Tools for Mapping Ontologies to Relational Databases: A Comparative Evaluation

Dorin Moldovan, Marcel Antal, Dan Valea, Claudia Pop, Tudor Cioara, Ionut Anghel, Ioan Salomie
Computer Science Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
{dorin.moldovan, marcel.antal, dan.valea, claudia.pop, tudor.cioara, ionut.anghel, ioan.salomie}@cs.utcluj.ro

Abstract —**This paper presents an analysis of the state of the art solutions for mapping a relational database and an ontology by adding reasoning capabilities and offering the possibility to query the inferred information. We analyzed four approaches: Jena with D2RQ, Jena with R2RML, KAON2 and OWL API. In order to highlight the differences between the four approaches, we used a nutrition diagnostics related ontology for the definition of the concepts and of the rules, and a relational database for the storage of the individuals. As performance evaluation, we focused on the time required to map the relational database to the ontology, and the time required to retrieve the information that is inferred about the diagnostics of a number of people. The obtained results show that the best performance in both cases is given by KAON2.**

Keywords—ontologies; relational database; ontology mapping; semantic reasoning; query.

I. INTRODUCTION

One of the main issues when dealing with a huge amount of data, like in the case of information related to nutrition problems and diets is represented by the way in which the data is stored. Different formats such as relational database records, xml files or text files may be used in order to save this information. Even though the databases provide great scalability and are used widely, there are some limitations: they store only data that is explicitly known, the performance of the system is reduced significantly if the number of tables is very large, and the extraction of meaning from data is slow. There is a need to use a common language in order to access this data that comes from different sources. Also, the solution should provide scalability and the syntax should be as close as possible to the human language.

Ontologies are a way to represent data about a specific domain in a semantical and hierarchical manner [1]. They provide ways to specify properties for the objects involved in the domain, and the relations among these objects. Ontologies are seen both as vocabulary, because they provide the means to represent the data associated to a specific domain, and as content theory, because they identify the classes of objects and the relations among these objects. Data represented by the ontologies can be split in two, as it is shown in [2,20]: TBox and ABox, based on the fact if the knowledge changes in time or not. The intentional knowledge (TBox) represents knowledge that is usually thought not to change, while the extensional knowledge (ABox) is subject to occasional or constant change. The TBox, also known as "terminological component", represents the conceptualization that is associated with a set of facts. On the other hand, the ABox, also known as "assertional component", represents the facts associated with the TBox. The TBox contains information such as: definitions of concepts and of properties, declaration of roles or concept axioms, classification, and so on. The ABox represents assertions such as: assertions of membership to concepts or roles, attributes assertions and linkages assertions. Their main disadvantage is their lack of scalability when dealing with queries involving large amounts of data.

To solve the above data representation and management issues an approach to store information from a specific field of knowledge is represented by ontologies, while in many cases the information is stored in a relational database. A representation of data which uses both an ontology and a relational database presents as advantages the fact that reasoning capabilities may be used, in order to infer new information from information which is already represented in the ontology, while new information can be inserted easily using tools that are not specific to ontologies such as Hibernate [16] or Java Persistence API (JPA) [17]. The importance of using both an ontology and a database is represented by the advantages that both representations bring into the mix. On one hand, the ontologies may be used to represent knowledge associated to a specific domain semantically, providing reasoning capabilities and possibilities to query the information using a language that can be understood more easily than SQL while, on the other hand, relational databases present features such as scalability, the fact that they can be accessed using different tools, and better security.

The purpose of the paper is to present and to compare four methods of realizing a mapping between a database and an ontology: Jena [7] and D2RQ [4], Jena and R2RML [9,10], KAON2 [11,12,21], and finally OWL API [13,14]. The comparison of these four approaches is not an ideal comparison because in the case of KAON2 the ontology is not defined in a specialized tool such as Protégé, as it is the case of the other three approaches. Also, for the fourth approach, we do not use a mapping file, and the data is not retrieved using SPARQL [3,6]. Even though the four approaches differ in some aspects, one common point is represented by the reasoning capabilities. In all of the four cases, new data may be produced from data that already exists, by using reasoning rules. In all of the four cases we assumed that the database is the same, and each

978-1-4673-8200-7/15/$31.00 ©2015 IEEE

approach used the same set of reasoning rules. Regarding the results, we expected to get the same functionality related results, but with varying different ontology loading and information retrieval times. The obtained results show that KAON2 has the best performance both for the loading and for the retrieval of the information, while the OWL API has the worst retrieval time. The reason why the retrieval time is significantly greater in the case of the OWL API than in the other three cases is because the data is not retrieved using SPARQL. We also concluded that in the case of using Jena and D2RQ the loading time is smaller than in the case of using Jena and R2RML, while the retrieval time is greater when using the first approach.

The rest of the paper is organized as follows: Section II presents an overview of the ontology mapping approaches, Section III presents an evaluation scenario, Section IV shows the results of the evaluation, and finally Section V presents the conclusions.

II. ONTOLOGY MAPPING APPROACHES

In this section we present four different approaches to map a relational database to an ontology revealing the most important technical aspects involved.

A. Jena and D2RQ

D2RQ is a tool that is used to transform the content which is stored in a relational database into a read-only RDF [18] graph. D2RQ offers the possibility to map the tables to ontology classes, the rows from the tables to ontology individuals, the columns to datatype properties, and the foreign keys to object properties. Jena is a tool that is implemented in the Java programming language. It can be used for the translation of the constructs and the statements of the semantic web into Java classes, objects, attributes and methods. Among the artifacts used by Jena, the following ones are the most important: subjects, predicates, objects, statements, data, queries and results, reasoners, and rules. The queries are written in SPARQL, which is the W3C recommendation query language for RDF. Jena allows multiple types of reasoners: internal or external. Also it provides support for SWRL rules. The rules can be written in Jena or they can come integrated with the ontology.

An architecture that uses D2RQ (a system which may be used to access relational databases as virtual, read-only RDF graphs) and Jena (a Java API for Ontology Management) is discussed in [8]. The information is stored in a relational database, and a mapping file is used to translate the information that is stored in the database into a read-only RDF (Resource Description Framework) graph. RDF is too fine-grained and irrestrictive to describe complex ontologies, and thus something more complex was developed in order to deal with this problem. The solution is represented by the OWL (Web Ontology Language). Fig. 1 describes an architecture that uses D2RQ as a mapping tool, and Jena as a tool to handle the content of the ontology.

The architecture from Fig. 1 explains briefly how to connect a relational database to an ontology [5]. Data is retrieved from a relational database using a mapping file, with the extension *.ttl*. After this operation, a D2RQ data model is obtained. This model will contain the individuals together with their associated properties that will populate the ontology.

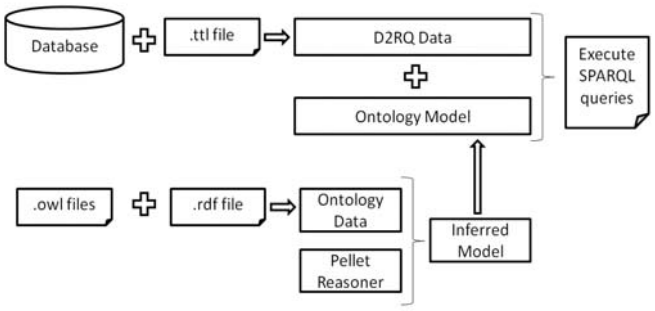

Fig. 1. An architecture that uses D2RQ and Jena (adapted from [5])

The ontology is defined in a tool, such as Protégé, and is saved under the extension *.owl*. In the case the ontology imports other ontologies, an *.rdf* file that handles these imports is defined. Information retrieved from these two files will represent the Ontology Data (the concepts from the ontology) and, by using a reasoner such as Pellet, an inferred model can be obtained. Finally, from this inferred model, and ontology model is created. The ontology model will represent the *TBox* component of the ontology, while the D2RQ data will represent the *ABox* component of the ontology. Adding the D2RQ data to the ontology model, allows us to query the information from the ontology by using SPARQL queries. The queries written in SPARQL use concepts defined in the ontology, and are easy to understand by the users of the application.

B. Jena and R2RML

Another approach that can be used to map a relational database to an ontology is described in [9]. This approach uses R2RML (RDB to RDF Mapping Language). More information about the syntax of R2RML can be found in [10]. As its name suggests, R2RML is a language that provides the ability to view a relational database in the RDF data model. The input for a R2RML mapping is an existing database that contains the information which will populate the RDF graph. Among the features provided by R2RML are the following ones: it can map tables from the database, it can compute a property with an R2RML view, it can link two tables, and it can translate database type codes to IRIs, and so on.

When comparing R2RML with D2RQ one can observe the fact that there are a lot of similarities between them. In fact the same logic is applied when the mapping between the relational database to the ontology data is performed: the tables from the relational database are mapped to classes from the ontology, the rows from the tables are mapped to ontology individuals, the columns that are not foreign keys are mapped to datatype properties, and also there is the possibility to map a foreign key to an object property. An R2RML mapping file also offers the possibility to write an SQL query which will produce a view of a table, and the table that corresponds to the obtained view is mapped further to the ontology concepts. Fig. 2 presents the architecture for data integration using R2RML and Jena.

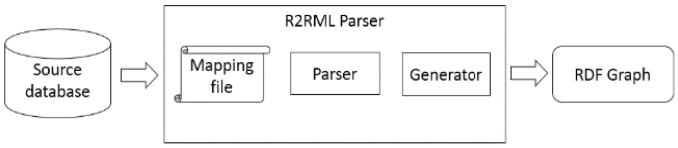

Fig. 2. An architecture that uses R2RML and Jena (adapted from [9])

Data is extracted from a relational database and the output is represented by an RDF graph. In order to obtain the RDF graph, the following steps are taken: the first step is to convert the contents from the database into result sets, the second step is performed by the Parser and it consists in generating of a set of instructions according to the mapping file, and finally the generator instantiates an RDF graph according to these instructions.

C. KAON2

KAON2 can be used to manipulate OWL-DL ontologies, and the queries can be formulated in SPARQL, even though not the entire SPARQL specification is supported. Also, KAON2 has some limitations, such as it cannot handle nominals (also known as enumerated classes), and it cannot handle very large numbers in statements of cardinality. KAON2 distribution contains 10 very well documented examples that show how to load an ontology and print its contents, how to create an ontology, how to add statements to it, how to get axioms from an ontology, how to add rules to an ontology, and how to map a relational database to an ontology. A detailed architecture of KAON2 can be seen in [12], and is presented in Fig. 3.

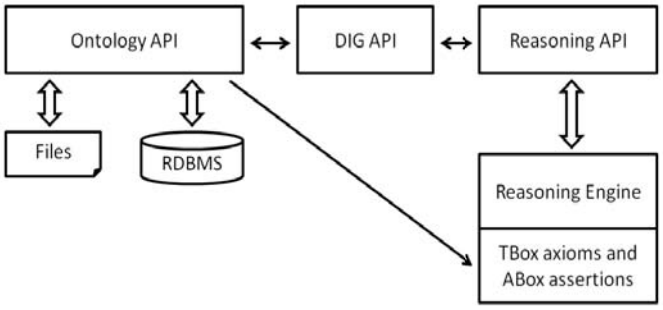

Fig. 3. KAON2 Architecture (adapted from [12])

The *Ontology API* is used for the manipulation of the ontology (loaded as OWL RDF or OWL XML file), thus for the addition and the removal of axioms. Assertions about the individuals, ABox assertions, can be stored in a relational database (RDBMS). KAON2 provides the possibility to map the entities of the ontology to database tables, and also KAON2 can query the database on the fly during reasoning. The *Reasoning API* allows the invocation of a variety of reasoning tasks, and the retrieval of the results from the *Reasoning Engine*. The APIs can be also invoked remotely through the *DL Implementors Group (DIG)* API.

D. OWL API

The OWL API is an API (Application Programming Interface) that can be used for working with the OWL ontologies. One of the main features of the OWL API is represented by the fact that it can load and save ontologies in a variety of formats. Also the reasoning functionality is treated separately. The OWL API provides a uniform view of the ontology and data structures that represent OWL ontologies. Also, it offers functionalities such as: creation, manipulation, parsing, rendering and reasoning.

Next we will discuss how we can populate the ontology with information that comes from a relational database. Data from a relational database can be used to populate an ontology by using *sql* queries [15]. The following rules are described in the paper: mapping tables, mapping columns, mapping data types, mapping constraints and mapping rows. The OWL-DL represents a sublanguage of the OWL which imposes a set of constraints on the use of the constructs which are specific to the OWL language. A table is mapped to a class in the case its columns are not foreign keys to other tables. If a column is not a foreign key, then it is mapped to a data type property. Mapping data types refers to the mapping between data types from SQL to data types from XSD. We will not insist on this kind of mapping. Also, the paper presents how the mapping of the constraints is realized. In this case, we are interested in the mapping of the foreign keys. Finally, the rows are mapped to instances. Fig. 4 shows how the information can be retrieved using OWL API and a relational database.

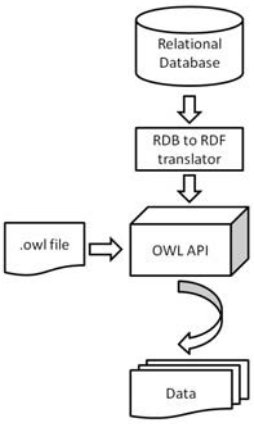

Fig. 4. An architecture that uses OWL API

The ontology that contains the concepts is loaded from an *.owl* file. The data that is used to populate the ontology is stored in a relational database. The module *RDB to RDF translator* is used to get the data from the database and to convert it into ontology information. After the ontology is populated with the data that comes from the relational database, the obtained model can be used in association with a reasoner that must be asked directly what individuals to return from the ontology.

III. USE CASE EVALUATION

In this section we will describe how the mapping between instances and concepts is performed by using the four methods described before, targeting their comparative evaluation. The ontology will contain reasoning rules written in SWRL, which means that in each of the four cases proposed by us, a certain amount of time will be allocated to the process of reasoning. We are interested in the time to load the instances from the database in the ontology, and in the time to retrieve information from the ontology using SPARQL queries, while reasoning is involved.

As use case ontology we have used the one described in Fig. 5. The ontology contains information about people (users), values (weight and height) and diagnostics (underweight, normal weight, overweight, and obese). The ontology contains three classes: *User*, *Diagnostic* and *Values*. A user can have the following data type properties: user id, first name, last name, gender and age. Users are also associated to a specific diagnostic by using the object property *hasDiagnostic*. Values are related to a user by using the object property *hasUser* which has the domain *Values* and the range *User*. The class *Values* has the following data type properties: id, weight, height, and BMI (body mass index). The ontology also contains four individuals that are instances of the class *Diagnostic.* These individuals are: *Underweight*, *NormalWeight*, *Overweight*, and *Obese*. The structure of the ontology can be seen below.

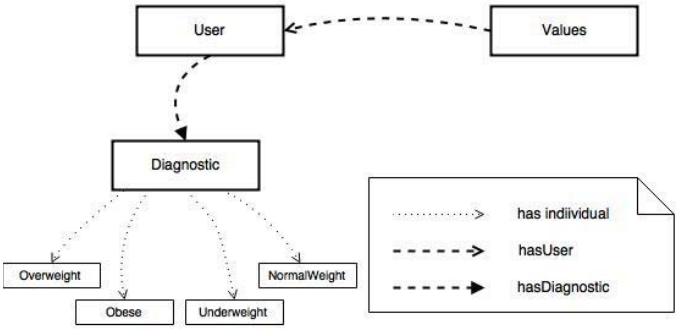

Fig. 5. Ontology's structure

The ontology has reasoning rules for the determination of the following values: *BMI*, *Underweight*, *NormalWeight*, *Overweight* and *Obese*.

$$BMI = \frac{Weight\ (in\ Kg) \times 10000}{Height^2\ (in\ cm)} \quad (1)$$

$$BMI < 18.5 \Rightarrow Underweight \quad (2)$$

$$18.5 \leq BMI < 25 \Rightarrow NormalWeight \quad (3)$$

$$25 \leq BMI < 30 \Rightarrow Overweight \quad (4)$$

$$30 \leq BMI \Rightarrow Obese \quad (5)$$

The first equation describes how the value of the body mass index (BMI) is computed. As we can see in (1), the value of BMI equals the fraction between the weight (expressed in kilograms) multiplied by 10000 and the square of height (expressed in centimeters). If the BMI is less than 18.5, then the person is classified as underweight (2). If the BMI is greater than or equal with 18.5 and less than 25, then the person has normal weight. If the BMI is greater than or equal with 25 and less than 30 then the person is overweight, and finally, if the value of the BMI is greater than or equal with 30 then the person is obese.

The database model used for ontology mapping is composed from two tables that correspond to the ontology classes: *User* and *Values*. The table *users* has the following properties associated with a user: first name, last name, gender, and age. The table *values* contains information about the weight and the height of the users. The structure of the database can be seen in Fig. 6.

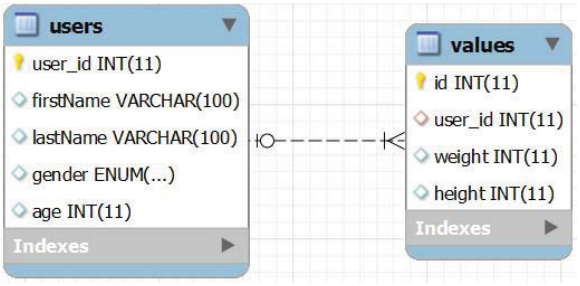

Fig. 6. Database model

A. Jena and D2RQ evaluation

The mapping between the relational database and the ontology is done using a mapping file that has the extension *.ttl*. The scope of this file is to specify how the mapping between the database and the ontology is performed. The file starts with a declaration of the namespaces. Some of these namespaces are provided below, in Fig. 7.

```
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix elders:
   <http://www.semanticweb.org/ontologies/2015/2/elders#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .
```

Fig. 7. D2RQ namespaces

After the namespaces used by the ontology are specified, it is necessary to show how the connection to the database is realized. The user has to specify the address at which the database can be found, the name of the driver which is used to establish the connection, the username, and the password. An example which illustrates how the connection is established can be seen in Fig. 8.

```
map:database a d2rq:Database;
   d2rq:jdbcDSN "jdbc:mysql://localhost/elders";
   d2rq:jdbcDriver "com.mysql.jdbc.Driver";
   d2rq:username "root";
   d2rq:password "admin";
   jdbc:autoReconnect "true";
   jdbc:zeroDateTimeBehavior "convertToNull"; .
```

Fig. 8. D2RQ Database Connection

A table from the relational database corresponds to a class from the ontology. The columns that contain primitive data types correspond to datatype properties, while the foreign keys correspond to object properties. Fig. 9. shows how to map a data property using D2RQ.

```
# Data Property age
map:age a d2rq:PropertyBridge;
   d2rq:belongsToClassMap map:users;
   d2rq:property elders:hasAge;
   d2rq:column "users.age";
   d2rq:datatype xsd:int; .
```

Fig. 9. Map a dataproperty using D2RQ

Reasoning rules can be written directly in Protégé, or they can be loaded from an external file. We will study the behavior of the reasoner Pellet. The rules are written directly in Protégé in SWRL syntax as shown in Fig. 10.

```
Values(?x), hasHeight(?x, ?h), hasWeight(?x, ?w), divide(?r, ?w, ?n),
   multiply(?n, ?h, ?h), multiply(?rez, ?r, 10000) -> hasBMI(?x, ?rez)
User(?x), Values(?y), hasUser(?y, ?x), hasBMI(?y, ?b),
   greaterThanOrEqual(?b, 30.0) -> hasDiagnostic(?x, Obese)
User(?x), Values(?y), hasUser(?y, ?x), hasBMI(?y, ?b),
   greaterThanOrEqual(?b, 25.0), lessThan(?b, 30.0) -> hasDiagnostic(?x,
   Overweight)
User(?x), Values(?y), hasUser(?y, ?x), hasBMI(?y, ?b),
   greaterThanOrEqual(?b, 18.5), lessThan(?b, 25.0) -> hasDiagnostic(?x,
   NormalWeight)
User(?x), Values(?y), hasUser(?y, ?x), hasBMI(?y, ?b), lessThan(?b, 18.5)
   -> hasDiagnostic(?x, Underweight)
```

Fig. 10. Rules written in Protégé

The ontology is queried using SPARQL. The first step when executing a SPARQL query is to connect the ontology model with the ontology data. The ontology model includes the concepts that are used by the ontology such as classes, definitions of object properties and datatype properties, while the ontology data contains the individuals that will populate the ontology. After the connection between these two models is established, queries written in a language similar with the language of the ontology can be written. Our scenario aims to write a SPARQL query that returns results that are inferred by the reasoner and which are not explicitly included in the database. An illustrative example (see Fig. 11) is a query that returns all the users and their diagnostics. Each user can have one of the following four diagnostics: *{Underweight, NormalWeight, Overweight, Obese}*.

```
SELECT ?firstName ?lastName ?diagnostic
   WHERE { ?user rdf:type elders:User . ?user elders:hasFirstName
      ?firstName . ?user elders:hasLastName ?lastName . ?user
      elders:hasDiagnostic ?diagnostic  }
```

Fig. 11. SPARQL query for the retrieval of the diagnostics

The ontology can be used with different types of reasoners. The scenario proposed by us studies the behavior of the Pellet reasoner (a complete OWL-DL reasoner).

B. Jena and R2RML evaluation

In the case of R2RML, the mapping between the database and the ontology is realized by using a file that has the extension *.ttl*. As in the case of D2RQ, the file starts with a declaration of the prefixes of the ontology. The database connection properties are not written explicitly in this file, as in the case of D2RQ. A feature of R2RML is the fact that it allows to create table views in the mapping file as in Fig. 12.

```
<EldersTableView> rr:sqlQuery """
   SELECT CONCAT('User_', user_id) AS userId, user_id, firstName
      , lastName, gender, age FROM elders.USERS; """ .
```

Fig. 12. R2RML Table View

R2RML can also be used to map object properties, as we can see in Fig. 13. In this case the join condition must be specified, in the form of an equality condition between a column from the child table and a column from the parent table. The reasoning rules and the querying of the ontology are similar with the ones that we used for the evaluation of Jena and D2RQ.

```
rr:predicateObjectMap [ rr:predicate      elders:hasUser ;
   rr:objectMap    [ a rr:RefObjectMap ;
      rr:parentTriplesMap  <TriplesMap1>;
      rr:joinCondition [ rr:child "user_id"; rr:parent "user_id"; ]; ]; ]
```

Fig. 13. Map an object property using R2RML

C. KAON2 evaluation

In the case of KAON2, the mapping between the database and the ontology is performed by using an *.xml* file. The purpose of this file is to specify how the concepts from the database (tables, columns, rows) are translated into ontology concepts. The first step is to specify the prefixes of the ontology as in Fig. 14.

```
<db:DBOntology
   db:name="http://www.semanticweb.org/ontologies/2015/2/elders"
   xmlns:db="http://kaon2.semanticweb.org/db#">
```

Fig. 14. KAON2 namespaces

The next step is to specify how the connection to the database is achieved. The mapping file should contain information about the address at which the database may be found, the credentials required to access that location, and the specific driver that is used for the connection to the database (see Fig. 15).

```
<db:Database db:connectionString="jdbc:mysql://localhost/elders"
   db:userName="root" db:password="admin"
   db:driverClassName="com.mysql.jdbc.Driver"/>
```

Fig. 15. KAON2 Database Connection

The tables are mapped to ontology classes, the columns that have primitive data types are mapped to datatype properties, while the foreign keys that specify relations between tables are mapped to object properties. Fig. 16 shows how to map a table to an ontology class.

```
<db:OWLClass db:name
   ="http://www.semanticweb.org/ontologies/2015/2/elders#User">
<db:Table db:tableName ="users">
<db:IndividualInteger db:fieldname="user_id" db:uriPrefix
   ="http://www.semanticweb.org/ontologies/2015/2/elders#User_"
   db:primaryKey ="true"/> </db:Table>
</db:OWLClass>
```

Fig. 16. Map a class using KAON2

Reasoning rules are written in the code, and added to the ontology. The reasoning rules can use standard arithmetic operators such as: +, -, *, /, %, and so on. Fig. 17 shows the implementation of rule (3).

```
Rule rule_normalWeight = KAON2Manager.factory().rule(
   KAON2Manager.factory().literal(true, hasDiagnostic,
   new Term[] { U, normalWeight }), new Literal[] {
   hasUser_M_U, KAON2Manager.factory().literal(true, bmi,
      new Term[] { M, BMI }), KAON2Manager.factory().literal(
      true, KAON2Manager.factory().ifTrue(2), new Term[] {
         KAON2Manager.factory().constant("$1 >= 18.5"), BMI }),
         KAON2Manager.factory().literal(true,
         KAON2Manager.factory().ifTrue(2),
         new Term[] {
            KAON2Manager.factory().constant(
            "$1 < 25.0"), BMI }) });
```

Fig. 17. KAON2 reasoning rule

This rule states that a person that has the BMI (Body Mass Index) in the range [18.5, 25.0) has as diagnostic *NormalWeight*. The ontology that is obtained from the mapping to the database does not allow update facilities, and thus a new ontology is created. The new ontology is the one obtained from the database and it will also contain the new rules and individuals that are inserted after the mapping to the database is achieved.

For querying the ontology our scenario uses a query written in SPARQL, Fig. 18. For each user, this query should return the first name, the last name and the diagnostic. The diagnostic has a value from the set: *{Underweight, NormalWeight, Overweight, Obese}*.

```
SELECT ?y ?z ?d WHERE {?x rdf:type
   <http://www.semanticweb.org/ontologies/2015/2/elders#User> ;
   <http://www.semanticweb.org/ontologies/2015/2/elders#hasFirstName>
      ?y ;
   <http://www.semanticweb.org/ontologies/2015/2/elders#hasLastName>
      ?z . ?x
<http://www.semanticweb.org/ontologies/2015/2/elders#hasDiagnostic>
   ?d . }
```

Fig. 18. SPARQL query for the retrieval of the diagnostics

D. MySQL and OWL API evaluation

In the case of OWL API we used another approach to map the database to the ontology. Instead of using a mapping file which specifies how the concepts from the database are related to the concepts from the ontology, we chose a solution which retrieves the information from the database and inserts the data in the ontology. The first step in this approach is represented by the retrieval of the information from the database. This process can be performed by using *sql* queries. The next step is represented by the insertion of the information retrieved from the database in the ontology. This step will be performed by retrieving data from the database using *sql* queries, and inserting the retrieved information in the ontology using specific methods which allow the insertion of individuals, datatype properties, object properties and so on. Each table from the database will correspond to an ontology class, the rows from the tables represent individuals that will populate the ontology, the columns that are not associated with foreign keys will represent datatype properties, and finally the foreign keys will represent object properties. The reasoning rules may come together with the ontology, when the ontology is defined in Protégé. For testing purposes we will use the reasoner Pellet.

To query the ontology we will call the reasoner directly in order to get the desired information. The reasoner will return all the individuals from the ontology that have the type *User*, and for each individual of this type, it will retrieve the first name, the last name, and the weight diagnostic. This approach does not use SPARQL as OWL API does not provide support for SPARQL. However, by using another framework, such as Jena, and inserting the model created in OWL API in a Jena model, the information can be retrieved using SPARQL.

IV. Evaluation Results

In order to compare the above presented alternatives we have considered 10 cases and varied the number of users that are stored in the database as {100, 200, …, 1000}. The users will be generated randomly in the database, together with the measurements associated with them. The height of the user will be a random number between 165 and 184, and the weight of the user will be a random number between 50 and 100. As output, we want to retrieve for each user the first name, the last name and the diagnostic. We aim to measure the time required to initialize the ontology with the users and their corresponding measurements that are taken from the relational database, and the time required to retrieve the information about the users.

The *time required to load the information* from the database to the ontology in all of the 4 cases is presented in Fig. 19 while the *time required to retrieve the information* from the ontology is shown in Fig. 20.

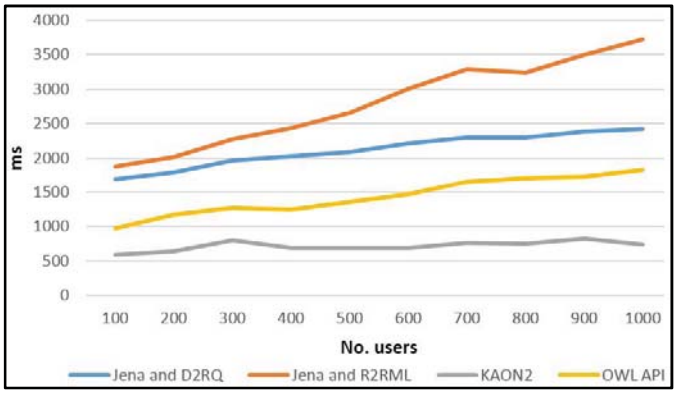

Fig. 19. Loading time in milliseconds (ms)

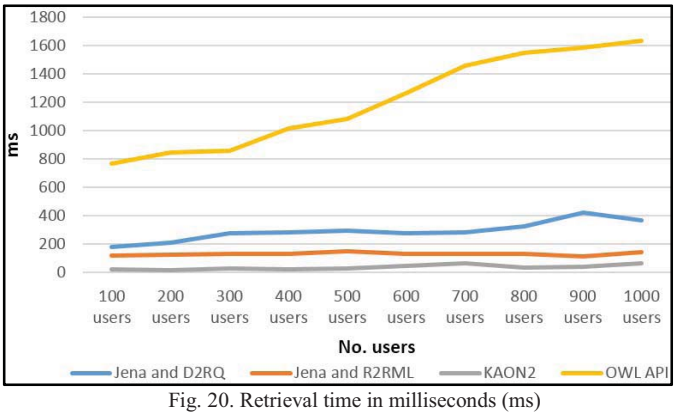

Fig. 20. Retrieval time in milliseconds (ms)

V. CONCLUSIONS

In this paper we presented different approaches for mapping the content stored in a relational database to ontology: Jena and D2RQ, Jena and R2RML, KAON2, and OWL API. We focused on three main aspects: how the mapping between the database and the ontology is performed, how the reasoning facilities are introduced and how data is retrieved. We concluded that the smallest loading time of the individuals from the relational data base in the ontology is obtained in the case of KAON2, followed by OWL API, and finally the approaches which use Jena. In the case of the retrieval of the information, the best performance is also given by KAON2, but the approaches that use Jena performed better than the approach that uses OWL API. This is justified by the fact that in the cases that use Jena (D2RQ and R2RML) SPARQL is used for retrieving data, while in the case of OWL API Pellet reasoner is used.

ACKNOWLEDGMENT

This work has been carried out in the context of the Ambient Assisted Living Joint Programme project DIET4Elders [19] and was supported by a grant of the Romanian National Authority for Scientific Research, CCCDI – UEFISCDI, project number AAL16/2013. This document is a collaborative effort. The scientific contribution of all authors is the same.

REFERENCES

[1] B. Chandrasekaran, J. R. Josephson, V. R. Benjamins, "What are Ontologies, and Why Do We Need Them?", IEEE Intelligent Systems and their Applications, vol. 14, pp. 20-26, January/February 1999

[2] M. Bergman, "The Fundamental Importance of Keeping an ABox and a TBox Split", AI3 Adaptive Information, Sunday 2009

[3] SPARQL, http://www.w3.org/TR/rdf-sparql-query/

[4] D2RQ, http://d2rq.org/

[5] M. G. Skjaeveland, "Tutorial on Semantic Technology at Semantic Days 2010", D2R, May 2010

[6] J. Perez, M. Arenas, C. Gutierrez, "Semantics and Complexity of SPARQL", ACM Transactions on Database Systems (TODS), vol. 34, no. 16, August 2009

[7] Apache Jena, https://jena.apache.org/

[8] C. Bizer, A. Seaborne, "D2RQ Treating Non-RDF Databases as Virtual RDF Graphs", 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan, November 2004

[9] N. Konstantinou, D. Kouis, N. Mitrou, "Incremental Export of Relational Database Contents into RDF Graphs", 4th International Conference on Web Intelligence, Mining and Semantics (WIMS'14), June 2014

[10] R2RML, http://www.w3.org/TR/r2rml/

[11] KAON2, http://kaon2.semanticweb.org/

[12] B. Motik, U. Sattler, "A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes", LPAR'06 Proceedings of the 13th international conference on Logic for Programming, Artificial Intelligence, and Reasoning, pp. 227-241, November 2006

[13] M. Horridge, S. Bechhofer, "The OWL API: A Java API for OWL Ontologies", Semantic Web, vol. 2, pp. 11 – 21, January 2001

[14] S. Bechhofer, N. Matentzoglu, "The OWL API: An Introduction", COMP60421: Ontology Engineering for the The Semantic Web, November 2014

[15] I. Astrova, N. Korda, A. Kalja, "Rule Based-Transformation of SQL Relational Databases to OWL Ontologies", In Proceedings of the 2nd International Conference on Metadata & Semantics Research, 2007

[16] Hibernate, http://hibernate.org/

[17] Java Persistence API, http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[18] RDF, http://www.w3.org/RDF/

[19] DIET4Elders AAL Project, www.diet4elders.eu

[20] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider, "The description logic handbook: theory, implementation, and applications", Cambridge University Press New York, NY, USA, 2003, ISBN:0-521-78176-0

[21] B. Motik, "KAON2 – Scalable Reasoning over Ontologies with Large Data Sets", ERCIM News 2008(72), 2008