International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:5, 2007

Storing OWL Ontologies in SQL Relational Databases

Irina Astrova, Nahum Korda, and Ahto Kalja

***Abstract*—**Relational databases are often used as a basis for persistent storage of ontologies to facilitate rapid operations such as search and retrieval, and to utilize the benefits of relational databases management systems such as transaction management, security and integrity control. On the other hand, there appear more and more OWL files that contain ontologies. Therefore, this paper proposes to extract ontologies from OWL files and then store them in relational databases. A prerequisite for this storing is transformation of ontologies to relational databases, which is the purpose of this paper.

***Keywords*—**Ontologies, relational databases, SQL, and OWL.

I. Introduction

THERE are two basic techniques for storing ontologies [1]. The first technique is to use file systems for storing ontologies in flat files. The main problem with this technique is that file systems do not provide scalability, sharability, or any query facility.

The second technique (that we follow) is to use database management systems for storing ontologies in databases. The main problem with this technique is that database management systems require that an ontology should have a fixed structure, which cannot be guaranteed as ontologies are often built in a distributed way. This means, for example, that one user may define an employee as having a social security number, but not foresee a martial status. This will not stop, however, another user from asserting that a given employee is married, adding a `martialStatus` data type property to an `Employee` class.

There are several options for storing ontologies in databases; e.g. relational, object or object-relational. Storing ontologies in relational databases is less straightforward than storing ontologies in object or object-relational databases, because relational database management systems do not support inheritance. However, relational database management systems have significant advantages over object or object-relational database management systems. In particular, relational database management systems provide maturity, performance, robustness, reliability, and availability.

Manuscript received on July 31, 2007. This work was supported in part by ESF (Estonian Science Foundation) under the grant nr. 5766.

Irina Astrova is with the Institute of Cybernetics, Tallinn University of Technology, Estonia (e-mail: irinaastrova@ yahoo.com).

Nahum Korda is with Technion, Israel Institute of Technology (e-mail: korda@technion.ac.il).

Ahto Kalja is with the Institute of Cybernetics, Tallinn University of Technology, Estonia (e-mail: ahto@cs.ioc.ee).

II. Motivation

There are three main reasons for storing ontologies in relational databases:

- **Legacy data:** When stored in relational databases, ontologies can interoperate with a large amount of data in existing relational databases.
- **Legacy applications:** When stored in relational databases, ontologies can be accessed from within existing relational database applications.
- **Large scale ontologies:** The ability of relational databases to store a large amount of data proves that the relational databases are also suitable for storing large scale ontologies that can contain millions of instances.

A prerequisite for this storing is transformation of ontologies to relational databases, which is the purpose of this paper.

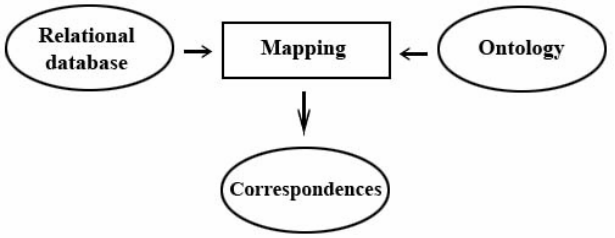
III. Transformation Problems

Transformation of ontologies to relational databases should handle the following problems:

- **Loss of data:** The result of the transformation should adequately describe the original data.
- **Structure loss:** In some cases, the transformation is not really lossless in the sense that not all constructs in an ontology can be mapped to a relational database. Therefore, the quality of the transformation should be analyzed.
- **Focus on structures**: Besides the mapping of structures, mechanisms should be provided for the mapping of data (i.e. instances).
- **Focus on data:** Data should be mapped, with incorporation of data types.
- **Applicability:** In some cases, the transformation is not really general in the sense that its application is rather restricted. E.g. if the transformation allows only for exotic ontologies, not being used in practical situations, then the transformation suffers from the applicability problem.
- **Correctness:** The transformation should have provable correctness.

IV. Related Work

A majority of the related work has been done in mapping between relational databases and ontologies; e.g. [2] – [5]. However, this mapping is quite different from transformation of relational databases to ontologies, as shown in Fig. 1.



(a) mapping between relational database and ontology

(b) transformation of relational database to ontology

Fig. 1 Mapping vs. transformation

The difference is that the mapping assumes the existence of both a relational database and an ontology, and produces a set of correspondences between the two. That is, the inputs to the mapping are both a relational database and an ontology, and the output is a set of correspondences that relate constructs of the relational database to those of the ontology. A construct in the relational database unrelated to any construct in the ontology is considered to be out of scope of the mapping. By contrast, the transformation assumes that only an ontology exists, whereas a relational database is produced from the ontology. That is, the input to the transformation is an ontology and the output is a relational database.

There are several approaches to transformation of ontologies to relational databases; e.g. [6] – [8]. However, all these approaches suffer from one or more of the following problems:

- They ignore restrictions that capture additional semantics.
- They are not implemented.
- They are semi-automatic (i.e. they can require much user interaction).
- They do not analyze structure loss caused by the transformation. Rather, they assume that all constructs of an ontology can be mapped to a relational database.

As an attempt to resolve these problems, we propose a novel approach to transformation of ontologies to relational databases, which is the main contribution of this paper. We assume that an ontology is written in OWL [9], the standard ontology language, and that a relational database is written in SQL [10], the standard relational database language.

V. Transformation

An ontology is considered to be an implementation of an ontological model. This model includes constructs for specifying classes, properties, data types, inheritance, restrictions, and other semantics, as shown in Fig. 2. However, the ontology does not need to include all constructs of the ontological model (i.e. it can use only a portion of the ontological model).

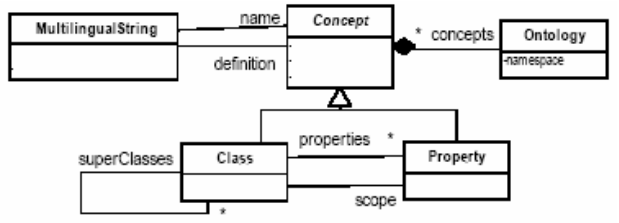


Fig. 2 Simplified ontological model

Similarly, a relational database is considered to be an implementation of a relational model. This model includes constructs for specifying tables, columns, data types, constraints, and other semantics, as shown in Fig. 3. However, the relational database does not need to include all constructs of the relational model (i.e. it can use only a portion of the relational model).

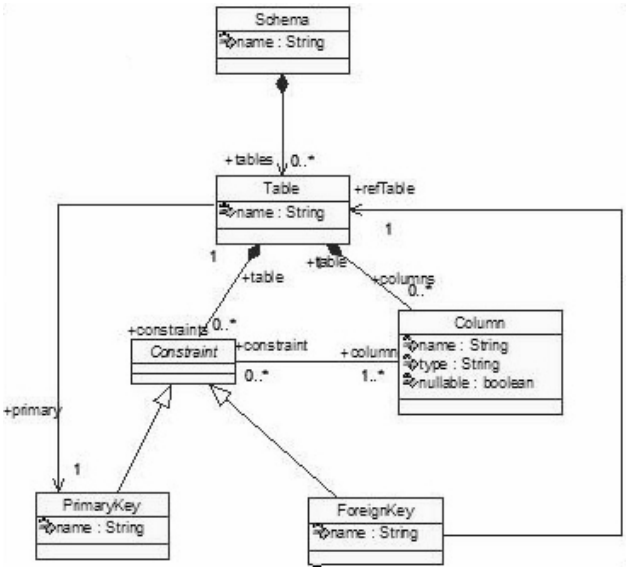


Fig. 3 Simplified relational model

Fig. 4 shows the basic idea behind our approach. Transformation of ontologies to relational databases is based on a set of rules called *mapping rules* that specify how to map constructs of the ontological model to the relational model. The mapping rules are then applied to an ontology (source) to produce a relational database (target). Since the mapping rules are specified on the model level, they are applicable to any

ontology that conforms to the ontological model.

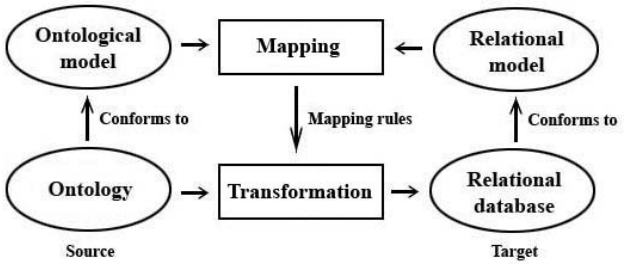


Fig. 4 Transformation of ontologies to relational databases

A. Mapping Rules

There are two types of properties that need to be considered: data type properties and object properties. In addition, properties can be single-valued or multivalued, required or optional; this has a great impact on the transformation.

If a property is *single-valued*, then it means that each instance in a class may have at most one value for the property. A single-valued property is identified in the following cases:

- Where a cardinality of the property has a (maximum) value of 1.
- Where the property is (inverse) functional.

In any other case, the property is *multivalued*.

If a property is *required*, then it means that each instance in a class must have at least one value for the property. A required property is identified in the following cases:

- Where a cardinality of the property has a (minimum) value greater than 0.
- Where the property is restricted to have some values from another class.
- Where the property is restricted to have a particular value.

In any other case, the property is *optional*.

Our approach maps constructs of an ontology to a relational database, applying the following rules:

Rule 1: A named class (including subclasses and association classes) maps to a table. This table is named with the name of the class. The table is assigned a primary key.

- A table that corresponds to an association class (i.e. the class that relates other classes) gets as its primary key a combination of foreign keys to all its related tables.
- A table that corresponds to a subclass gets as its primary key a foreign key to its "superclass" table.
- Any other table gets an "auto-number" primary key. This key is named with the name of the table suffixed with ID, such as `EmployeeID` for an `Employee` table.

Rule 2: If a data type property is single-valued, then it maps to a column in the table that corresponds to the class specified as the domain of the data type property. This column is named with the name of the data type property. The column uses as its type the type specified as the range of the data type property converted from XSD to SQL (see Section B).

Rule 3: If a data type property is multivalued, then it maps to a table. This table is named with the name of the data type property suffixed with Value, such as `hobbyValue` for a `hobby` data type property. The table gets as its primary key a combination of a corresponding column and a foreign key to the table that corresponds to the class specified as the domain of the data type property.

E.g. a `hobby` data type property in Fig. 5 is multivalued (i.e. an employee can have zero or more hobbies). Since SQL does not support multivalued columns, a `hobbyValue` table is created. This table gets as its primary key a combination of an `EmployeeID` column (that is a foreign key to an `Employee` table) and a `hobby` column. If the `hobby` data type property were single-valued, then the `hobbyValue` table would not be created but just the `hobby` column in the `Employee` table.

Rule 4: If an object property is both single-valued and optional, and there is a single-valued inverse of the object property (a one-to-zero-or-one relationship), then the inverse of the object property maps to a foreign key in the table that corresponds to the class specified as the range of the object property. This key references the primary key in the table that corresponds to the class specified as the domain of the object property. The name of the foreign key is the name of the inverse of the object property. (The object property does not map to any foreign key, because creating two foreign keys for the relationship would mean a circular dependency.)

Rule 5: If an object property is single-valued and Rule 4 is not applied (a zero-or-one-to-one, one-to-one or many-to-one relationship), then the object property maps to a foreign key in the table that corresponds to the class specified as the domain of the object property. This key references the primary key in the table that corresponds to the class specified as the range of the object property. The name of the foreign key is the name of the object property.

Rule 6: If an object property is multivalued and there is a single-valued inverse of the object property (a one-to-many relationship), then the inverse of the object property maps to a foreign key in the table that corresponds to the class specified as the range of the object property. This key references the primary key in the table that corresponds to the class specified as the domain of the object property. The name of the foreign key is the name of the inverse of the object property.

Rule 7: If an object property is multivalued and Rule 6 is not applied (a many-to-many relationship), then the object property maps to a table. This table is named with the name of the object property. The table gets as its primary key a combination of two foreign keys. One foreign key references the primary key in the table that corresponds to the class specified as the domain of the object property. Another foreign key references the primary key in the table that corresponds to the class specified as the range of the object property.

Rule 8: A value restriction on a data type property maps to a `CHECK` constraint on the corresponding column.

Rule 9: An inverse functional property maps to a `UNIQUE` constraint on the corresponding column.

Rule 10: A required property maps to a `NOT NULL` constraint on the corresponding column.

Rule 11: An enumerated data type maps to a `CHECK` constraint with enumeration.

Rule 12: An instance in a class maps to a row in a corresponding table.

In addition, to support multilingual ontologies, a `RDFSProperty` table is created for multilingual strings to store multilingual labels and comments of classes and properties.

B. Data Type Conversion

Most of the transformation of data type properties has to do with converting data types from XSD to SQL. Unlike SQL, OWL does not have any built-in data types. Instead, it uses XSD data types such as `string`, `integer`, `float`, `boolean`, `time` and `date`.

TABLE I
DATA TYPE CONVERSION

XSD data type	SQL data type
short	SMALLINT
unsignedShort	SMALLINT
integer	INTEGER
positiveInteger	INTEGER
negativeInteger	INTEGER
nonPositiveInteger	INTEGER
nonNegativeInteger	INTEGER
int	INTEGER
unsignedInt	INTEGER
long	INTEGER
unsignedLong	INTEGER
decimal	DECIMAL
float	FLOAT
double	DOUBLE PRECISION
string	CHARACTER VARYING
normalizedString	CHARACTER VARYING
token	CHARACTER VARYING
language	CHARACTER VARYING
NMTOKEN	CHARACTER VARYING
Name	CHARACTER VARYING
NCName	CHARACTER VARYING
time	TIME
date	DATE
datetime	TIMESTAMP
gYearMonth	DATE
gMonthDay	DATE
gDay	DATE
gMonth	DATE
boolean	BIT
byte	BIT VARYING
unsignedByte	BIT VARYING
hexBinary	CHARACTER VARYING
hexBinary	CHARACTER VARYING
anyURI	CHARACTER VARYING

Table I shows how to convert data types from XSD to SQL. This conversion is simple for the XSD data types that directly correspond to SQL data types. E.g. if XSD data type is `string`, then SQL data type is `CHARACTER VARYING`. However, the conversion becomes a challenge for "unsupported" data types. E.g. a `ssn` data type property in Fig. 5 uses `positiveInteger` as its range. However, there is no `positiveInteger` in SQL. Therefore, a `ssn` column uses `INTEGER` as its type, combined with a `CHECK` constraint: `CHECK (ssn > 0)`.

C. Example

To illustrate the transformation, Fig. 5 shows an ontology and a relational database that is produced from this ontology, applying the mapping rules.

```
<owl:Class rdf:ID="Employee"/>
<owl:Class rdf:ID="Project"/>
<rdf:ObjectProperty rdf:ID="involves">
  <rdfs:domain rdf:resource="#Project"/>
  <rdfs:range rdf:resource="#Employee"/>
</rdf:ObjectProperty>
<rdf:ObjectProperty rdf:ID="involvedIn">
  <owl:inverseOf rdf:resource="#involves"/>
</rdf:ObjectProperty>
<rdf:ObjectProperty rdf:ID="manages">
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:range rdf:resource="#Project"/>
</rdf:ObjectProperty>
<rdf:ObjectProperty rdf:ID="managedBy">
  <owl:inverseOf rdf:resource="#manages"/>
</rdf:ObjectProperty>
<owl:Class rdf:ID="Project">
  <rdfs:subClassOf>
    <owl:Restriction>
    <owl:onProperty
rdf:resource="#managedBy"/>
      <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:DatatypeProperty rdf:ID="ssn">
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:range
rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
<owl:InverseFunctionalProperty rdf:ID="ssn"/>
<owl:DatatypeProperty rdf:ID="hobby">
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:range rdf:resource="&xsd;sting"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="sex">
  <rdfs:domain rdf:resource="#Employee"/>
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf>
        <rdf:List>
          <rdf:first
rdf:datatype="&xsd;string">Male/>
          <rdf:rest>
            <rdf:List>
              <rdf:first
rdf:datatype="&xsd;string">Female/>
              <rdf:rest
rdf:resource="&rdf;nil"/>
            </rdf:List>
          </rdf:rest>
        </rdf:List>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
```

```
</owl:DatatypeProperty>
<owl:Class rdf:ID="SoftwareProject">
  <rdfs:subClassOf rdf:resource="#Project"/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="type">
  <rdfs:domain
rdf:resource="#SoftwareProject"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:Class rdf:ID="SoftwareProject">
  <rdfs:subClassOf>
    <owl:Restriction>
    <owl:onProperty rdf:resource="#type"/>
      <owl:hasValue rdf:resource=Software/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
                        ↓
CREATE TABLE Employee(
  EmployeeID INTEGER PRIMARY KEY,
  ssn INTEGER CHECK (ssn > 0) UNIQUE,
  sex VARCHAR CHECK IN ('Male', 'Female'))
CREATE TABLE Project(
  ProjectID INTEGER PRIMARY KEY,
  managedBy  INTEGER  REFERENCES  Employee  NOT
NULL)
CREATE TABLE involves(
  EmployeeID INTEGER REFERENCES Employee,
  ProjectID INTEGER REFERENCES Project,
  PRIMARY KEY(EmployeeID, ProjectID))
CREATE TABLE hobbyValue(
  hobby VARCHAR,
  EmployeeID INTEGER REFERENCES Employee,
  PRIMARY KEY (hobby, EmployeeID))
CREATE TABLE SoftwareProject(
   ProjectID  INTEGER  PRIMARY  KEY  REFERENCES
Project,
   type VARCHAR CHECK (type='Software'))
```

Fig. 5 Example of transformation of ontology to relational database

VI. Implementation

Our approach is implemented in a utility called QUALEG DB. This utility is capable of automatic transformation of an ontology (written in OWL) to a relational database (written in SQL).

As shown in Fig. 6, the utility is a transformation engine that parses an OWL file (that contains an ontology), performs consistency and error checks, and generates an SQL script. This script is then executed via an ODBC driver by a relational database management system to create a relational database.

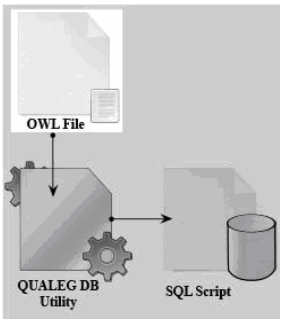


Fig. 6 Software architecture of QUALEG DB

The utility requires minimum user interaction. The only thing users need to do is to select or specify the name for an OWL file and the name for an SQL script, as shown in Fig. 7.

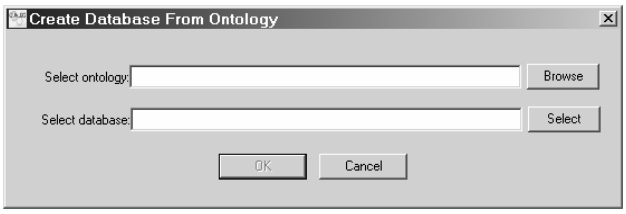


Fig. 7 Graphical user interface of QUALEG DB

When parsing an ontology, the utility checks the ontology to ensure that the ontology meets all requirements of the relational database management system and is consistent. This checking is important because it prevents certain kinds of errors in the resulting relational database. Examples of consistency and error checks include the following:

- Class and property names should not exceed 15 characters.
- Class and property names should not contain any other character except a letter, a digit and an underscore.
- Individuals in an enumerated class should be unique.
- Values in an enumerated data type should be unique.
- Both a domain and a range should be specified for a property unless the property is an inverse of an object property. (For the inverse of the object property, the domain and the range can be inferred from the object property.)

Violation of any of these checks will lead to errors. If the utility encounters any error during transformation, it will display the error to the user (as shown in Fig. 8) and continues the transformation unless the error is terminal. The "incorrect" construct that has caused the error will be excluded from the transformation.

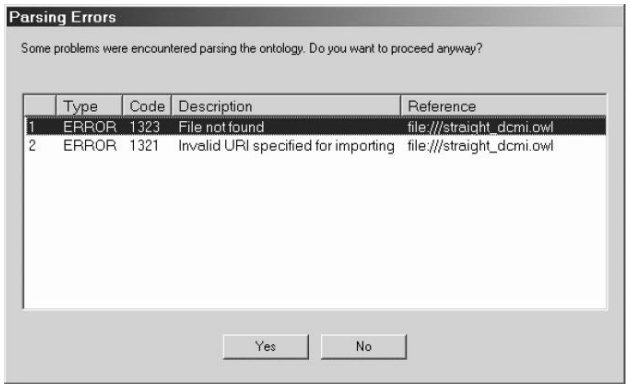


Fig. 8 Consistency and error checks

VII. Quality of Transformation

Since a relational model does not support all constructs of an ontological model, some of the constructs in an ontology will necessarily be lost when transforming the ontology to a relational database. Therefore, we need to analyze structure loss caused by this transformation. One way to do this is to retransform the resulting relational database to an ontology and see if the transformation is reversible. By reversible, we mean that transformation of an ontology to a relational database followed by reverse transformation of the resulting relational database to an ontology will yield the original ontology.

Let T_1 be transformation of an ontology O_1 to a relational database R. Let T_2 be reverse transformation of the relational database R to an ontology O_2. The transformation T_1 is said to be *reversible* if the ontology O_2 is equivalent to the ontology O_1. That is, $T_1(O_1) = R \wedge T_2(R) = O_2 \Rightarrow O_2 \equiv O_1$. The ontology O_2 is said to be *equivalent* to the ontology O_1 if a lexical overlap measure [11] denoted as $L(O_1, O_2)$ takes a value of 1. That is, $L(O_1, O_2) = 1 \Rightarrow O_2 \equiv O_1$. The lexical overlap measure is calculated as follows: $L(O_1, O_2) = |L_1 \cap L_2| / |L_1|$, where L_1 is a set of all constructs in the ontology O_1 and L_2 is a set of all constructs in the ontology O_2.

VIII. Conclusion and Future Work

We have proposed a novel approach to automatic transformation of ontologies to relational databases, where the quality of transformation is also considered. Our approach has been implemented in the QUALEG DB utility. This utility can be applied to any relational database management system that supports the standard SQL, because the utility does not rely on any SQL dialect. The utility can map all constructs of an ontology to a relational database, with the exception of those constructs that have no correspondences in the relational database (e.g. subproperties). The utility names the constructs of an ontology using the names of relational database constructs (converting the names as appropriate or required by name length restrictions in the relational database management system).

The main problem with our approach is the naming strategy, in particular, when an ontology that imports another ontology is transformed into a relational database. Unlike OWL, SQL does not support namespaces. A simple solution to this problem is to keep class names unique over multiple ontologies (as done in the QUALEG DB utility), but a more sophisticated naming strategy needs to be developed in the future.

References

[1] R. Harrison, and C. Chan, "Distributed ontology management system," in *Proc. 18th Annual Canadian Conf. on Electrical and Computer Engineering*, Saskatoon, Canada, 2005, pp. 661-664.

[2] Y. An, A. Borgida, and J. Mylopoulos, "Inferring complex semantic mappings between relational tables and ontologies from simple correspondences," in *Proc. OMT Conf.*, Agia Napa, Cyprus, 2005, pp. 1152-1169.

[3] J. Barrasa, O. Corcho, G. Shen, and A. Gomez-Perez, "R2O: An extensible and semantically based database-to-ontology mapping language," in *Proc. Workshop on Semantic Web and Databases*, Edinburgh, Scotland, 2004, pp. 1069-1070.

[4] N. Konstantinou, D. Spanos, M. Chalas, E. Solidakis, and N. Mitrou, "VisAVis: An approach to an intermediate layer between ontologies and relational database contents," in *Proc. Int. Workshop on Web Information Systems Modeling*, Luxembourg, Grand Duchy of Luxembourg, 2006.

[5] Z. Xu, S. Zhang, and Y. Dong, "Mapping between relational database schema and OWL ontology for deep annotation," in *Proc. of IEEE/WIC/ACM Int. Conf. on Web Intelligence*, Hong Kong, China, 2006, pp. 548-552.

[6] A. Gali, C. Chen, K. Claypool, and R. Uceda-Sosa, "From ontology to relational databases," in *Proc. Int. Workshop on Conceptual-Model Driven Web Information Integration and Mining*, Shanghai, China, 2004, pp. 278-289.

[7] E. Vysniauskas, and L. Nemuraite, "Transforming ontology representation from OWL to relational database," *Information Technology and Control*, vol. 35A, no. 3, 2006, pp. 333-343.

[8] I. Astrova, A. Kalja, and N. Korda, "Automatic transformation of OWL ontologies to SQL relational databases," in *Proc. IADIS European Conf. Data Mining (MCCSIS)*, Lisbon, Portugal, 2007, pp. 145-149.

[9] *OWL Web Ontology Language Reference*, 2004, Available: http://www.w3.org/TR/owl-ref

[10] *Database Language SQL*. ANSI X3.135, 2002, Available: http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

[11] M. Sabou, "Extracting ontologies from software documentation: A semi-automatic method and its evaluation," in *Proc. Workshop on Ontology Learning and Population*, Valencia, Spain, 2004.