R_2O, an Extensible and Semantically Based Database-to-ontology Mapping Language

Jesús Barrasa, Óscar Corcho, Asunción Gómez-Pérez

Ontology Engineering Group, Departamento de Inteligencia Artificial,
Facultad de Informática, Universidad Politécnica de Madrid, Spain
`jbarrasa@eui.upm.es, {ocorcho,asun}@fi.upm.es`

Abstract. We present R_2O, an extensible and declarative language to describe mappings between relational DB schemas and ontologies implemented in RDF(S) or OWL. R_2O provides an extensible set of primitives with well-defined semantics. This language has been conceived expressive enough to cope with complex mapping cases arisen from situations of low similarity between the ontology and the DB models.

1 Introduction and Motivations

There is a large quantity of data on Web pages generated from relational DBs. This information is often referred to as the Deep Web [2] as opposed to the surface Web comprising all static Web pages. In this paper we face the problem of "upgrading" this large amount of existing content into Semantic Web content.

Let us set the following scenario: we have a legacy DB and we want to generate Semantic Web content from it. Until now, three approaches have been reported. The first one, described in [11,12] is based in the semi-automatic generation of an ontology from our DB's relational model. Then mappings are defined between the DB and the generated ontology. Because the level of similarity between both is very high, mappings will be quite direct and complex mapping situations do not usually appear. This approach does not allow the population of an existing ontology, which is a big limitation. A second approach [6], proposes the manual annotation of dynamic Web pages which publish DB content, with information about the underlying DB and how each content item in a page is extracted from the DB. This approach does not deal neither with complex mapping situations and assumes we want to make our database schema public, which is not always the case. The third approach is the one proposed in this paper. It tries to map an existing DB to an appropriate existing ontology implemented in RDF(S) or OWL. The term appropriate may mean, depending on the situation: The one whose domain has a higher coverage of the domain modeled in the DB, the one whose domain best covers the specific part of the DB to be migrated to the Semantic Web or the one that maximizes the extraction of information according to a particular metric, among others.

The literature references very few languages for expressing mappings between ontologies and DBs. Recent approaches like D2R MAP [4] and extended D2R [1] have tackled this problem but they lack of expressiveness for writing complex mapping transformations and are not fully declarative. The language presented in this paper is intended to extend and enhance the mapping description capabilities of these two ones.

Complementary approaches to this work can be also found in the Intelligent Information Integration area, in which data from existing heterogeneous DBs are extracted according to ontologies and then combined. The main differences with respect to our approach is that in these systems the mapping between the ontologies and the DBs from which the ontology instances are extracted are not created declaratively but with ad-hoc software implementations. Examples of such systems are Observer [8] and Picsel [5], among others.

The most important aspect of our approach is that we will use the DB and the ontology "as they are" and we will just define a declarative specification of the mappings between their modeling components. That is the reason why the R_2O (Relational to Ontology) language which is the base of our approach, has been conceived expressive enough to cope with complex mapping situations arisen from low similarity between the ontology and the DB model (one of them is richer, more generic or specific, better structured, etc., than the other).

This paper is organized as follows: Section 2 describes the main features of the R_2O language. Section 3 enumerates a set of significant mapping situations covered by the R_2O language. Section 4 provides an informal description of R_2O, together with representative examples of mappings expressed in this language. Section 5 draws some conclusions and gives a glimpse of future trends. The appendix provides the R_2O language grammar in BNF notation.

2 Main Features of R_2O

In this section we present the rationale we followed for defining the R_2O language. We present what is R_2O, what is not R_2O and how to use it.

2.1 The Problem to Be Solved

Our objective is to facilitate the upgrade and extraction of DB content into instances of an RDF(S) or OWL ontology under the assumption that DB and ontology models are possibly different and both pre-exist and are not created specifically for this purpose. The approach taken consists of creating a mapping description document using R_2O with all the correspondences between the components of the DB's SQL schema and those of the ontology. Such mappings are processed automatically by the ODEMapster mapping processor to populate the ontology. The grey area in figure 1 shows the results of the mapping definition and execution.

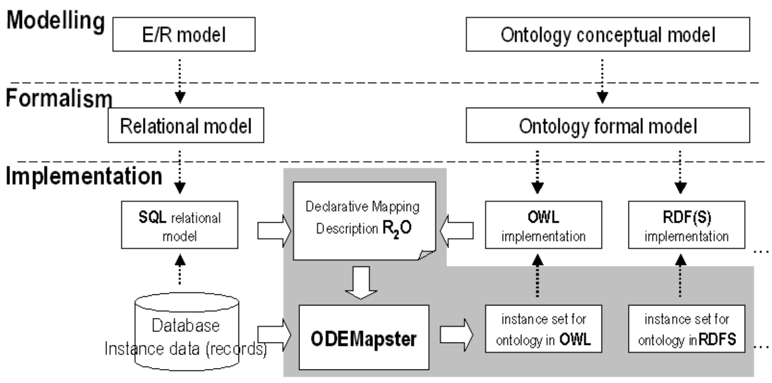
Modelling
E/R model
Ontology conceptual model
Formalism
Relational model
Ontology formal model
Implementation
SQL relational model
Declarative Mapping Description R2O
OWL implementation
RDF(S) implementation
Database Instance data (records)
ODEMapster
instance set for ontology in OWL
instance set for ontology in RDFS

Fig. 1. R2O mapping architecture

2.2 What Is R_2O and What Is Not

R_2O is an extensible, fully declarative language to describe mappings between relational DB schemas and ontologies. It is intended to be expressive enough to describe the semantics of these mappings. R_2O is a RDBMS independent high level language that works with any DB implementing the SQL standard. Its main features are:

1. A R_2O mapping defines how to create instances in the ontology in terms of the data stored in the DB. A R_2O mapping definition can be used to automatically populate an ontology with instances extracted from the DB content. So the intended flow of data is from the DB to the ontology. This can be done as a batch process of massive instance creation or on demand via query translation.
2. R_2O can be used to express mappings generated by existing automatic mapping discovery tools like Cupid [7,10].
3. A R_2O mapping definition can be used for self verification. Due to its fully declarative nature, inconsistencies and ambiguities in the definition of a mapping can be automatically detected.
4. A R_2O mapping definition can also be used to verify the integrity of parts of a DB according to an ontology, applying the ontology's axioms to the DB's elements.
5. A R_2O mapping definition can be used to automatically characterize data sources to allow dynamic query distribution in intelligent information integration approaches.

In general, mappings defined in R_2O language are to be generated and exploited by tools, middleware, APIs, etc., that can carry out tasks like the preceding ones. In this paper we will focus only on functionality number 1.

What follows is instead out of the scope of the R_2O language.

1. R_2O does not aim at defining degrees of similarity between DB elements and ontology components. It only states under which conditions and after what transformations the DB elements are equivalent to the ontology components.

2. R_2O does not define bi-directional transformation functions. The DB elements are transformed to ontology components but not vice versa.
3. Mapping definitions in R_2O are not intended to be short nor compact, nor are they intended to be read by humans.
4. Mapping documents in R_2O are not intended to be generated manually. Graphical user interfaces are to be provided for both browsing and editing R_2O mapping documents.

3 The Mapping Cases

This section presents different mapping situations arising from Database-to-ontology mapping scenarios which are intended to be covered by the R_2O language.

A Database-to-ontology mapping can be defined as a set of correspondences (aka mapping elements) that relate the vocabulary of a relational DB schema with that of an ontology. That is, we want to relate a DB's tables, columns, primary and foreign keys, etc., with an ontology's concepts, relations, attributes, etc.

According to the level of overlap in the domains covered by the DB and the ontology, we distinguish the three cases presented graphically in figure 2.

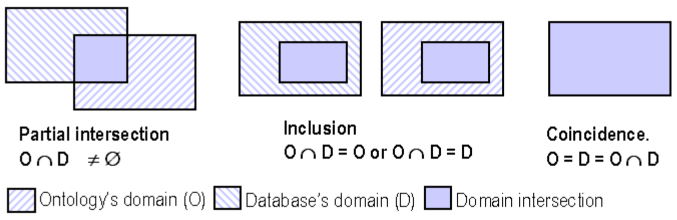

Fig. 2. Levels of overlap between the domains covered by the DB and the ontology

Because domains do not always coincide and because the design modeling criteria used for building the DB are different from those used for ontology creation, the correspondences between their corresponding elements will be sometimes straightforward, sometimes tricky. If we have a look at how components of the DB schema map ontology concepts, we can distinguish, as shown graphically in figure 3, the following cases:

- Case 1. One DB table maps one concept in the ontology. In this case the columns of the table map the attributes and/or relations of the concept, and with each DB table record we generate an instance of the concept. With the data of the record we fill in the attribute values on the instance.
- Case 2. One DB table is used to instantiate more than one concept in the ontology, but only one instance per concept. In this case each column of the table maps the attributes and/or relations of the same or different concepts, and with each DB table record we generate an instance of each concept. With the data of the record we fill in the attribute values on each instance.

– Case 3. One DB table is used to instantiate more than one concept in the ontology, but multiple instances of the ontology can be generated. In this case each column of the table maps the attributes and/or relations of the same or different concepts, and with each DB table record we generate one or more instances of each concept. With the data of the record we fill in the attribute values of the instances.

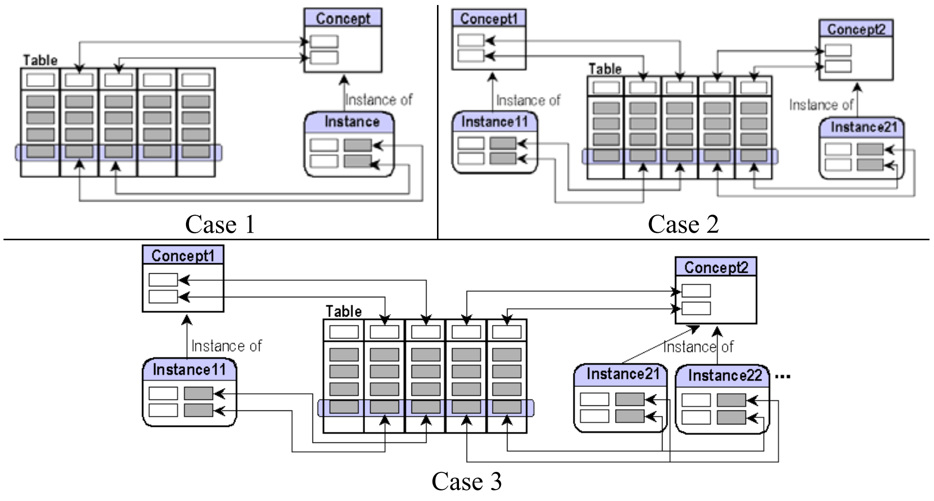

Fig. 3. Mapping cases classification for concepts

It is important to mention that sometimes all the columns in a table map properties of the concepts but sometimes only a few of them are needed. The same happens for records. In both cases, before generating ontology instances, some standard relational algebraic operations (projection, selection, etc.) should be executed. We distinguish the cases presented in figure 4.

– Direct Mapping. A DB table directly maps a concept in the ontology. Every record of the table will correspond to an instance of an ontology concept.

– Join/Union. A set of DB tables map a concept in the ontology when they are joined. Every join record of the joined tables correspond to an instance of an ontology concept.

– Projection. It appears when a subset of the columns of a DB table are needed to map a concept in the ontology.

– Selection. A subset of the rows of a DB table map a concept in the ontology.

– Any combination of them are also possible.

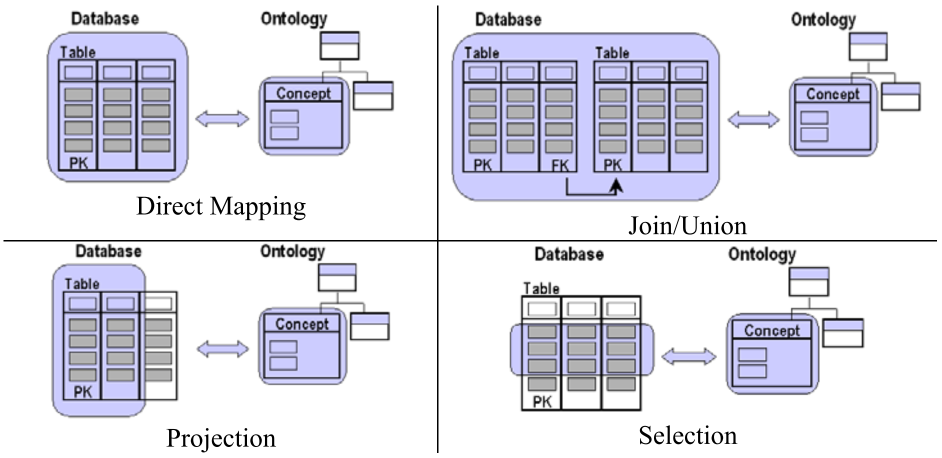

Fig. 4. Mapping cases classification for concepts

The values of the attributes and relations can be filled in directly from the values of the fields in a DB record or after the application of a transformation function. The function can affect more than one data field. Figure 5 presents these ideas.

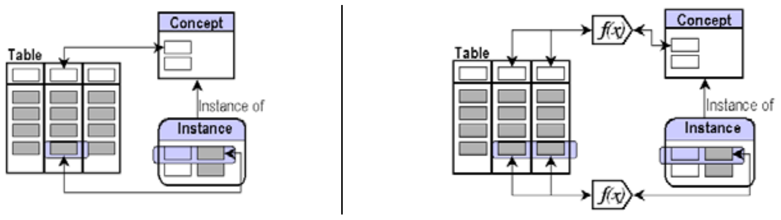

Fig. 5. Mapping cases classification for attributes and relations

Although SQL relational algebra operations covers many cases, there are situations in which some additional transformations might be needed. Examples are more complex operations like natural language processing techniques over text data fields, regular expression matching for dates, URL or email extractions, etc. The R_2O language provides means for specifying declaratively such selections and transformations.

4 The R_2O Language

This section gives an informal description of the R_2O language. To improve readability we use a compact pseudo XML syntax where opening tags are indicated by **bold** text, grouping of sub-content is indicated by indentation and closing tags are omitted. A mapping description in R_2O is a structure made up of several components, some of which may themselves be structures, some are optional, and some may be repeated. We will write **component?** if it is an optional component, **component+** if it is a component that may be repeated one or more times (i.e., that must occur at least once) and **component*** if it is a component that may be repeated zero or more times (i.e., that may be completely omitted). We provide as well examples of the language use.

4.1 A Mapping Description Specified in R_2O

A mapping description in R_2O consist of the following components: A set of instance URIs to be added to the instance set extracted from the DB (**import?),** a description of the DB's schema (**dbschema-description***), one or more ontology URIs for which instances will be generated when executing the R_2O mapping (**ontology+)**, and the list of mapping definitions (**conceptmapping-definition**+) between the components of the DB schema and the ontology.

Table 1. Example of use of a R_2O mapping description

Example of mapping description
import http://www.instancesets.net/instance1
import http://www.instancesets.net/instance2
dbschema-desc <*dbschema-description*>
dbschema-desc <*dbschema-description*>
ontology http://www.ontologies.net/onto1#
ontology http://www.ontologies.net/onto2#

4.2 Description of DB Schemas

A DB schema description **(dbschema-desc)** provides a copy of the main structural elements in the DB's SQL schema. It can be extracted automatically from the source DB and the only elements that need to be added manually are the implicit references. The DB schema definition is a "sort of internal" representation of a DB and will be needed to restrict the domain and range of the components of a mapping definition as will be seen later. Some technical information about the DB (url, port, user/pwd, etc.) necessary for implementation is omitted for the sake of clarity. Table 2 presents an example of use of a DB schema description.

A **dbschema-desc** consists the name of the DB (**name**), a NL description of the schema (**documentation?**), and one or more table descriptions (**hasTable**+) where each DB table is described by means of **(table-desc).**

A table description (**table-desc**) provides a description of a DB table. A **table-desc** consists of a name of the table (**name**), the type of the table (**tableType**) that can be either system table, user table or view, its NL description (**documentation?**), and a set of column descriptions (**column-description**+).

A column description **(column-description)** can be either a key column (**keycol-desc)**, a foreign key column (**forkeycol-desc)** or a non key column (**nonkeycol-desc)**. Any of them consist of a name for the column (**name**), a type for the data it contains

(**ColumnType**), its natural language (NL) description (**documentation?**), and the key column referred **(refers-to?)** if it is a foreign key **forkeycol-desc**.

Sometimes implicit references exist between columns that are not explicitly declared as such in the DB schema, in this case we will also have the referred column (**implicitlyrefers-to?**). If a DB is correctly defined it should not be necessary. We provide this as a solution for badly designed DB schemas.

Table 2. Example of use of a DB schema description

Example of a DB schema description
dbschema-desc
name FISUB
has-table
name FundingOpps
documentation “Stores funding info”
keycol-desc
name FundingOpps.FundId
columnType integer
documentation “Identifies a f.o.”
nonkeycol-desc
name FundingOpps.FundTitle
columnType string
forkeycol-desc
name FundingOpps.FundSector
columnType integer
refers-to Sector.Id
documentation “Points at Sector”
has-table
name Sector
documentation “Productive sectors.”
keycol-desc
name Sector. Id
columnType integer

4.3 Definition of Concept Mappings

This section presents how to define using R_2O the concepts of the ontology in terms of the DB elements. A concept mapping definition **(conceptmap-def)** is equivalent to a *basic mapping expression* as defined in [9]. A concept mapping definition associates the name of a class in the ontology with a description of how to obtain it from the DB. A **conceptmap-def**, as presented in table 3, consists of the following components.

Table 3. Example of use of a concept mapping definition. The concept mapping is identified by a single DB column (transformation and cond-expr are described later)

Example of concept mapping definition
conceptmap-def
name Customer
identified-by Users.userID
uri-as
<*transformation*>
applies-if
<*cond-expr*>
documentation Select all rows from table Users with 'true' in column isPreferential.

- The identifier of a concept (URI of the class) in the target ontology (**name)**
- NL description of the rationale behind the concept mapping (**documentation?**).
- One or more columns that identify **(identified-by**+) the concept uniquely in the DB. Each column is described with the **column-desc** element previously defined.
- A pattern expressed in terms of transformations (see **transformation** elements in section 4.5) describing how URIs (**uri-as**+) for the new instances extracted from the DB will be generated. URIs will normally be obtained from the key columns after application of some transformations. The absence of this element will generate anonymous instances.
- A concept in the ontology is described (**described-by***) by a set of attributes and relations. As we will see in section 4.6 property mapping definition **(propertymap-def)** associates the name of an attribute and/or relation in the ontology with a description of how to obtain them from the DB columns along with the transformations (**transformation**) needed. The URI extraction described in the preceding point is actually a particular case of this.
- A mapping will only be applied under certain conditions. The element **applies-if?** contains a conditional expression (see **cond-expr** in section 4.4) describing these conditions. In other words, it specifies the subset of values from the DB that will be transformed to populate this concept.
- Sometimes more than one table will be implied in the definition of a concept mapping, and join operations will be needed. The optional (**joins-via?**) element describes how these tables are joined in case they use "implicit joins". If this information can be obtained from the DB schema description (only foreign keys are used for joins) the **joins-via?** element will be omitted. The rationale behind this element is that the mapping designer might want to specify a particular join, not valid in all cases but useful in the context of a particular concept mapping (sort of a "specific local join"). The information in the **joins-via?** element can overwrite that in the DB schema definition or be added to it. It will contain a **join-list** which consists of a group of one or more **join** elements, each of them describing a pair of columns (**hasJoin**+) and a flag (**overwrites**) indicating if the join list is to be used together with the ones defined in the DB schema description or if we want them to be overwritten. Columns are not necessarily key nor foreign key columns.

4.4 Describing Conditions and Conditional Expressions

As described in section 3 not all the records in a table generate instances of the concepts in the ontology so we will need to describe under which conditions the mapping takes place. A conditional expression **(cond-expr)** can be either a single condition **(condition)**, or a boolean combination of multiple ones using the operators **AND**, **OR** and **NOT** as presented in table 4.

Table 4. Example of use of a condition expression. The condition is true if the value of column period is “Contemporary” or if the date is after “01/01/1999”

R_2O condition expression example
OR
equals
arg-restriction
on-param value1
has-column Paintings.period
arg-restriction
on-param value2
has-value string “Contemporary”
date-after
arg-restriction
on-param date1
has-column Paintings.date
arg-restriction
on-param date2
has-value date “01/01/1999”

A condition **(condition)** describes an invocation to a single conditional operation defined with the primitives (**primitive-condition**) provided by R_2O and assigns argument values (**arg-restriction***) to each of the parameters required by the particular conditional operation. The core list of R_2O primitive conditional functions is: numerical and string equality (**equals**, **equals_str**), numerically and alphanumerically lower than (**lo_than, lo_than_str**), numerically and alphanumerically higher than (**hi_than, hi_than_str**), the keyword is contained in the string (**in_keyword**), numerically and alphanumerically into a range (**between**, **between_str**), a date precedes, succeeds or is equal to another one (**date_before**, **date_after**, **date_equal**). For each condition R_2O defines: its parameters and their domain types, indicating whether they are needed or optional, as well as descriptions of their use. The complete list of primitive conditional functions is available at http://www.esperonto.net/r2o. An excerpt of this information appears in table 5:

Table 5. Excerpt of the R_2O condition set.

Condition	Params	Domain	Needed	Condition description
Lo_than	value1 value2	float U decimal U double float U decimal U double	Yes default=0	Compares two values numerically. Returns value1<value2

As we mentioned before, **arg-restriction*** is used for assigning values and their types to arguments. Values can be taken typically from a DB table column, issued by a transformation or in the simplest case, be constant values. So, an **arg-restriction** element is defined by means of a parameter name (**on-param**) and the type of argument we want to assign to the parameter. R_2O distinguishes the following types: constants (**has-value?**), a DB table column (**has-column?**), and a transformation (**has-transform?**). So **has-value?** contains a constant value for the parameter, whose type are XML Schema Datatypes; **has-column?** contains a column (previously described as **column-desc**) indicating that values for this formal parameter will be taken dynamically for each row from this DB table column; **has-transform?** contains a **transformation** (see section 4.5) to allow composition of transformations and the use of transformations' results as an input to **conditions**.

4.5 Describing Transformations

As mentioned in section 3, the mapping between DB field values and ontology properties and relations is not always direct. We will need to specify the necessary transformations to be applied to them. A transformation **(transformation)** describes an invocation to a single primitive transformation defined with the primitive (**primitive-transformation**) provided by R_2O and assigns argument values (**arg-restriction***) to each of the parameters required by the particular transformation. Table 6 presents an example of use of a transformation. Note that the **arg-restriction*** element is already defined in previous section.

Table 6. Example of use of a transformation. The transformation concatenates a constant string with the content of two columns(name and IATA)

R_2O transformation example
concat
arg-restriction
on-param string1
has-value string "Coordinates correspond to airport "
arg-restriction
on-param string2
has-transform
concat
arg-restriction
on-param string1
has-column Airports.name
arg-restriction
on-param string2
has-column Airports.IATA

The core list of R_2O primitive transformation (**primitive-transformation**) is: get character at position n (**get_nth_char**), get the string delimited by a particular character (**get_delimited**), get the substring between an upper and a lower limit (**get_substring**), concatenate strings (**concat**), add, subtract, multiply or divide numbers (**add**,**subtract**,**multiply**,**divide**), a constant value (**constant**). In table 7 we define an R_2O transformation by giving the type returned, a list of parameters and their domain types indicating whether they are needed or optional, as well as a description of their use. A complete list can be found at http://www.esperonto.net/r2o.

Arbitrarily complex expressions can be formed through the composition of multiple transformations. This is done by using them as arguments inside other transformations. For instance the expression **concat(get_delimited(**'#'**,**t1.c1**), concat(' -> ', get_nth_word(**'3'**,**t2.c3**)))** gets for each row of table t1 the substring delimited by '#' and '#' in column c1, then gets the third word in column c3 of the same table and then concatenates both results mediating the string ' -> '.

Table 7. Excerpt of the R_2O transformation set.

Transf.	Return	Params	Domain	Needed	Condition description
get_substring	string	str lo_limit hi_limit	string string string	Yes At least one	Extracts the substring between upper & lower limit.

4.6 Attribute and Relation Mappings

A property mapping description (understanding properties as attributes and relations) associates an attribute or a relation belonging to a concept in the target ontology with an expression describing how to obtain it from the DB. Depending on the type of property we deal with and how do we get its values from the DB, these kinds of mappings can either be described with **attributemap-def**, **relfromatt-def** or **relationmap-def**. The first one describes attribute mappings and the rest relation mappings.

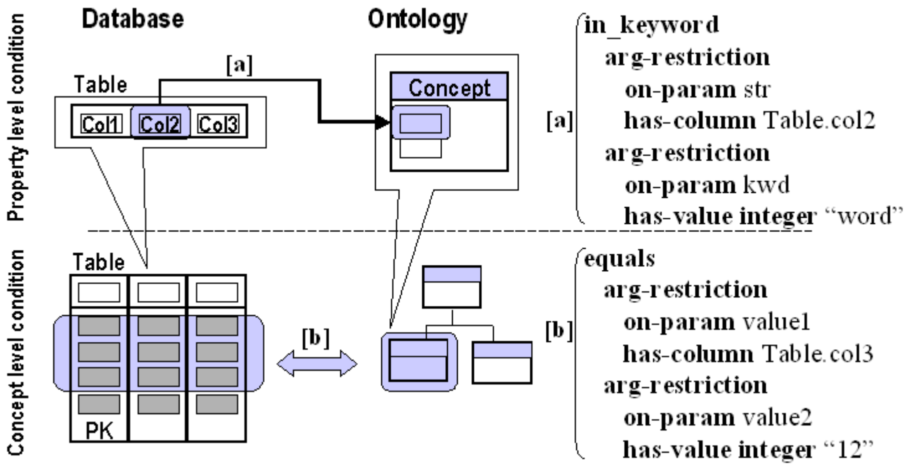

Fig. 6. Mapping cases classification for concepts

We will also add a new level of complexity by adding conditions at the property level. With this, we allow multivaluation for properties as we enhance the language expressivity. This idea is shown in figure 6 but will be explained in detail later.

An **attributemap-def** contains an identifier (**name**) of the property in the target ontology (its URI). To generate its value, we will use zero or more DB columns (previously described with a **column-desc** element) so we declare them with a **use-dbcol?** element. After that, a set of "case type" elements are listed *(case [condition1:action1; condition2:action2…] end-case;)*. Depending on what condition applies, different transformations are performed. This idea is represented by a **Selector?** element which will contain zero or more **applies-if** - **aftertransform** pairs (condition-action).

If the **applies-if** element is missing, it will be considered as true and the transformation will be performed. If the **aftertransform** element is missing a direct mapping will be applied. This situation is explained in detail along with some other notation particularities of R_2O in http://www.esperonto.net/r2o. In the **applies-if?**, a **cond-expr** element describes under which conditions the attribute mapping is applicable, or in other words, which is the subset of values from the DB schema that will be mapped according to the concept matching being defined. Note that the columns appearing in this **cond-expr** can belong to different tables from those stated in the **identified-by** element of the concept mapping definition this property definition belongs to. In this case two situations may arise:

1. If no extra information is provided and the tables containing the columns in the condition are reachable without ambiguities (there is a single foreign key from one table to the other) from those the ones specified in the **identified-by** of the concept mapping description, the join is made automatically.
2. If a table restriction is provided, it will be considered as local to a property mapping definition as opposed to these defined inside a concept mapping definition which are global ones.

The **aftertransform+** element contains the **(transformation)** on the DB columns that participate in obtaining the property being defined. The structure of a **transformation** is that described in the previous section.

The cases in which a data field after a transformation generates a resource would lead to the creation of a relation rather than an attribute. These cases are represented with the **relfromatt-def** element, the structure of which is identical to that of the **attributemap-def** element with the extra element **newObject-type?** Containing the type of the new resource generated with the transformation (if any).

A relation mapping definition (**dbrelationmap-def)** describes how to obtain the target resource of a relation from its corresponding implementation in the DB. Relations in the DB are specified through the use of foreign keys and should be described properly in the **dbschema-desc** part. A **dbrelationmap-def** then consists of an identifier (**name**) of the relation in the target ontology (its URI) and the name (**to-concept**) of the concept mapping element (previously defined as such, and consequently described in terms of some DB tables) to which this property will be a link. This information should be enough to find out the link between tables implied in both the definitions of the source and target concepts of the relation. Additional information on the semantics of attribute and relation mappings and how are they interpreted in R2O is available at http://www.esperonto.net/r2o.

The following examples show a property mapping of each type. The first one uses the **dbrelationmap-def** to define a relation mapping that links a funding opportunity to its productive sector. A link between table *FundingOpps* and Sectors exists because a foreign key has been defined on column *FundingOpps.sector* pointing at *sectorId* primary key in column *Sectors*. The second one uses the **attributemap-def** element to rate a paper as “Interesting” if it is about ontologies and DBs. This verification is based on keyword search on the values of rows of table *Papers* on field *keywords*. The last one uses the **relfromatt-def** to create instances of relation *officiallyAnnounced*. This relation links a funding opportunity to the official publication it is published in. An official publication instance is created for each property instance and its URI is obtained from the *legalRef* column in table *FundingOpportunity* after a simple transformation.

Table 8. Example of use of a property (attribute and relation) mappings

Use of **dbrelationmap-def**
dbrelationmap-def
name hasSector
toconcept Sector

Example of use of **attributemap-def**	Example of use of **relfromatt-def**
attributemap-def	**relfromatt-def**
name paperRating	**name** officiallyAnnounced
selector	**newobject-type** OfficialPublication
applies-if	**selector**
AND	**aftertransform**
in_keyword	**concat**
arg-restriction	**arg-restriction**
on-param string	**on-param** string1
has-column Papers.keywords	**has-value string**
arg-restriction	“http://officialPubs.com/num-“
on-param keyword	**arg-restriction**
has-value string “ontologies“	**on-param** string2
in_keyword	**has-transform**
arg-restriction	**get-delimited**
on-param string	**arg-restriction**
has-column Papers.keywords	**on-param** string
arg-restriction	**has-column**
on-param keyword	FundingOpportunity.legalRef
has-value string “DB“	**arg-restriction**
aftertransform	**on-param** start-delim
constant	**has-value string** “[“
arg-restriction	**arg-restriction**
on-param const_val	**on-param** end-delim
has-value string “Interesting”	**has-value string** “]“

5 Conclusions and Future Work

In this paper we presented R_2O, a Database-to-ontology mapping language, whose strength lies on its expressivity, its declarative nature and on its DBMS and Ontology Language independence. With R_2O we facilitate the “upgrade” of DB content into instances of an ontology under the assumption that DB and ontology models are different and both are existing ones and have not been created specifically for this purpose. The ODEMapster processor presented in [1] has been enhanced to process R_2O documents, and can carry out some of the operations presented in section 2.2.

R_2O has been used in the context of the ESPERONTO project, in particular for the Fund Finder application (http://www.esperonto.net/fundfinder) which is about migrating relational DB content about funding opportunities to the Semantic Web. The DB containing the data was migrated and the ontology was populated with instances extracted from the DB using R_2O and ODEMapster.

Regarding the future trends of our work, intensive testing with other DBs is being carried out and will continue as well as the development of tools, middleware, APIs, etc, to generate and exploit R_2O mapping descriptions. A graphical user interface for both visualizing and writing R_2O mapping documents is currently under development.

References

1. Barrasa J, Corcho O, Gómez-Pérez A. *FundFinder – A case study of Database-to-ontology mapping*. Semantic Integration Workshop, ISWC 2003. Sanibel Island, Florida. Sept 2003
2. Bergman MK. *The Deep Web: Surfacing hidden value*. White paper. Sept 2001
3. Beckett D, Grant J (2003) *SWAD-Europe Deliverable 10.2: Mapping Semantic Web Data with RDBMSes*. Technical report
4. Bizer C. *D2R MAP – A DB to RDF Mapping Language.* 12th International World Wide Web Conference, Budapest. May 2003
5. Goasdoué F, Lattes V, Rousset M (2000) *The Use of CARIN Language and Algorithms for Information Integration: The PICSEL Project.* International Journal of Cooperative Information Systems (IJCIS) 9(4):383–401
6. Handschuh S, Staab S, Volz R. *On deep annotation.* 12th International World Wide Web Conference, Budapest. May 2003
7. Madhavan J, Bernstein P, Rahm E. *Generic Schema Matching with Cupid.* Proceedings of the 27th VLDB Conference. Roma, Italy, 2001
8. Mena E, Illarramendi A, Kashyap V, Sheth AP (2000) *OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies.* International Journal on Distributed and Parallel DBs 8(2):223–271
9. Mena E, Illaramendi A. *Ontology-based query processing for global information systems.* Kluwer Academic Publishers. Pags:86-88. 2001
10. Rahm E, Bernstein P. *A survey of approaches to automatic schema matching.* The VLDB Journal, Volume 10 , Issue 4 (December 2001) pages 334 – 350
11. Stojanovic L., Stojanovic N., Volz R. *Migrating data-intensive Web Sites into the Semantic Web* Symposium on Applied Computing. Madrid, Spain, March 2002
12. Stojanovic N., Stojanovic L., Volz R. *A Reverse Engineering Approach for Migrating Data-intensive Web Sites to the Semantic Web.* Intelligent Information Processing. Montreal 2002

Appendix

In this appendix we provide the BNF notation of the R_2O language grammar, grouped according to the different types of transformations that can be performed with the language.

BNF for R_2O mapping descriptions

(1) r2o::= import? dschema-description+ conceptmapping-definition+ ontology+
(2) import::= **import** literal
(3) ontology::= **ontology** literal
(4) literal::= '<string literal>'

BNF for R_2O DB schema descriptions

(5) dschema-description::= **dbschema-desc** name documentation?
(**has-table** table-desc)+
(6) name::= **name** literal
(7) documentation::= **documentation** literal
(8) table-desc::= name tabletype? documentation? (column-description)+
(9) tabletype::= **tableType** literal
(10) column-description::= (**keycol-desc** | **forkeycol-desc** | **nonkeycol-desc**) name columnType documentation? col-reference? implicit-col-reference?
(11) columnType::= **columnType** datatype
(12) col-reference::= **refers-to** literal
(13) implicit-col-reference ::= **implicitlyrefers-to** literal
(14) datatype::= **string** | **boolean** | **decimal** | **float** | **double** | **date** | **integer** ...
(XML Schema Datatypes)

BNF for concept mapping definitions in R_2O

(15) conceptmapping-definition::= **conceptmap-def** name documentation?
identified-by+ (**uri-as** transformation)?
(**described-by** propertymap-def)*
(**applies-if** cond-expr)? (**joins-via** join-list)?
(16) identified-by::= **identified-by** literal
(17) join-list::= documentation? (**hasjoin** joindesc)+ (**overwrites** literal)?
(18) joindesc::= (**hasCol** literal)+

BNF for condition expressions in R2O

(19) cond-expr::= orcond-expr | **AND** andcond-expr orcond-expr
(20) orcond-expr::= notcond-expr **|** **OR** orcond-expr notcond-expr
(21) notcond-expr::= condition | **NOT** condition
(22) condition::= primitive-condition (**arg-restriction** arg-restriction)*

(23) primitive-condition::= **lo_than** | **loorequal_than** | **lo_than_str** | **loorequal_than_str** | **hi_than** | **hiorequal_than** | **hi_than_str** | **hiorequal_than_str** | **equals** | **equals_str** | **in_keyword** | **in_set** | **in_set_str** | **between** | **between_str** | **date_before** | **date_after** | **date_equal**

(24) arg-restriction::= parameter-selector restriction

(25) parameter-selector::= **on-param** literal

(26) restriction::= **has-value** constant-value | **has-column** literal | **has-transform** transformation

(27) constant-value::= datatype literal

BNF for transformations in R2O

(28) transformation::= primitive-transformation (**arg-restriction** arg-restriction)*

(29) primitive-transformation::= **get_nth_char** | **get _delimited** | **get_substring** | **concat** | **add_type** | **Subtract_type** | **Multiply_type** | **divide_type** | **constant**

BNF for property mappings in R2O

(30) propertymap-def::= attributemap-def | relfromatt-def | relationmap-def

(31) attributemap-def::= **attributemap-def** name use-dbcol* selector* documentation

(32) relfromatt-def::= **relfromatt-def** name use-dbcol* selector* newobj-type? documentation?

(33) relationmap-def::= **relationmap-def** to-concept

(34) use-dbcol::= **use-dbcol** literal

(35) selector::= **selector** (**applies-if** cond-expr)? (**aftertransform** transformation)?

(36) newobj-type::= **newobject-type** literal

(37) to-concept::= **to-concept** literal