a new entity is created with the name provided in the declaration. The entity is linked to its prefix. The type is assigned accordingly. In case of any property declaration, a record in the "Properties" table is also created. This record is linked back to the newly created entity using the "Entity ID" column in the "Properties" table. In the case of an object-property entity, a new record in the "Object properties" table is also created in addition to the one in the "Properties" table. It is also linked back in the same fashion. The named entities should be added first, since their records in the database have to be referenced for the creation of other, more complex entities.

Direct entity relations. Operators for the description of direct relations between entities are: "SubClassOf", "EquivalentClasses", "DisjointClasses", "InverseObjectProperties", "SubDataPropertyOf", "SubObjectPropertyOf", "Equivalent-DataProperties", "EquivalentObjectProperties", "DisjointDataProperties", "Dis-jointObjectProperties", "HasKey", "SameIndividual", "DifferentIndividuals", "ClassAssertion". All these operators define a simple relation between two entities within the ontology. All of them correspond to records in the "Attributes" table. The attributes table links the referenced entities using the "Entity ID" and "Target ID" columns. The type of the relation is defined using the corresponding type from the attribute types. They all will have a zero value in the "Number" column. Some of the operators will use the same type of attribute. "SubClassOf" and "ClassAssertion" can both use the type "SubOf", since the exact type of the relationship can be inferred by the type of the related entities.

Property specific attributes. Keywords describing properties are: "InverseFunctionalObjectProperty", "ReflexiveObjectProperty", "IrreflexiveObjectProperty", "SymmetricObjectProperty", "AsymmetricObjectProperty", "TransitiveObjectProperty", "DataPropertyDomain", "ObjectPropertyDomain", "DataPropertyRange", "ObjectPropertyRange", "FunctionalDataProperty", "FunctionalObjectProperty". Statements using these operators describe aspects of properties. The last three operators are used to update information in the "Properties" table. The domain and range of the properties are defined using references to entities from the main entity table in the columns named "Domain" and "Range". The asserted functionality of the property is stored in the "Functional" column of the "Properties" table. All the other descriptions of an object property are stored in the "Object properties" table.

Property assertions for individuals. Assertions are defined using "DataPropertyAssertion", "ObjectPropertyAssertion", "NegativeDataPropertyAssertion" and "NegativeObjectPropertyAssertion". These assertions are related to individuals in the ontology. The table "Property assertions" holds the information related to both data and object properties of an individual. The column "Entity ID" in this table holds the reference to the individual who possesses these properties. The column "Positive" stores a Boolean value indicating whether the assertion is a normal positive assertion or a negative assertion. The column "Property ID" holds the reference of the type of property this assertion uses. The last column "Value ID" holds a reference to the entity, which represents the target object in the case of an object property assertion or the value in the case of a data property assertion. Values are also stored in the entity table using the type "Data" and linked to their XSD (XML Schema) type entity using a "SubOf" attribute.

Annotation assertions. Annotations are linked to any entity by using the "AnnotationAssertion" operator. There are two kinds of annotations in OWL2. First, inline annotations can be added to most entities during their declaration, alongside the description. Second, annotation can be added to entities by using the annotation assertion mechanism. In both cases, annotations are added to the database as annotation entities and linked to the entity, which is being annotated by using the "Property assertions" table. The column "Property ID" indicates the type of annotation.

Collection-based complex classes. These complex classes are defined using "DataIntersectionOf", "ObjectIntersectionOf", "DataUnionOf", "ObjectUnionOf", "DisjointUnion", "DataOneOf", "ObjectOneOf", "DataComplementOf", "ObjectComplementOf" and "ObjectInverseOf". All the complex classes defined by these operators can be represented in the database by creating a new entity with the related type. Members are added using the "attributes" table. For example, in case of data union a new entity is created in the "entities" table. This entity will have the ontology prefix, automatically generated name, since complex classes themselves are usually nameless, and the type "DataUnion". Once the core entity is created, additional records in the "Attributes" table are needed. In this case, these will be records of the type "Collection item", where "Entity ID" is a

reference to the union entity and “Target ID” will be references to the member entities mentioned in the operator parameters.

Definition of an object-property chain. Chains are defined using “ObjectPropertyChain”. Chains are very similar to complex classes representing collections. The largest difference to collections is the use of the “Number” column in the “Attributes” table. This is necessary for the specification of the order of the object properties in the chain. This would not be very important for chains where every object property is the same, and only the number of links in the chain is important. However, for more complex chains, consisting of different object-properties, the structure of the chain should be preserved.

Relation-based complex classes. Complex classes based on relationships to other entities using properties are defined by “DataSomeValuesFrom”, “ObjectSomeValuesFrom”, “DataAllValuesFrom”, “ObjectAllValuesFrom”, “DataHasValue”, “ObjectHasValue”, "ObjectHasSelf", “DataMinCardinality”, “ObjectMinCardinality”, “DataMaxCardinality”, “ObjectMaxCardinality”, “DataExact-Cardinality” and “ObjectExactCardinality”. Cardinality classes, as well as some other complex classes are stored in the database using the “Complex classes” table. The common characteristic of these concepts is that they are defined by their relations to other concepts. Therefore, the table holds references to the identifiers of both properties and target entities. Cardinality classes will additionally need a numeric indicator of how many relations are allowed. The “Target ID” column is optional to those complex classes because it allows for specifications of only properties.

Data type definitions. New data types are defined using “DatatypeRestriction” and “DatatypeDefinition”. In case of a simple new data type, which is based on one of the XSD data types, a new data type entity is created and by using a “Sub-Of” attribute, the relation to the basic type is established. However, in case of a more complex data type with restrictions, multiple attributes are needed to describe the restriction. Because of the complexity of describing a restriction, no separate tables were created for that purpose. A restriction entity is created, and using the relevant attributes from the “Attributes” table the restriction is described. Values are also stored as entities in the “Entities” table.

IV. Implementation of the Schema in Java with Derby

Both the described database schema and the translation rules have been implemented in a prototype. The prototype has been written in JAVA. It uses an embedded Derby database. At this point in time the prototype is capable of opening an OWL2 file, written in the functional syntax, and translating the contained knowledge into the database. Figure 3 shows a view of the database containing ontology entities.

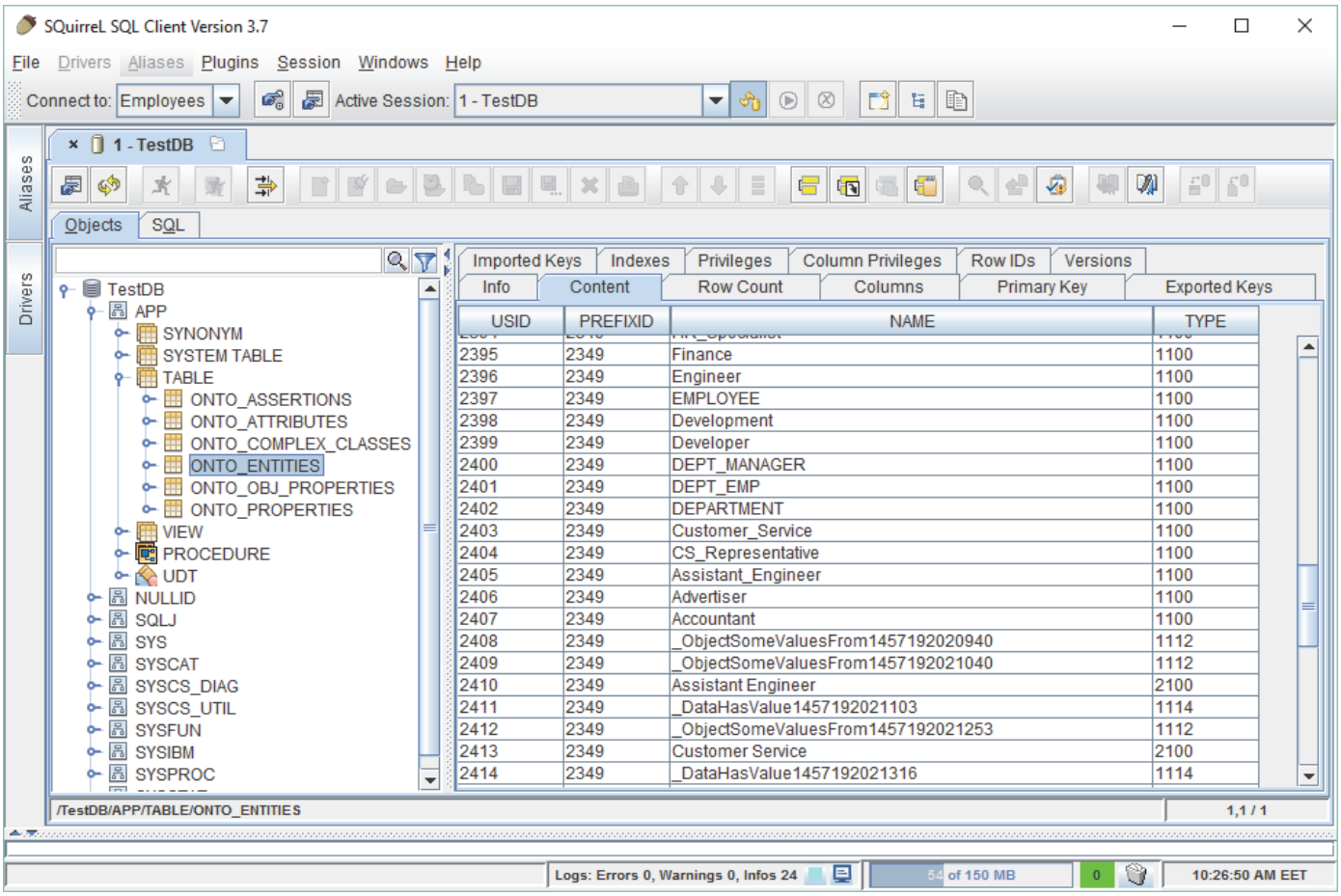

Fig. 3. Database view of the ontology data.

The current process involves creating an ontology using Protégé. Protégé is a very convenient tool for working with ontology. Once the ontology is defined, it is stored in an OWL file. This allows for further modification at a later point in time. The created file can be opened by the developed prototype. The prototype creates a model of the information and creates a list of unique entities. The entities are added to the database main entity table. Additional data are also added to the surrounding tables using references to the entities. Named entities are added first. Next, entities using named entities in their definitions are added. Entities using other complex entities in their definitions are added as soon as all the entities they depend on have been added first.

The information contained in the database can be accessed by the prototype. At this point in time, the prototype is capable of obtaining a list of all entities based on type. This is convenient when it is necessary to obtain concepts or other specific parts of the ontology. It is also possible to obtain entities by their name. Once, one or more entities are obtained, it is possible to systematically expand them. For example, based on the "SubOf" relationship stored in the "Attributes" table, it is possible to find the direct ancestors of a chosen entity. Since the ancestors are pointed to by their ID, it is possible to obtain their names. In those cases, when the names of the related entities are enough, no further steps should be taken. However, if any of the found entities are of further interest, additional information can be obtained as needed. The depth of the search and the completeness of any entity are optional.

The created prototype is also capable of using the ontology information, contained in the database for reasoning, and other related ontology tasks.

V. Conclusion

This paper described the proposed database schema for any ontology-based system requiring easy access to ontology knowledge. The proposed schema makes it possible to obtain needed pieces of the ontology, or the full ontology, from any point, capable of accessing a database. Because of the none-complicated nature of the schema, any solution can make use of the proposed method. This is possible because of the core entity list. Any software agent capable of accessing a database can use the central entity list as a dictionary. By querying the "Entities" table and providing a type value as a filter, anyone can obtain a list of names relevant to the described domain. No additional ontology knowledge or reasoning capabilities are needed for this simple task. A more complex tool can obtain more than just the names and use the ontology to its fullest extent.

It should be noted that there are some downsides. The proposed schema is purely for storage and does not provide any reasoning or integrity control. It is the knowledge engineer duty to make sure that the knowledge is correct and complete. However, since the proposed schema is part of a larger system, these tasks will be part of that system and can be done automatically. Another particularity is that primitive data are stored in text form. Numeric data mentioned in the ontology will be translated into data entities. The data themselves are stored in the entities name. This leads to the necessity of parsing text into numeric values before they can be used for numeric operations. Additional actions may have to be performed for other types of primitive data as well. Some minor loss of information, related to annotations, does occur. For example, there is no way to distinguish between inline comments and annotation assertions of comments. In OWL2 it is possible to annotate a direct relation ("EqualTo", "SubOf", etc). In the schema only annotation of entities is possible at this time.

The described database schema and method for converting OWL2 files are sufficient to store all the knowledge described in the file with almost no loss of information.

The discussed schema is simple to implement in any relational database. It is possible to add it to an existing database or to create a new dedicated database specifically for the ontology.

Acknowledgment

The present research has been supported by Doctoral Studies Grant No. 3412000-DOK.DITF awarded by the Faculty of Computer Science and Information Technology (Riga Technical University).

References

[1] H. Gorskis and A. Borisov, "Storing an OWL 2 Ontology in a Relational Database Structure," in *Environment. Technology. Resources. Proceedings of the International Scientific and Practical Conference*, vol. 3, 2015, pp. 71–75. https://doi.org/10.17770/etr2015vol3.168

[2] The Apache Jena homepage. [Online]. Available: https://jena.apache.org Accessed: March 20, 2016.

[3] M. Sir, Z. Bradac and P. Fiedler, "Ontology versus Database," in *IFAC-PapersOnLine*, vol. 48, issue 4, pp. 220–225, 2015. https://doi.org/10.1016/j.ifacol.2015.07.036

[4] A. Gali, C. X. Chen, K. T. Claypool and R. Uceda-Sosa, "From ontology to relational databases," in *Conceptual Modeling for Advanced Application Domains: ER Workshops 2004* (Lecture Notes in Computer Science), S. Wang et al. Eds., Berlin, Germany: Springer, vol. 3289, 2004, pp. 278–289. https://doi.org/10.1007/978-3-540-30466-1_26

[5] L. T. T. Ho, C. P. T. Tran and Q. Hoang, "An Approach of Transforming Ontologies into Relational Databases," in *Intelligent Information and Database Systems: 7th Asian Conference, ACIIDS* (Lecture Notes in Computer Science), 2015, pp. 149–158. https://doi.org/10.1007/978-3-319-15702-3_15

[6] I. Astrova, N. Korda and A. Kalja, "Storing OWL ontologies in SQL relational databases," *International Journal of Electrical, Computer and Systems Engineering*, vol. 1, no. 5, 2007, pp. 242–247.

[7] I. Astrova, A. Kalja, E. Jaeger, M. Jones, B. Ludascher and S. Mock, "Storing owl ontologies in sql3 object-relational databases," in *AIC'08 – Proceedings of the 8th Conference on Applied Informatics and Communications*, Rhodes, Greece, 2008, pp. 99–103.

[8] F. Zhang, Z. M. Ma and W. Li, "Storing OWL ontologies in object-oriented databases," *Knowledge-Based Systems*, vol. 76, pp. 240–255, March 2015. https://doi.org/10.1016/j.knosys.2014.12.020

[9] E. Vysniauskas and L. Nemuraite, "Mapping of OWL ontology concepts to RDB schemas," in *Information Technologies' 2009: Proceedings of the 15th International Conference on Information and Software Technologies*, 2009, pp. 317–327.

[10] E. Vysniauskas and L. Nemuraite, "Transforming ontology representation from OWL to relational database," *Information Technology and Control*, vol. 35, no, 3, pp. 333–343, 2015.

[11] G. Bumans, "Mapping between Relational Databases and OWL Ontologies: An Example. Scientific Papers," *Computer Science and Information Technologies*, vol. 756, pp. 99–117, 2010.

[12] B. Motik, I. Horrocks and U. Sattler, "Bridging the gap between OWL and relational databases," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, issue 2, pp. 74–89, Apr. 2009. https://doi.org/10.1016/j.websem.2009.02.001

[13] OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). [Online]. Available: https://www.w3.org/TR/owl2-syntax Accessed: March 20, 2016.
[14] OWL 2 Web Ontology Language Primer (Second Edition). [Online]. Available: https://www.w3.org/TR/owl2-primer Accessed: March 20, 2016.

Henrihs Gorskis is a fourth-year Doctoral student majoring in Information Technology at Riga Technical University (RTU). He received his Mg. sc. ing. degree in 2013. He is currently employed as a Researcher at RTU. His research interests include data mining, ontology engineering, ontology-based database access and evolutionary computing and programming. He is especially fond of the Java programming language and uses it for both work and personal application development.
E-mail: henrihs.gorskis@rtu.lv