From Relational Databases to OWL Ontologies

Agustina Buccella[1], Miguel R. Penabad[2], Francisco J. Rodriguez[2],
Antonio Fariña[2], and Alejandra Cechich[1]
[1] Departamento de Ciencias de la Computación,
Universidad Nacional del Comahue, Buenos Aires 1400, 8300 Neuquén, Argentina
{abuccell,acechich}@uncoma.edu.ar
[2] Laboratorio de Base de Datos, Departamento de Computación
Universidade da Coruña, Campus de Elviña s/n, 15071 A Coruña, Spain
{penabad,fari}@udc.es, franjrm@uvigo.es

Abstract

The use of ontologies and ontology languages like OWL has attracted much attention, mainly in the Semantic Web and the information integration research fields. We have addressed the latter issue, proposing an architecture and a method, based on the use of ontologies, to integrate several sources of information, possibly of different natures, into a federated system. In this paper, we address the first step of our integration method: to automatically build an initial ontology from an existing database that is to be integrated in the federation. We show a procedure that takes as input the relational schema (the SQL "create table" sentences) of an existing database and, following a set of rules, transforms it into an OWL ontology. The initial ontology can be later modified or extended, if more semantic knowledge about the domain is needed

1 Introduction

The federation of different data sources is a long-standing and thoroughly studied problem. Since the appearance of the ontologies and the proliferation of the *Semantic Web*, this problem has regained much attention.

The autonomy of the information sources, their geographical distribution and the heterogeneity among them, are the main problems we must face to perform the integration [12]. The semantic heterogeneity has been one of the most researched aspects in the last years. Works like [9,14] are aimed to fill the semantic gap among the information sources, using the semantic information provided by the ontologies.

In recent works [4,5,6] we have proposed an architecture and a method to solve semantic heterogeneity problems [18]. Figure 1 shows a part of the architecture of our federation system, which is based on a hybrid ontology approach [19]. As we can see, the architecture is composed of a *global ontology* or *shared vocabulary* containing the generic concepts that will be used to query the system, and one *source ontology* for each data source. The *Ontology Mapping (OM)* component deals with the information flow between the source ontologies and the shared vocabulary. Once the user chooses the concepts from the global ontology and makes the query, the system will use the OM to know the related concepts on each information source. The *source ontologies* are used for this purpose, given that they provide the specific terminology for the information sources and the means to obtain the desired data.

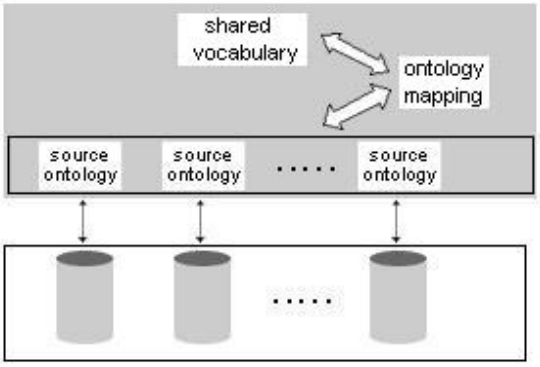

Fig. 1. A part of our integration system

The method we propose to add a new data source to an existing federation consists of two main stages: *building the source ontology* and *building the mappings between the source ontology and the shared vocabulary*. These stages are briefly explained below. See [5,6] for a full explanation.

Building the source ontology: This stage takes as input a data source and its result is an OWL ontology [17,1] for the data source. It contains two main steps: *generating the OWL initial ontology,* and *adding semantics*. The first step, building an initial ontology from the source, is the aim of this paper, and will be described in the following sections. The second stage, *adding semantics*, allows the expert user (for example, using an ontology editor as Protégé [10] with the OWL plug-in) to add restrictions, classes and/or properties to the initial ontology. Knowing the domain of the

information source and understanding the structures, the user is able to provide more semantics to the ontology.

Building the mappings between the source ontology and the shared vocabulary: This stage contains three main steps: *searching for similarities*, *adding mapping into the OM* and *adding the new information into the shared vocabulary*. Using the ontologies built in the previous stage, the first step searches for similarities among concepts and properties of the source ontology and the current shared vocabulary. We use the similarity functions defined on [15,16]. The second step, adding mapping into the OM, adds the mapping found in the similarity process to the OM. Finally, in the last step, adding the new information into the shared vocabulary, the shared vocabulary is updated with the new classes and properties only contained in the source ontology. Thus, the shared vocabulary will make available all the information the sources ontologies offer.

Our goal is to automate as much as possible of this process. In this paper, we will focus on the first stage of our method, building the source ontology, more specifically in the first step, *generating the OWL initial ontology*. Currently, almost all information we want to federate is found on either Web pages or relational databases. We shall discuss in this paper how to (semi)automatically build the initial ontology for an existing relational database.

As for the target language, we have chosen OWL due to its widespread use in the Semantic Web [3]. Besides, OWL allows formalizing a domain by defining classes and properties of those classes, to define individuals asserting properties about them, and to reason about these classes and individuals to the degree permitted by the formal semantics of the OWL language. OWL can be (partially) mapped to a description logic [2] making possible the use of existing reasoners such as FACT [13] and RACER [11].

The rest of this paper is organized as follows. Section 2 discusses the most important languages used in database modelling, and how easy they are to translate into OWL. In Section 3 we show the rules to create the OWL initial ontology using the DDL schema of an existing relational database. Conclusions and future work are shown in the last section.

2 Database modeling languages

As stated in the previous section, our main goal is to integrate different sources of information into a federated system. The first step of our approach is to build a *source ontology* for each data source. This step is usually performed manually, thus being a tedious, time-consuming and error-prone task. To avoid these problems, we want to automate as much as possible this task. Given that most data sources are databases, specifically relational databases, we shall focus in this section in the different modelling languages and how then can be (semi)automatically translated to an ontology language.

Relational databases are usually built following three steps: Define the conceptual model (usually in an ER diagram); define the logical model (relational schema); and implement it using DDL (Data Definition Language) statements of SQL.

This three steps use three different languages (ER, relational and DDL). Let us see their advantages and disadvantages with respect to their transformation into an ontology language (in this paper, we shall consider OWL as the target language).

2.1 Entity-Relationship Model

The Entity-Relationship model [7] is a conceptual model that allows, through the use of ER diagrams, to describe a particular domain. Being a conceptual model, it is closer to the "semantic" point of view of the ontologies than the relational language or the DDL.

However, ER diagrams also have several important drawbacks. First, it is difficult to parse an ER diagram to automatically build an ontology, because it is a graphical representation (however, this could be overcome, because a graphical ER diagram should be easy to convert into a set of formal definitions). Additionally, there are no explicit declarations of data types (domains of the attributes), and no constraints, except participation and cardinality of the relationships, can be reflected in an ER diagram (i.e., no restrictions like "age is a positive integer" are possible).

2.2 The Relational Model

The relational model (RM) [8] has a strong mathematical foundation. This makes it closer to the description logic, which is the basis for many ontology languages, including OWL.

Tables, equivalent to ontology classes, are easily represented. The main advantage with respect to ER is that the relational model includes domains for the attributes.

The main problem of a relational schema to be converted in an ontology comes from the relationships. One-to-many (an one-to-one) relationships without attributes are easily found through foreign keys; if the foreign key is a subset of the primary key, it is almost sure that the one-to-many relationships links a weak entity to a strong one. However, many-to-many relationships, as well as one-to-many if they include attributes, are somehow hidden, because they generate new tables that are apparently identical to "entity" tables.

Finally, the participation and cardinalities of the relationships are also difficult to represent.

2.3 Data Definition Language

The advantages and disadvantages of the relational model generally apply to the Data Definition Language (DDL), considering a DDL sentence as the implementation (creation) of a relational table. However, it is more powerful, since it can add some expressive power that is not available in the theoretical RM. For example, attribute domains can be more

explicit, and many types of constraints can be defined, either via column of table constraints, or using more complex techniques, such as the use of assertions or triggers.

Finally, there is a vital factor that affects the ability of a database model to be used in our system: its availability. Unfortunately, databases are built in many cases without much effort on conceptual or logical modelling, thus the only documentation we can count on is the DDL of relational schema as implemented in a particular DBMS. Given that we want to federate existing databases, we shall use the DDL as the input language to our method to build an ontology. We leave as future work to provide the same set of rules to perform the translation from the ER or RM to OWL.

In the following section we shall see a short example with an ER diagram and the corresponding DDL sentences for the creation of the tables, in order to better understand the transformation rules into OWL.

3 The Ontology Creation Process

In this section we will show the automated migration process based on the SQL/DDL-code used to build an existing database. By a series of steps we will build an initial ontology which will be used for integration tasks. After this initial step, expert users can add more semantics to the ontology to capture additional restrictions and semantics about the domain.

The SQL/DDL code will be analyzed in the same order as the tables have been created. Therefore, tables without foreign keys are explored first. Each CREATE TABLE sentence is analyzed to find the table name and attributes, building the classes and datatype properties in OWL. If the table has FOREIGN KEY constraints, we will propose the creation of a series of object properties and classes which denote the same semantics. One-to-many, many-to-many and weak-entity relationships will be taken into account to transform them into OWL classes, properties and restrictions. Finally, we shall consider the translation of simple CHECK constraints and CREATE DOMAIN sentences into OWL.

3.1 The example database

Figure 2 shows an ER diagram that models museums, with information about when works of art are shown in different rooms, about the partners that support a given museum, and its employees. This minimum example shows all the different types of entities and relationships that can be present in an ER diagram: strong and weak entities (the attributes and keys are omitted for simplicity), and binary (one-to-many and many-to-many) and ternary relationships.

The translation of this ER diagram into a relational model is shown in Table 1. It shows the attributes of each table, where underlined sets of attributes represent the primary keys. Additionally, foreign keys (with referenced tables) are also shown.

We do not show here the full set of SQL/DDL sentences used to build the entire database. Instead, we will choose individual sentences in order to show specific translation rules from SQL to OWL.

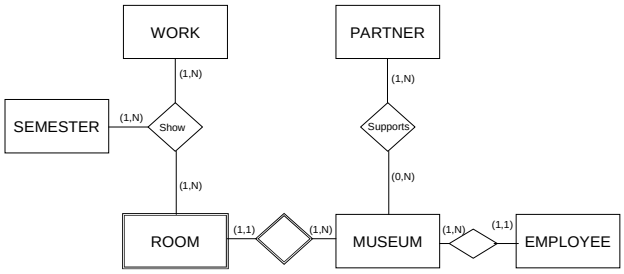

Fig. 2. ER diagram of the example database

Relation	***Foreign Keys (Referenced table)***
MUSEUM(NAME, ADDRESS)	
EMPLOYEE(EMP_ID, NAME, ADDRESS, MUS_NAME)	MUS_NAME (MUSEUM)
PARTNER(PARTNER_ID, NAME)	
SUPPORTS(PARTNER_ID, MUS_NAME)	PARTNER_ID PARTNER) MUS_NAME (MUSEUM)
ROOM(MUS_NAME,ROOM_ID)	MUS_NAME (MUSEUM)
WORK (WORK_NAME, AUTHOR)	
SEMESTER(SEMNO, YEAR)	
SHOWS(WORK_NAME, SEMNO, YEAR, MUS_NAME, ROOM_ID)	WORK_NAME (WORK) SEMNO, YEAR (SEMESTER) MUS_NAME, ROOM_ID (ROOM)

Table 1. Relational model of the example database

3.2 Creating the ontology

The source ontology is created by parsing the SQL/DDL sentences. The sentences are parsed in the order they are written:

1. Tables without foreign keys
2. Tables with foreign keys. Note that these tables can come from entities in the ER model (it is irrelevant whether they correspond to "weak" entities or not) or from relationships among entities in the ER model (either many-to-many, one-to-many with attributes, or with a degree greater than 2).

As indicated, we first analyze tables without foreign key constraints, to build the OWL classes and properties. Following the example, the *Museum* table can be one of the first tables created. Thus, we have the sentence:

```
CREATE TABLE Museum (
          name CHAR(20),
          address CHAR(50) NOT NULL,
          CONSTRAINT PK_MUSEUM
            PRIMARY KEY (name)

);
```

The *Museum* table contains two attributes but no foreign key restrictions, so it is just transformed into an OWL class, and its attributes are dealt with. In this case, the *Museum* table contains the *address* and *name* attributes, that are created in OWL, as DataType Properties, because they relate instances of classes with RDF literals and XML Schema DataTypes. These properties are created as Functional Properties because they have at most one unique value for each object. Besides, these properties do not have domain and range defined because they can be used by other classes. Therefore, we create the *prop_address* and *prop_name* properties.

Then, these properties must be assigned to the class with an *owl:allValuesFrom* restriction to denote the specific range. The range of both properties is a string type. The *owl:allValuesFrom* restriction requires that for every instance of the class that has instances of the specified property, the property values are all members of the class indicated by the *owl:allValuesFrom* clause.

In addition, the NOT NULL constraint that affects the attributes are also taken into account (the *name* attribute implicitly has this constraint because it is a primary key; the *address* attribute contains a NOT NULL contraint). These constraints are included in the ontology by assigning a cardinality restriction to the *Museum* class, since just using a Functional Property does not express this requirement. Therefore, the OWL-code to represent the *Museum* table is:

```
<owl:DatatypeProperty rdf:ID="prop_address">
<rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#
FunctionalProperty"/></owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="prop_museum">
<rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#
FunctionalProperty"/></owl:DatatypeProperty>

<owl:Class rdf:ID="Museum">
<rdfs:subClassOf>      <owl:Restriction>
<owl:onProperty>
<owl:DatatypeProperty
rdf:about="#prop_address"/>
</owl:onProperty>        <owl:allValuesFrom
rdf:resource="http://www.w3.org/2001/XMLSche
ma#string"/>
      </owl:Restriction>
</rdfs:subClassOf>
    <rdfs:subClassOf>      <owl:Restriction>
<owl:onProperty>
<owl:DatatypeProperty
rdf:about="#prop_address"/>
</owl:onProperty>        <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSche
ma#int">1</owl:cardinality>
</owl:Restriction>    </rdfs:subClassOf>
<rdfs:subClassOf>    <rdfs:subClassOf>
<owl:Restriction>
        <owl:onProperty>
<owl:DatatypeProperty
rdf:about="#prop_museum_name"/>
</owl:onProperty>        <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSche
ma#int">1</owl:cardinality>
      </owl:Restriction>
</rdfs:subClassOf>    <rdfs:subClassOf>
<owl:Restriction>        <owl:onProperty>
<owl:DatatypeProperty
rdf:about="#prop_museum_name"/>
</owl:onProperty>        <owl:allValuesFrom
rdf:resource="http://www.w3.org/2001/XMLSche
ma#string"/>      </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

With this code, we are saying that: *prop_address* is a property of *Museum* class, every instance of the *prop_address* property within *Museum* class are string instances, exactly *one* of the *prop_address* properties of a *Museum* must point to an individual that is a string, *prop_name* is a property of *Museum* class and exactly *one* of the *prop_museum_name* properties of a *Museum* must point to an individual that is a string.

This process is followed to translate all tables without foreign keys. When the table has foreign keys, there are two different cases: when the table has *exactly* one, and when it has two ore more. For the first case, we shall use the *Employee* table as example. This table has its own attributes and one FOREIGN KEY constraint that references *Museum*. From the semantic point of view, this is a one-to-many relationship. The SQL/DDL sentence is:

```
CREATE TABLE Employee (
       emp_id INTEGER,
       name CHAR(20),
       address CHAR(20),
       mus_name CHAR(20) NOT NULL,
       CONSTRAINT PK_EMPLOYEE
            PRIMARY KEY (emp_id),
       CONSTRAINT FK_MUSEUM
            FOREIGN KEY(mus_name)
            REFERENCES Museum(name)
);
```

In OWL, these types of relationships generate the creation of a set of classes and properties. After creating the *Employee* class and its attributes as explained above, we must create the attribute with the FOREIGN KEY constraint. In order to do so, the *prop_employee_museum* property is created as a Functional Object Property. This property contains the *Employee* class as domain and *Museum* as range. In order to denote the two roles of the one-to-many relationship, the inverse of this property is also created. That is, one property defines in which museum an employee works (*prop_employee_museum* property), and the inverse defines the employees of the museum (*inverse_of_ prop_employee_museum* property). As the relationship was one-to-many, this last property is not functional.

Unlike DataType properties used to represent "common" attributes, properties used to represent foreign keys have a predefined domain and range, so we do not need to assign them to the *Employee* and *Museum* class. In the case of *prop_emp_museum* property, as the SQL/DDL-code has a NOT NULL constraint, a cardinality restriction of 1 is also added. Thus, an employee works on exactly one museum. For *inverse_of_ prop_emp_museum* property, using the SLQ/DDL-code it is not possible to know the minimal cardinality (it would be possible to know if using the ER diagram, which shows a minimal cardinality of 1). Thus, the analyst can (and in this case should), after the ontology is built, add this cardinality, to represent that every museum has at least one employee.

The relationship between a weak entity and a strong one is a special case of one-to-many relationship. In SQL/DDL-code, this relationship appears when the primary key is compound, and the foreign key is a subset of this primary key. Following our example, each *Room* of a museum is a weak entity of *Museum*. In OWL, these relationships are defined in a very similar way as the foreign keys in the example above, but the cardinality restriction of 1 is always present (because the foreign key is part of the primary key).

To translate the *Room* table into OWL, two properties are created: a Functional Datatype Property (*prop_room_id*) and a Functional Object Property (*prop_room_museum*).

Prop_room_id is a Functional Datatype Property with an *owl:allValuesFrom* restriction and a cardinality of 1, because the *room_id* attribute is part of the primary key.

For *prop_room_museum*, the domain and range are defined as *Room* and *Museum*, respectively, adding also a cardinality of 1. Again, the inverse of *prop_room_museum* is also defined, because we want to denote the two roles of the relationship and also to give the opportunity to the users to include the exact cardinality of the *Museum* class in the relationship, given that this cardinality can not be obtained from the SQL/DDL-code.

When the table has two foreign key constraints, we have two different situations: when the table comes from a many-to-many relationship without additional attributes (the OWL class is not created, just the relationships) and when it has other attributes (we create the class and its properties).

Let us consider the first situation. The *Supports* table has two foreign keys:

```
CREATE TABLE Supports (
       partner_id INTEGER,
       mus_name CHAR(20),
       CONSTRAINT PK_SUPPORTS
          PRIMARY KEY(   partner_id,
                         mus_name),
       CONSTRAINT FK_MUS
          FOREIGN KEY(mus_name)
          REFERENCES Museum(name),
       CONSTRAINT FK_PARTNER
          FOREIGN KEY(partner_id)
          REFERENCES Partner(partner_id)
);
```

In this case, a property *prop_partner_museum* is created, defining *Partner* as domain and *Museum* as range; the inverse of this property (*inverse_of_prop_partner_museum*) is also defined. As in previous examples, the cardinalities can not be inferred from the SQL/DDL-code, so they can be later added by expert users. In this case, *prop_partner_museum* does not have any restriction because there are partners without supported museums, but *inverse_of_prop_partner_museum* property must have the minimal cardinality restriction.

The second situation happens when the table contains additional attributes. For example, if *Supports* had a *date* attribute. In this case, the *Support* class is created, as well as its attributes as properties, like in the previous cases.

When a table contains more than two foreign keys, it is necessary to create the OWL class, regardless of whether it has additional attributes or not. This is because OWL does not allow properties with a degree greater than 2. In our example the *Shows* table has three foreign keys:

```
CREATE TABLE Shows (
       room_id INTEGER,
       mus_name CHAR(20),
       work_name CHAR(20),
       semester INTEGER,
       year INTEGER,
       CONSTRAINT PK_SHOWS
          PRIMARYKEY( room_id,
                      mus_name,
                      work_name,
                      semester,
                      year),
       CONSTRAINT FK_WORK
          FOREIGN KEY(work_name)
          REFERENCES Work(work_name),
       CONSTRAINT FK_MUS2
          FOREIGN KEY(room_id, mus_name)
          REFERENCES Room(room_id,
                          mus_name),
       CONSTRAINT FK_SEM
          FOREIGN KEY(semester, year)
          REFERENCES Semester(semester,
                              year)
);
```

In order to represent this situation in OWL, we must create the *Shows* class. Then, as in the previous examples, one property for each foreign key is created, relating the *Shows* class with the referenced class. These properties are Functional Object Properties. Also, we create the inverses of each of them, as the following OWL fragment shows.

```
<owl:FunctionalProperty
rdf:ID="prop_shows_room">    <rdfs:domain
rdf:resource="#Shows"/>    <rdfs:range
rdf:resource="#Room"/>    <owl:inverseOf
rdf:resource="#inverse_of_prop_shows_room"/>
<rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#
ObjectProperty"/></owl:FunctionalProperty>
<owl:FunctionalProperty
rdf:ID="prop_shows_semester">
<rdfs:domain rdf:resource="#Shows"/>
<rdfs:range rdf:resource="#Semester"/>
    <owl:inverseOf
rdf:resource="#inverse_of_prop_shows_semeste
r"/>    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#
ObjectProperty"/></owl:FunctionalProperty>
<owl:FunctionalProperty
rdf:ID="prop_shows_work">    <rdfs:domain
rdf:resource="#Shows"/>
    <rdfs:range rdf:resource="#Work"/>
    <owl:inverseOf
rdf:resource="#inverse_of_prop_shows_work"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#
ObjectProperty"/></owl:FunctionalProperty>
```

The cardinality equal to 1 restriction for the three properties must be added in the *Shows* class because they are primary keys. Again, the cardinality restrictions

in the *Work*, *Room* and *Semester* classes can not be obtained from the SQL/DDL-code. Therefore, the users must add them when it is necessary.

3.3 Considering constraints and domains

Besides the CREATE TABLE sentences, it is usual to find in the DDL schema of a database other constructs that could be of use to implement the source ontology. Among these, we find simple CHECK constraints (inside CREATE TABLE statements) and CREATE DOMAIN sentences.

These statements can be included in the ontology by creating classes to represent the domains. For example, the *Work* table can have a CHECK restriction in which the authors must begin with the 'A' letter:

```
CREATE TABLE Work (
        work_name CHAR(20),
        author CHAR(20),
        CHECK (author LIKE 'A% ')
);
```

This restriction can be represented in OWL by creating the *beginning_with_A* class which denotes the strings values which begin with the 'A' letter. Also, the *prop_author* property is created to represent the *author* attribute.

```
<owl:Class rdf:ID="Work">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty
rdf:about="#prop_author"/>
</owl:onProperty>
<owl:allValuesFrom>          <owl:Class
rdf:ID="beginning_with_A"/>
</owl:allValuesFrom>
</owl:Restriction>
</rdfs:subClassOf></owl:Class>
```

Enumerated data types can be easily created by using the *owl:oneof* constructor. CREATE DOMAIN sentences are translated in a similar way as check constraints, because we also create new class to denote the constraint. Later, the class properties that correspond to attributes of this domain are attached to the domain class.

4. Conclusions and Future Work

We have briefly described an integration method which serves as a guide to integrate a new data source into a federation system. In this paper, we only consider relational databases as the data sources to be federated. After analyzing the advantages and disadvantages of the three languages that can be used to describe a relational database (ER diagram, relational schema, and DDL statements of SQL), we have chosen SQL/DDL because of its expressivity and mainly because it is always available. In this paper we have later focused on the first step of the method, *generating the OWL initial ontology*, specifying a set of rules to transform the tables of the relational schema into an OWL ontology. One-to-many, many-to-many, weak-entity and ternary relationships have been taken into account to build this ontology.

As a future work, we plan to add other more complex constraints specified in SQL in order to obtain all possible information of the relational databases, such as more complex CREATE DOMAIN sentences or CHECK constraints, or even triggers. Also, it is our intention to define this set of transformation rules for other languages, such as ER diagrams or UML class diagrams.

References

1. Antoniou, G., Harmelen F. Web Ontology Language: OWL. Handbook on Ontologies in Information Systems. Staab & Studer Editors. Springer-Verlag, 2003.
2. Baader, F., Calvanese, D., McGuiness, D., Nardi, D. and Patel-Schneider, P. editors. The Description Logic Handbook - Theory, Implementation and Applications. Cambridge University Press, ISBN 0-521-78176-0, 2003.
3. Berners-Lee, T. *Weaving the Web*. Texere Publishing Ltd. ISBN: 0752820907. June 2001.
4. Buccella A., Cechich A. and Brisaboa N.R. An Ontology Approach to Data Integration. Journal of Computer Science and Technology. Vol.3(2). Available at http://journal.info.unlp.edu.ar/default.html, 2003, (pp. 62-68).
5. Buccella A., Cechich A. and Brisaboa N.R. An Ontological Approach to Federated Data Integration. 9° Congreso Argentino en Ciencias de la Computación, CACIC'2003, La Plata, October 6-10, 2003, (pp. 905-916)..
6. Buccella A., Cechich A. and Brisaboa N.R. A Context-Based Ontology Approach to Solve Explanation Mismatches. Jornadas Chilenas de Computación. JCC 2003. Chillán, Chile, November 3-9, 2003.
7. Chen, P. The Entity-Relation model- Toward a unified view of data. *ACM Transaction on database systems*, Vol.1(1), March 1976, (pp. 9-36).
8. Codd, E. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, Vol.13(6), 1970, (pp. 377-387).
9. Euzenat, J., Valtchev, P. An integrative proximity measure for ontology alignment. CEUR Workshop Proceedings. Sanibel Island, Florida, October 20, 2003.
10. Gennari, J., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., Noy, N. F., Tu, S. W. The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. Technical Report, SMI-2002-0943, 2002.
11. Haarslev, V. and Moller, R. RACER system description. In P. Lambrix, A. Borgida, M. Lenzerini, R. Moller, and P. Patel-Schneider, editors, Proceedings of the International Workshop on Description Logics, number 22 in CEUR-WS, Linkoeping, Sweden, July 30-August 1 1999, (pp. 140-141).
12. Hasselbring, W. Information System Integration. Communications of the ACM. June 2000.
13. Horrocks, I. The FaCT system. In H. de Swart, editor, Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98, number 1397 in Lecture Notes in Artificial Intelligence, pages 307--312. SpringerVerlag, Berlin, May 1998.
14. Maedche, A. and Staab, S. Measuring Similarity between Ontologies. In: Proc. Of the European Conference on Knowledge Acquisition and Management - EKAW-2002.

Madrid, Spain, October 1-4, 2002. LNCS/LNAI 2473, Springer, 2002, (pp. 251-263)..
15. Rodriguez, A., Egenhofer, M. Determining Semantic Similarity among Entity Classes from Different Ontologies. IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 2, March/April 2003.
16. Rodriguez, A., Egenhofer, M. Putting Similarity Assessments into Context: Matching Functions with the User's Intended Operations. Context 99, Lecture Notes in Computer Science, Springer-Verlag, September 1999.
17. Smith, M.K.,Welty, C., McGuinness, D.L. OWL Web Ontology Language Guide. W3C, http://www.w3.org/TR/2004/REC-owl-guide-20040210/. 10 February 2004.
18. Visser, P., Jones, D., Bench-Capon, T., Shave, M. An Analysis of Ontology Mismatches; Heterogeneity versus Interoperability. AAAI 1997 Spring Symposium on Ontological Engineering.
19. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H.. Schuster, G., Neumann, H. and Hübner, S. "Ontology-based Integration of Information - A Survey of Existing Approaches," In: Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing, Seattle, WA, Vol. (Pages 108-117). 2001.