Rule-Based Transformation of SQL Relational Databases to OWL Ontologies

Irina Astrova[1], Nahum Korda[2], and Ahto Kalja[1]

[1] Institute of Cybernetics, Tallinn University of Technology,
Akadeemia tee 21, 12618 Tallinn, Estonia
irinaastrova@yahoo.com
[2] Straight Technology Ltd,
London, UK
nahum.korda@straighttech.com

Abstract. This paper proposes a novel approach to automatic transformation of relational databases to ontologies, where the quality of transformation is also considered. A relational database is written in SQL, and an ontology is written in OWL. The proposed approach can be used for upgrading today's Web to the Semantic Web. The high cost of manual building ontologies from scratch is one of the main obstacles for the development of the Semantic Web. On the other hand, the Semantic Web can benefit from reuse of the vast amount of relational database information available on the Web today.

Keywords: Ontologies, relational databases, OWL, and SQL.

1 Introduction

"One of the main driving forces for the Semantic Web has always been the expression, on the Web, of the vast amount of relational database information in a way that can be processed by machines" [1]. Indeed, most information on the Web is not machine-processable, because it is often represented in HTML. This language describes how the information looks like and not what it is. In order for machines to process the information, it must be represented in an ontology language (e.g. OWL) and linked to ontologies. An ontology can be used for annotating HTML pages with semantics.

Manual or semi-automatic semantic annotation [2] is time-consuming, subjective and error-prone. It is even impossible on scale of the Web that contains billions of pages. Most pages even do not exist until they are dynamically generated from relational databases at the time of submitting HTML forms. Moreover, the high cost of building ontologies from scratch is another obstacle for the semantic annotation. An alternative to the semantic annotation is automatic or semi-automatic transformation of relational databases to ontologies, which is the purpose of this paper.

1.1 Transformation Problems

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

- **Loss of data:** The result of the transformation should adequately describe the original data.
- **Loss of semantics:** In some cases, the transformation is not really lossless in the sense that not all constructs in a relational database can be mapped to an ontology. Therefore, the quality of the transformation should be analyzed.
- **Focus on structures**: Besides the mapping of structures (i.e. tables, columns, etc.), 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 relational databases, not being used in practical situations, then the transformation suffers from the applicability problem.
- **Correctness:** The transformation should have provable correctness.

1.2 State of the Art

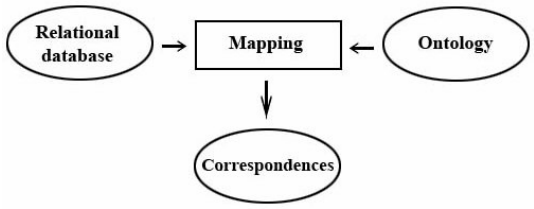
While there are several tools available for transforming relational databases to ontologies, many situations are too complex or require more flexibility than the existing tools enable. E.g. DataGenie [3] is a Protégé's plug-in that is capable of importing legacy data from a relational database (namely, Oracle) to an ontology. This import is simple: each table maps to a class, each column maps to a data type property and each row maps to an instance. In addition, foreign keys can be replaced with Protégé's instance pointers. The backside of this simplicity is that DataGenie fails to discover inheritance, object properties and restrictions. Moreover, DataGenie imposes a strict policy on the direction of import. In particular, users can move data from a relational database to an ontology, but not in a reverse direction (i.e. from an ontology to a relational database).

In the rest of the paper, we assume that a relational database is written in SQL [4], the standard relational database language, and that an ontology is written in OWL [5], the standard ontology language.

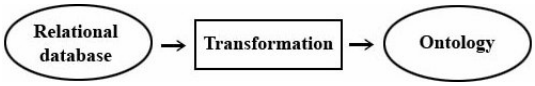
2 Related Work

A majority of the related work has been done on mapping between relational databases and ontologies; e.g. [6], [7], [8], and [9]. However, this mapping is quite different from transformation of relational databases to ontologies, as Fig. 1 shows. 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 a relational database exists, while an ontology is produced from the relational database. That is, the input to the transformation is a relational database and the output is an ontology.



(a): Mapping between relational database and ontology.



(b): Transformation of relational database to ontology.

Fig. 1. Mapping vs. transformation.

There are several approaches to transformation of relational databases to ontologies; e.g. [10], [11], [12], and [13]. However, all these approaches suffer from at least one of the following problems:

- They do not discover inheritance, thus producing an ontology that looks rather "relational"; i.e. the ontology has the same flat structure as the original relational database.
- They do not discover restrictions, symmetric and transitive properties either.
- They ignore constraints that capture additional semantics.
- They are not implemented.
- They are semi-automatic; i.e. they can require much user interaction.
- They do not analyze loss of semantics caused by the transformation. Rather, they assume that a relational model is a subset of an ontological model and thus, all constructs of a relational database can map to an ontology.

As an attempt to resolve these problems, we propose a novel approach to transformation of relational databases to ontologies. A relational database is written in SQL whereas an ontology is written in OWL.

3 Transformation

A relational database is an implementation of a relational model. This model includes constructs for specifying tables, domains, columns, data types, constraints, and other semantics, as Fig. 2 shows. 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).

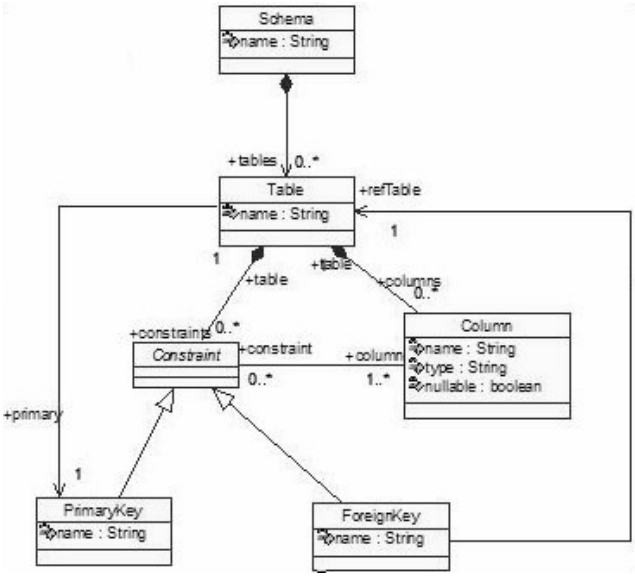


Fig. 2. Simplified relational model.

Similarly, an ontology is an implementation of an ontological model. This model includes constructs for specifying classes, properties, data types, inheritance, restrictions, and other semantics, as Fig. 3 shows. 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).

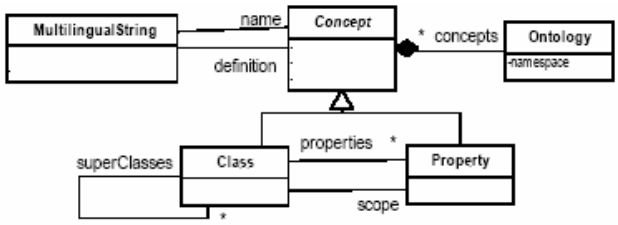


Fig. 3. Simplified ontological model.

Fig. 4 illustrates the basic idea behind our approach. Transformation of relational databases to ontologies is based on a set of rules called *mapping rules* that specify how to map constructs of the relational model to the ontological model (see Section

4). These rules are then applied to a relational database (source) to produce an ontology (target). Since the rules are specified on the model level, they are applicable to any relational database that conforms to the relational model.

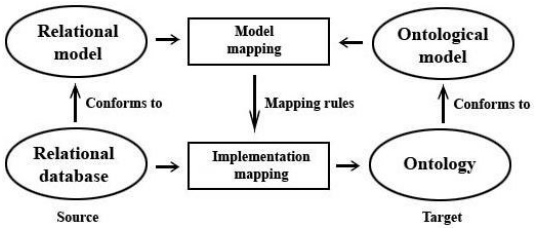


Fig. 4. Transformation of relational databases to ontologies.

4 Mapping Rules

Our approach maps constructs of a relational database to an ontology, using the names of constructs of the relational database as the names of constructs of the ontology. A prerequisite for this mapping is the mapping of constructs of a relational model to an ontological model. This mapping is defined by a set of rules for:

- Mapping tables
- Mapping columns
- Mapping data types
- Mapping constraints
- Mapping rows.

Next these rules will be illustrated by example. An example is the relational database for a company.

4.1 Mapping Tables

A table is mapped to a class unless all its columns are foreign keys to two other tables. Then it is mapped to two object properties (one is an inverse of another).

The primary key of a table `Involvement` in Fig. 5 is composed of foreign keys to two other tables `Project` and `Employee`, indicating a binary relationship (many-to-many). Since the table `Involvement` consists entirely of the foreign keys that are part of the primary key, it is mapped to two object properties: `EmployeeID` (that uses classes `Project` and `Employee` as its domain and range, respectively) and `ProjectID`. The latter is an inverse of the former, meaning that the relationship is bidirectional (i.e. a project involves employees and an employee is involved in projects).

```
CREATE TABLE Involvement(
 EmployeeID INTEGER REFERENCES Employee,
```

```
 ProjectID INTEGER REFERENCES Project,
 PRIMARY KEY (EmployeeID, ProjectID))
                                    ↓
<owl:ObjectProperty rdf:ID="EmployeeID">
 <rdfs:domain rdf:resource="#Project"/>
 <rdfs:range rdf:resource="#Employee"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ProjectID">
 <owl:inverseOf rdf:resource="#EmployeeID"/>
</owl:ObjectProperty>
```

Fig. 5. Table is mapped to two object properties.

The primary key of a table `Involvement` in Fig. 6 is composed of foreign keys to two other tables `Employee` and `Project`, indicating a binary relationship, again. However, since this table now has an additional column `hours`, it is mapped to a class `Involvement`.

```
CREATE TABLE Involvement(
 EmployeeID INTEGER REFERENCES Employee,
 ProjectID INTEGER REFERENCES Project,
 hours INTEGER,
 PRIMARY KEY (EmployeeID, ProjectID))
                                    ↓
<owl:Class rdf:ID="Involvement"/>
```

Fig. 6. Table is mapped to class.

The primary key of a table `Involvement` in Fig. 7 is composed of foreign keys to three other tables `Employee`, `Project` and `Skill`, indicating a ternary relationship. Since only binary relationships can be represented through object properties, this table is mapped to a class `Involvement`.

```
CREATE TABLE Involvement(
 EmployeeID INTEGER REFERENCES Employee,
 ProjectID INTEGER REFERENCES Project,
 SkillID INTEGER REFERENCES Skill,
 PRIMARY KEY (EmployeeID, ProjectID, SkillID))
                                    ↓
  <owl:Class rdf:ID="Involvement"/>
```

Fig. 7. Table is mapped to class (contd.).

4.2 Mapping Columns

A column is mapped to a data type property accompanied by a maximum cardinality of 1 unless it is a foreign key. (For mapping foreign keys, see Section 4.4.4.)

A column `ssn` in a table `Employee` in Fig. 8 is not a foreign key. Therefore, this column is mapped to a data type property `ssn` that uses a class `Employee` as its domain. Since the column `ssn` may have only one value for each row in the table `Employee` (atomicity), this property has a maximum cardinality of 1. Alternatively,

the property `ssn` could be defined as functional, which is the same as saying that the maximum cardinality is 1.

```
CREATE TABLE Employee(
 ssn INTEGER)
                                        ↓
<owl:DatatypeProperty rdf:ID="ssn">
 <rdfs:domain rdf:resource="#Employee"/>
 <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
<owl:Class rdf:ID="Employee">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#ssn"/>
   <owl:maxCardinality rdf:datatype=
"&xsd;nonNegativeInteger"1/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Fig. 8. Column is mapped to data type property with maximum cardinality of 1.

4.3 Mapping Data Types

Most of the mapping of columns has to do with mapping data types from SQL to XSD. Unlike SQL, OWL does not have any built-in data types. Instead, it uses XML Schema Data types (XSD). Table 1 shows how to map data types from SQL to XSD.

Table 1. Mapping data types.

SQL data type	XML Schema data type
SMALLINT	short
INTEGER	integer positiveInteger negativeInteger nonPositiveInteger nonNegativeInteger int long
DECIMAL	decimal
NUMERIC	decimal
FLOAT	float
REAL	float
DOUBLE PRECISION	double
CHARACTER	string
CHARACTER VARYING	string
TIME	time
TIME WITH TIME ZONE	time
DATE	date
TIMESTAMP	datetime
TIMESTAMP WITH TIME	datetime

ZONE	
INTERVAL	duration
BIT	boolean
BIT VARYING	byte

A column `ssn` in Fig. 9 uses `INTEGER` as its data type. Therefore, a data type property `ssn` uses `integer` as its range.

```
CREATE TABLE Employee(
 ssn INTEGER)
                                    ↓
<owl:DatatypeProperty rdf:ID="ssn">
 <rdfs:domain rdf:resource="#Employee"/>
 <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

Fig. 9. SQL data type INTEGER is mapped to XML Schema data type integer.

A column `ssn` in Fig. 10 uses `INTEGER` as its data type, again. However, there is now a constraint `CHECK` on the column `ssn`. This constraint specifies a data range for the column `ssn` to be all integers greater than 0 (i.e. all positive integers). Therefore, a data type property `ssn` uses `positiveInteger` as its range.

```
CREATE TABLE Employee(
 ssn INTEGER CHECK (ssn > 0))
                                    ↓
<owl:DatatypeProperty rdf:ID="ssn">
 <rdfs:domain rdf:resource="#Employee"/>
 <rdfs:range rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

Fig. 10. SQL data type INTEGER is mapped to XML Schema data type positiveInteger.

4.4 Mapping Constraints

4.4.1 Mapping Constraints UNIQUE

`UNIQUE` is a column constraint. It is mapped to an inverse functional property.

A constraint `UNIQUE` in Fig. 11 specifies that a column `ssn` in a table `Employee` is unique, meaning that no two rows in the table `Employee` have the same value for the column `ssn` (i.e. social security numbers uniquely identify employees). Therefore, this constraint is mapped to an inverse functional property.

```
CREATE TABLE Employee(
 ssn INTEGER UNIQUE)
                                    ↓
<owl:InverseFunctionalProperty rdf:ID="ssn"/>
```

Fig. 11. Constraint UNIQUE is mapped to inverse functional property.

4.4.2 Mapping Constraints NOT NULL

NOT NULL is a column constraint. It is mapped to a minimum cardinality of 1.

A constraint NOT NULL in Fig. 12 specifies that a column ssn in a table Employee is not null, meaning that all rows in the table Employee have values for the column ssn (i.e. all employees are assigned social security numbers). Therefore, this constraint is mapped to a minimum cardinality of 1.

```
CREATE TABLE Employee(
 ssn INTEGER NOT NULL)
                                    ↓
<owl:Class rdf:ID="Employee">
 <rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#ssn"/>
   <owl:minCardinality rdf:datatype=
"&xsd;nonNegativeInteger"1/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Fig. 12. Constraint NOT NULL is mapped to minimum cardinality of 1.

4.4.3 Mapping Constraints PRIMARY KEY

There are two forms of constraint PRIMARY KEY: using it as a column constraint (to refer to a single column) and using it as a table constraint (to refer to multiple columns). A column constraint PRIMARY KEY is mapped to both an inverse functional property and a minimum cardinality of 1.

A constraint PRIMARY KEY in Fig. 13 specifies that a column ssn in a table Employee is a primary key, which is the same as saying that the column ssn is both unique and not null. Therefore, this constraint is mapped to both an inverse functional property and a minimum cardinality of 1.

```
CREATE TABLE Employee(
 ssn INTEGER PRIMARY KEY)
                                    ↓
<owl:InverseFunctionalProperty rdf:ID="ssn"/>
<owl:Class rdf:ID="Employee">
 <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#ssn"/>
    <owl:minCardinality rdf:datatype=
"&xsd;nonNegativeInteger"1/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Fig. 13. Constraint PRIMARY KEY is mapped to both inverse functional property and minimum cardinality of 1.

4.4.4 Mapping Constraints REFERENCES and FOREIGN KEY

REFERENCES is a column constraint, whereas FOREIGN KEY is a table constraint. Both constraints are used for specifying foreign keys. A foreign key can be mapped to four different constructs in the ontology: an object property, class inheritance, a symmetric property and a transitive property.

A constraint REFERENCES in Fig. 14 specifies that a column ProjectID in a table Task is a foreign key to another table Project, indicating a binary relationship (one-to-one or many-to-one). Since the foreign key is not (part of) the primary key, it is mapped to an object property ProjectID that uses classes Task and Project as its domain and range, respectively.

```
CREATE TABLE TASK(
 TaskID INTEGER PRIMARY KEY,
 ProjectID INTEGER REFERENCES Project)
                                    ↓
<owl:ObjectProperty rdf:ID="ProjectID">
 <rdfs:domain rdf:resource="#Task"/>
 <rdfs:range rdf:resource="#Project"/>
</owl:ObjectProperty>
```

Fig. 14. Foreign key is mapped to object property.

A constraint REFERENCES in Fig. 15 specifies that a column ProjectID in a table Task is a foreign key to another table Project, indicating a binary relationship, again. However, since the foreign key is now part of the primary key, this relationship is tighter than the previous one. Therefore, the foreign key is mapped to an object property ProjectID accompanied by a cardinality of 1.

```
CREATE TABLE TASK(
 TaskID INTEGER,
 ProjectID INTEGER REFERENCES Project,
 PRIMARY KEY (TaskID, ProjectID))
                                    ↓
<owl:ObjectProperty rdf:ID="ProjectID">
 <rdfs:domain rdf:resource="#Task"/>
 <rdfs:range rdf:resource="#Project"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Task">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#ProjectID"/>
    <owl:cardinality rdf:datatype= "&xsd;nonNegativeInteger"1/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Fig. 15. Foreign key is mapped to object property with cardinality of 1.

A constraint FOREIGN KEY in Fig. 16 specifies that a column ProjectID in a table SoftwareProject is a foreign key to another table Project, indicating a binary relationship, again. However, since the foreign key is now the primary key, it

is mapped to class inheritance: `SoftwareProject` is a subclass of `Project` (i.e. a software project is a project).

```
CREATE TABLE SoftwareProject(
 ProjectID INTEGER PRIMARY KEY,
 FOREIGN KEY (ProjectID) REFERENCES Project)
                                    ↓
<owl:Class rdf:ID="SoftwareProject">
 <rdfs:subClassOf rdf:resource="#Project"/>
</owl:Class>
```

Fig. 16. Foreign key is mapped to class inheritance.

A constraint `REFERENCES` in Fig. 17 specifies that a column `spouse` in a table `Employee` is a foreign key to the same table, indicating a unary relationship. Therefore, the foreign key is mapped to a symmetric property `spouse` that uses a class `Employee` as both its domain and range (i.e. if one employee is a spouse of another employee, then the second employee is a spouse of the first employee).

```
CREATE TABLE Employee(
 EmployeeID INTEGER PRIMARY KEY,
 spouse INTEGER REFERENCES Employee)
                                    ↓
<owl:SymmetricProperty rdf:ID="spouse">
 <rdfs:domain rdf:resource="#Employee"/>
 <rdfs:range rdf:resource="#Employee"/>
</owl:SymmetricProperty >
```

Fig. 17. Foreign key is mapped to symmetric property.

A constraint `REFERENCES` in Fig. 18 specifies that a column `subtask` in a table `Task` is a foreign key to the same table, indicating an unary relationship, again. However, since the foreign key is now accompanied by a trigger `ON DELETE CASCADE`, this relationship consists of a whole and a part, where the part cannot exist without the whole (i.e. if a task is deleted, then all its subtasks must also be deleted). Therefore, the foreign key is mapped to a transitive property `subtask` that uses a class `Task` as both its domain and range (i.e. if one task is a subtask of another task and the other task is a subtask of yet another task, then the first task is a subtask of the third task).

```
CREATE TABLE Task(
 TaskID INTEGER PRIMARY KEY,
 subtask INTEGER REFERENCES Task ON DELETE CASCADE)
                                    ↓
<owl:TransitiveProperty rdf:ID="subtask">
 <rdfs:domain rdf:resource="#Task"/>
 <rdfs:range rdf:resource="#Task"/>
</owl:TransitiveProperty >
```

Fig. 18. Foreign key is mapped to transitive property.

4.4.5 Mapping Constraints CHECK

There are two forms of constraint `CHECK`: using it as a column constraint (to refer to a single column) and using it as a table constraint (to refer to multiple columns). A column constraint `CHECK` is mapped to a value restriction unless it has an enumeration. (For mapping enumerations, see Section 4.4.6).

A constraint `CHECK` in Fig. 19 specifies that all rows in a table `Project` have a value `Software` for a column `type`. Therefore, a data type property `type` is restricted to have the same value for all instances of a class `Project`.

```
CREATE TABLE Project(
 type VARCHAR CHECK (type='Software'))
                                    ↓
<owl:Class rdf:ID="Project">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#type"/>
    <owl:hasValue rdf:datatype="&xsd;string">Software
    </owl:hasValue>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Fig. 19. Constraint CHECK is mapped to value restriction.

4.4.6 Mapping Constraints CHECK with enumeration

A constraint `CHECK` with enumeration is mapped to an enumerated data type.

A constraint `CHECK` in Fig. 20 specifies a data range for a column `sex` in a table `Employee` through a list of possible values `Male` and `Female`. Therefore, this constraint is mapped to an enumerated data type, with one element for each possible value in the list.

```
CREATE TABLE Employee(
 sex VARCHAR CHECK (sex IN ('Male', 'Female')))
                                    ↓
<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:first>
     <rdf:rest>
      <rdf:List>
       <rdf:first rdf:datatype="&xsd;string">Female
       </rdf:first>
       <rdf:rest rdf:resource="&rdf;nil"/>
      </rdf:List>
     </rdf:rest>
    </rdf:List>
   </owl:oneOf>
```

```
  </owl:DataRange>
 </rdfs:range>
</owl:DatatypeProperty>
```

Fig. 20. Constraint CHECK with enumeration is mapped to enumerated data type.

4.5 Mapping Rows

A row is mapped to an instance.

A row in a table `Project` in Fig. 21 has a value `Software` for a column `type`. Therefore, this row is mapped to an (anonymous) instance of a class `Project` that has the same value for a data type property `type`.

```
INSERT INTO Project (type) VALUE ('Software')
                                    ↓
<Project>
 <type rdf:datatype="&xsd:string">Software
 </type>
</Project>
```

Fig. 21. Row in table is mapped to instance of class.

5 Implementation

Our approach is implemented in a tool called QUALEG DB (http://www.qualeg.eupm.net). This tool is capable of automatic transformation of a relational database (written in SQL) to an ontology (written in OWL). As its core, the tool is a transformation engine that parses an SQL script and generates an OWL file that contains an ontology, including definitions (classes, properties and restrictions) and instances (values and individuals).

The tool requires minimum user interaction. The only thing users need to do is to select or specify the name for an SQL script and the name for an OWL file, as Fig. 22 shows.

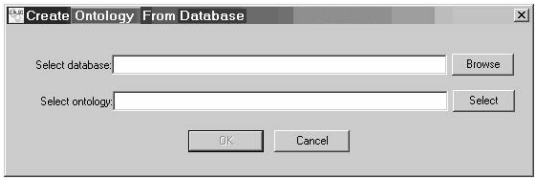


Fig. 22. Graphical user interface of QUALEG DB.

When parsing the SQL script, the tool performs consistency and error checks. These checks are important because they prevent certain kinds of errors in the resulting ontology. Violation of any of the checks will lead to errors. If the tool encounters any error, it will display this error to the user (as Fig. 23 shows) and continues the transformation unless the error is terminal. However, an incorrect construct that has caused the error will be excluded from the transformation.

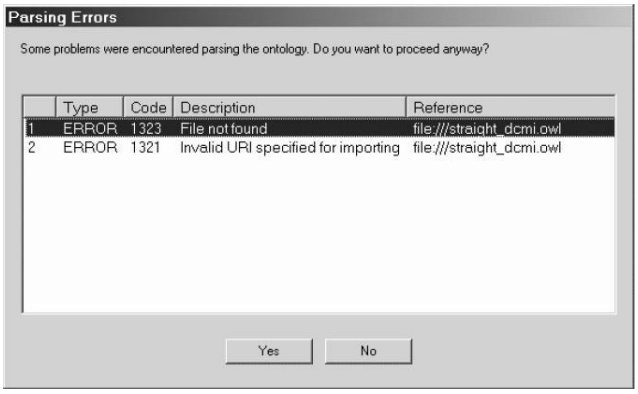


Fig. 23. Consistency and error checks in QUALEG DB.

6 Quality of Transformation

Since an ontological model does not support all constructs of a relational model (e.g. a constraint DEFAULT has no correspondence in the ontological model), some of the semantics captured in a relational database will necessarily be lost when transforming the relational database to an ontology. Therefore, we need to analyze loss of semantics caused by this transformation. One way to do this is to retransform the resulting ontology to a relational database and see if the transformation is reversible. By reversible, we mean that transformation of a relational database to an ontology followed by reverse transformation of the resulting ontology to a relational database will yield the original relational database.

More formally, let T_1 be transformation of a relational database R_1 to an ontology O. Let T_2 be reverse transformation of the ontology O to a relational database R_2. The transformation T_1 is said to be *reversible* if the relational database R_2 is equivalent to the relational database R_1. That is, $T_1(R_1) = O \wedge T_2(O) = R_2 \Rightarrow R_2 \equiv R_1$.

The relational database R_2 is said to be *equivalent* to the relational database R_1 if a lexical overlap measure [14] denoted as $L(R_1, R_2)$ takes a value of 1. That is, $L(R_1, R_2) = 1 \Rightarrow R_2 \equiv R_1$. The lexical overlap measure is calculated as follows:

$$L(R_1, R_2) = |L_1 \cap L_2| / |L_1|$$

where L_1 is a set of all constructs in the relational database R_1 and L_2 is a set of all constructs in the relational database R_2.

7 Conclusion

We have proposed a novel approach to automatic transformation of relational databases to ontologies, where the quality of transformation is also considered. Our approach has been implemented in the tool QUALEG DB. This tool can be applied to any relational database management system that supports SQL, because the tool does not rely on any SQL dialect. The tool can map all constructs of a relational database to an ontology, with the exception of those constructs that have no correspondences in the ontology (e.g. a constraint `DEFAULT`).

The tool has been used in a European project called QUALEG (Quality of Service and Legitimacy in e-Government) (http://www.qualeg.eupm.net) and proven in practice. The tool can be used for upgrading the Web to the Semantic Web. This upgrade goes through two basic steps: (1) transforming a relational database to an ontology; and (2) linking HTML pages (that are dynamically generated from the relational database) to the ontology. Not only does the upgrade save efforts in developing the Semantic Web from scratch, but it also makes the vast amount of relational database information on the Web machine-processable.

By contrast to the semantic annotation, the transformation requires the original HTML pages to be changed minimally. The only change is a link to the ontology. Furthermore, the transformation requires less user interaction, thus giving more opportunity for automation.

8 Future Work

In the future, we'll prove the correctness of transformation using theoretical mappings; e.g. [15]. A general process of proving the correctness of transformation will consist of three basic steps. First, we'll describe the required properties of ontology explicitly. We'll prefer to have (independent) well-formedness conditions here, as this facilitates the systematic treatment in the next two steps. Second, we'll describe transformation of a relational database to an ontology. This transformation is defined by mapping rules that can be enhanced using guidance parameters. These parameters are interpreted as producing the ontology having certain desirable qualities. Third, we'll prove that the result of the transformation meets the well-formedness conditions. As a consequence, the resulting ontology is correct in the sense that all the well-formedness conditions are met. In addition, when the guidance parameters are used, we'll prove that the resulting ontology has the desirable qualities (defined by the guidance parameters) as well.

Acknowledgments. This research is partly sponsored by ESF (Estonian Science Foundation) under the grant nr. 5766.

References

1. Berners-Lee, T.: Relational Databases on the Semantic Web. (2002) http://www.w3.org/DesignIssues/RDB-RDF.html
2. Erdmann, M., Maedche, A., Schnurr, H., Staab, S.: From Manual to Semi-automatic Semantic Annotation: About Ontology-based Text Annotation Tools. In: Linköping Electronic Articles in Computer and Information Science Journal. Vol. 6, No. 2 (2001)
3. DataGenie: (2007) DataGenie. http://protege.cim3.net/cgi-bin/wiki.pl?DataGenie
4. SQL: (2002) Database language SQL. ANSI X3.135. www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt
5. OWL: (2004) OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref
6. An, Y., Borgida, A., Mylopoulos, J.: Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. In: OTM'05, On The Move Federated Conference (2005)
7. Barrasa, J., Corcho, O., Shen, G., Gomez-Perez, A.: R2O: An extensible and semantically based database-to-ontology mapping language. In: SWDB'04, 2nd Workshop on Semantic Web and Databases (2004)
8. Konstantinou, N., Spanos, D., Chalas, M., Solidakis, E., Mitrou, N.: VisAVis: An approach to an intermediate layer between ontologies and relational database contents. In: WISM'06, International Workshop on Web Information Systems Modeling (2006)
9. Xu, Z., Zhang, S., Dong, Y.: Mapping between relational database schema and OWL ontology for deep annotation. In WI'06, IEEE/WIC/ACM International Conference on Web Intelligence (2006)
10. Li, M., Du, X., Wang, S.: Learning Ontology from Relational Database. In: Proceedings of the 4th International Conference on Machine Learning and Cybernetics. Vol. 6 (2005) 3410–3415
11. Shen, G., Huang, Z., Zhu, X., Zhao, X.: Research on the Rules of Mapping from Relational Model to OWL. In: Proceedings of the Workshop on OWL: Experiences and Directions. Vol. 216 (2006)
12. Astrova, I., Kalja, A.: Towards the Semantic Web: Extracting OWL Ontologies from SQL Relational Schemata. In: Proceedings of IADIS International Conference WWW/Internet (2006) 62–66
13. Buccella, A., Penabad, M., Rodriguez, F., Farina, A., Cechich, A.: From Relational Databases to OWL Ontologies. In: Proceedings of the 6th National Russian Research Conference (2004)
14. Sabou, M.: "Extracting ontologies from software documentation: A semi-automatic method and its evaluation," in Proc. Workshop on Ontology Learning and Population, Valencia, Spain (2004)
15. Motik, B., Horrocks, I., Sattler, U.: Integrating Description Logics and Relational Databases, Manchester, UK (2006)