BUILDING ONTOLOGIES FROM MULTIPLE INFORMATION SOURCES

Raji Ghawi, Nadine Cullot

Laboratory LE2I, University of Burgundy, 21000 Dijon, France, {first.last}@u-bourgogne.fr

Abstract. Ontologies provide a promised technology to solve the semantic heterogeneity problem. In literature, many works tackle the development of ontologies from scratch or from one information source. But building an ontology from several heterogeneous information sources has not been yet well investigated by researchers. We propose an ontology-evolution-based approach to create an ontology from several heterogeneous information sources, where an initial ontology is evolved progressively by involving an information source in each step. Two types of information sources are considered, relational databases and XML documents. An evolution step includes a set of ontology change operations as well as a set of mapping bridges definitions between the entities of the ontology and the entities of the source being involved. The process of involving an information source in the development of an ontology is semi-automatic, that is, a human intervention is needed to suggest initial correspondences between ontology and information source entities. These correspondences are treated automatically to apply needed changes on the ontology and to elaborate suitable mappings. The automatic treatment of initial correspondences is based on pre-defined rules that determine which are the possible solutions for each possible case of correspondence.

Keywords: Building ontologies, Ontology evolution, Databases, XML.

1 Introduction and Motivation

In order to achieve an efficient integration of distributed heterogeneous information sources, conflicts of these heterogeneous sources have to be solved at syntactic, structural and semantic levels. In particular, semantic interoperability of information sources became critical issue. Ontologies provide a promised technology to solve the semantic heterogeneity problem, because they allow the representation of common semantics of the domain of discourse. Ontology has become one of the hottest issues of research among different communities (Artificial Intelligence, Databases, Semantic Web, etc.). An Ontology is defined as an "explicit specification of a conceptualization" [11], therefore, it can be used to describe the semantics of the information sources and to make their content explicit. In ontology-based information integration approaches, ontologies are mainly used for the explicit description of the information source semantics. According to the way of how ontologies are employed in information integration, three different approaches can be distinguished: single ontology approach, multiple ontologies approach and hybrid approach. Single ontology approach uses only one global ontology. All information sources are linked to this global ontology by relations expressed via mappings that identify the correspondence between each information source and the ontology. In multiple ontologies approach, each information source is described by its own ontology and inter-ontology mappings are used to express the relationships between the ontologies. The hybrid approaches combine the two previous approaches. Each information source has its own ontology and the semantics of the domain of interest as a whole is described by a global reference ontology. In these approaches there are two types of mappings: mappings between an information source and its local ontology and mappings between local ontologies and the global ontology.

In addition to content explication, ontologies can also be used as means to interrogate information integration systems. That is, the user formulates a query in terms of the ontology, then the integration system decomposes the global query into sub-queries for each appropriate source, collects and combines the query results, and returns the results.

Whatever the role of ontologies, the important question that arises is how to obtain them? The development of ontologies has been investigated by many researchers. In the literature, two major approaches of ontology development have been addressed, 1) ontology construction from scratch, and 2) ontology development using existing resources. It became clear that developing ontologies from scratch is costly and difficult [2,18]. Therefore, many development methodologies (such as [17,6]) include a phase of integration of existing ontologies, in which reusable components are imported as much as possible.

However, in the context of semantic interoperability, where ontologies are used to represent the semantics of underlying data sources (that may be relational databases or XML documents), it is crucial to involve the underlying information sources in the process of ontology development. In the literature, there are many approaches that handle the creation of an ontology from one existing information source (see section 2).

It is often necessary to involve more than one information source to build an ontology. We think that the creation of an ontology from several heterogeneous information sources has not been yet well investigated by

researchers. To our knowledge, there is no "suitable" approach to handle such process. The aim of this paper is to propose an integrated approach to build an ontology from several heterogeneous information sources.

During the creation of an ontology from information sources, an important point has to be taken in account which is the mappings between the created ontology and the underlying sources. The operations to elaborate and update these mappings have to be specified and handled. An appropriate mapping specification language is also needed to describe the elaborated mappings.

In this paper, we propose an ontology-evolution-based method to build an ontology from several heterogeneous information sources. This method reuses several techniques from various fields: database-to-ontology mapping, XML-to-ontology mapping, and ontology change management. Because of the lake of space, we only present the side of our approach concerning databases, while the side of XML is omitted. In section 2, we give a background on the fields of database-to-ontology mapping and ontology change management. In section 3, we give an overview of our proposed approach, whereas in section 4, we present some details on involving a database in our ontology-evolution-based approach to build ontologies. Section 5 concludes the paper.

2 Background

Ontology development often requires involving information sources. In particular, relational database and XML documents gain significant importance as dominant means to store information. On the one hand, databases are widely recognized as the backbone of information systems. Specifically, relational databases are the most widely used due to their simplicity and the mathematical formal theories that support them. On the other hand, XML brought interoperability at a syntactic level, and has reached a wide acceptance as data exchange format. In the following, we review some existing approaches of developing ontologies from relational databases.

2.1 Database-to-Ontology Mapping

The database-to-ontology mapping approaches can be classified into two main categories, 1) approaches that create an ontology from a database, and 2) approaches that map a database to an existing ontology.

2.1.1 Ontology creation from a database

The objective of these approaches is the creation of an ontology from a relational database and the migration of the contents of the database to the generated ontology. The mappings here are simply the correspondences between each created ontology component (concept, property, etc.) and its original database component (table, column, etc.). In these approaches, the database model and the generated ontology are very similar. Mappings are quite direct and complex mapping situations do not usually appear. In some approaches, the ontology creation process is straightforward, i.e., direct transformations of database tables and columns into ontology concepts and properties respectively. In other approaches, the creation involves the discovery of additional semantic relations between database components (such as the referential constraints) and takes them into account while constructing ontology concepts and properties.

Stojanovic et al. in [16] propose an approach to generate ontologies from relational databases based on pre-defined mapping rules. This approach takes in account different types of relationship between database tables, where mapping rules are defined for different situations. When several rules can be applied, a user intervention is needed to choose the most suitable rule.

DataGenie [9] is a tool that allows the automatic generation of an ontology from a relational database. This generation process is simple and direct, tables are transformed to classes, columns are transformed to properties, and foreign keys are transformed to object properties.

Relational.OWL [5] is an OWL ontology representing abstract components of a relational schema. The schema of any relational database can be described as an instance of this ontology and then be used to represent the data stored in that specific database.

2.1.2 Mapping a database to an existing ontology

In these approaches, the goal is to establish mappings between an existing ontology and a database, and/or populate the ontology with the database contents. In this case, the mappings are usually more complex than those in the previous category, because the modeling criteria used for designing databases are different from those used for designing ontology models [1], thus different levels of overlap (not always coincidence) can be found between the database domain and the ontology one.

KAON Reverse[1] is a tool for mapping relational database content to ontologies. Firstly, a set of mapping rules are defined manually to describe the relation between the database schema and the ontology structure, then according to these, the instances are automatically exported to the ontology. There are two principal types of mappings: 1) *Table Mapping* that relates a table to a concept, and 2) *Column Mapping* that relates a table column to an attribute or to a relation. A column mapping can only be defined in the context of a table mapping.

Vis-A-Vis [12] is a tool that allows the mapping of relational databases to existing ontologies. A mapping is done by adding a new property to an ontology class. This property consists of an SQL expression which will be executed and return the dataset corresponding to the ontology class. This tool also performs a set of consistency checks to insure the validation of mappings. For example, two disjoint classes cannot have mappings to two datasets having common records.

2.1.3 Database-to-ontology mapping specification

Several languages have been proposed to formally express database to ontology mappings:

D2RQ [3] is a RDF-based declarative language to describe mappings between relational database schemas and OWL/RDFS ontologies. The D2RQ Platform uses these mappings to enable applications to provide an RDF-view on a non-RDF database. In D2RQ, basic concept mappings are defined using class maps that assign ontology concepts to database sets. A class map specifies how instances of the class are identified, and it has a set of property bridges, which specify how the properties of an instance are created. Object property bridges should have a primitive that indicates the related concept bridge, and a join primitive that indicates how the related tables have to be joined together.

R2O [1] is another declarative language that describes mappings between database schemas and ontologies. It is more expressive than D2RQ as it provides an extendable set of condition and transformation primitives. After the manual generation of a R2O document, it is processed by a generic query engine, called ODEMapster, that automatically populates the ontology with instances extracted from the database content. This operation can be done in two modes: 1) query driven, i.e. parsing a specific query and translating its result, or 2) massive dump, i.e. creating a semantic RDF repository and translating the full database to it.

2.2 Ontology Change Management

The problem of modifying an ontology in response to a certain need is known in the literature as the *ontology change* problem. Flouris et al. [8] identify and distinguish several facets of this problem such as: ontology alignment, merging, mapping, evolution and versioning. In particular, they define the ontology evolution as "*the process of modifying an ontology in response to a certain change in the domain or its conceptualization*" [7].

When an ontology is evolved, it is very often required to determine whether the new version of the ontology is compatible with the old version. In databases, compatibility usually means the ability to access all of the old data through the new schema (the change does not cause any loss of instance data). Noy et al. distinguish in [13] several dimensions of compatibility of ontologies, namely; 1) instance-data preservation, where no data is lost in transformation from the old version to the new one, 2) ontology preservation, where a query result obtained using the new version is a superset of the result of the same query obtained using the old version, and 3) consequence preservation, if an ontology is treated as a set of axioms, all the facts that could be inferred from the old version can still be inferred from the new version.

Noy et al. also distinguish two modes of ontology evolution: traced and untraced evolution [13]. In the traced mode, the evolution is treated as a series of changes in the ontology. After each change operation, we consider the effects on the concerned parts (the instance data, related ontologies and applications) depending on the used dimension of compatibility. The traced mode is quite similar to schema-evolution.

Stojanovic et al. present in [15] a complete ontology evolution process consisting of six phases. In the *change capturing* phase, the changes to be performed are identified. In the *change representation* phase, the changes are formally represented. In order to guarantee the validity of the ontology at the end of the change process, any problems that will be caused when the required changes are actually implemented have to be identified and addressed. This is the role of the *semantics of change* phase. In the *change implementation* phase, the changes are physically applied to the ontology. The *change propagation* phase follows, where the implemented changes need to be propagated to all concerned parts such as the instances, the dependent ontologies and applications. Finally, the ontology engineer can review the changes and possibly undo them during the *change validation* phase.

There are two major types of changes, 1) atomic (elementary) changes which are simple and fine-grained such as the addition of a concept, and 2) complex (composite) changes that are more coarse-grained. It is

[1] http://kaon.semanticweb.org/alphaworld/reverse/view

possible to replace a complex change by a series of atomic changes, but it is not generally appropriate because this may cause undesirable side-effects [15].

Change operations can also be classified according to the change effects with respect to the instance-data–preservation dimension of compatibility [13], i.e. whether instance data can still be accessed through the changed ontology. Three operations effects are distinguished, 1) information-preserving changes, no instance data is lost, 2) translatable changes, no instance data is lost if a part of the data is translated into a new form, and 3) information-loss changes, it cannot be guaranteed that no instance data is lost. Usually, addition operations are information-preserving, removal operations are information-loss, merge and split operations are translatable changes.

3 Building Ontologies from Several Information Sources

Ontology building from several information sources is the process of creating a target ontology model by combining the components of several overlapped source models, and establishing semantic bridges between the components of the created ontology and their corresponding components of information sources. As we have seen in the previous section, there are several approaches that address the problem of building one ontology from one information source (a database or an XML document). However, to our knowledge, there is not yet any works in the literature that address the problem of creating an ontology from several heterogeneous information sources.

We think that two ways are intuitively possible to tackle this problem, 1) ontology merging based approach, and 2) schema merging based approach.

In *ontology-merging-based* approach, an ontology (called partial) is independently created from each information source using existing available tools. Then, all partial ontologies are merged consecutively for giving the final ontology. There are several approaches and tools for merging ontologies, we refer to [14] for a survey on this field. In this approach, mappings are elaborated between each source and the final ontology using existing mapping tools.

In *schema-merging-based* approach, the schemas of all information sources are merged consecutively, using existing schema-merging approaches, to give a unified schema. Then, the target ontology is created from this unified schema. The suitable mappings are elaborated between each source and the final ontology using existing mapping tools.

However, in this paper we propose an alternative approach to create an ontology from several heterogeneous information sources that is based on ontology evolution.

3.1 Ontology-Evolution-based Approach

In our ontology-evolution-based approach, the target ontology evolves each time a new source is added. The process starts with an initial ontology that can be an existing 'autonomous' ontology or created from the first source[2]. Then, each time a 'new' information source is involved, the current ontology is evolved by adding new concepts and properties from the involved source. We will use the term 'evolution step' to denote the evolution of the ontology from its old version to a new version when an information source is involved. An evolution step can be seen as the process of mapping of an information source to an existing ontology, but with a change of the ontology when necessary. That is, the following general strategy is used:

- If a source entity (database table or column, etc.) corresponds to an ontology entity, then we establish a mapping bridge between them, and we do not change the ontology.
- If a source entity has no corresponding ontology entity, then we change the ontology by creating an entity that corresponds to the source entity, and we establish a mapping bridge between the source entity and the ontology entity that we created.

Figure 1 illustrates the evolution step. Firstly, a human expert indicates initial correspondences between the entities of the information source being involved and the entities of the current ontology. Then, the initial correspondences are automatically treated according to pre-defined rules. The result of this treatment is a set of ontology change operations that are required to modify the old version of the ontology towards the new version, as well as, a set of mapping bridges between the entities of the ontology (the new version) and the entities of the information source.

The mapping bridges are saved to a mapping document that describes the relationship between the new version of the ontology and the current information source. Furthermore, it is necessary to review and update the mappings between the ontology and the previously involved sources in order to adjust these mappings according to any changes occurred in the ontology.

[2] As mentioned in section 2.1.1, there are several approaches to create an ontology from a single database. However, we use our own tool, called DB2OWL, for this purpose [4,10].

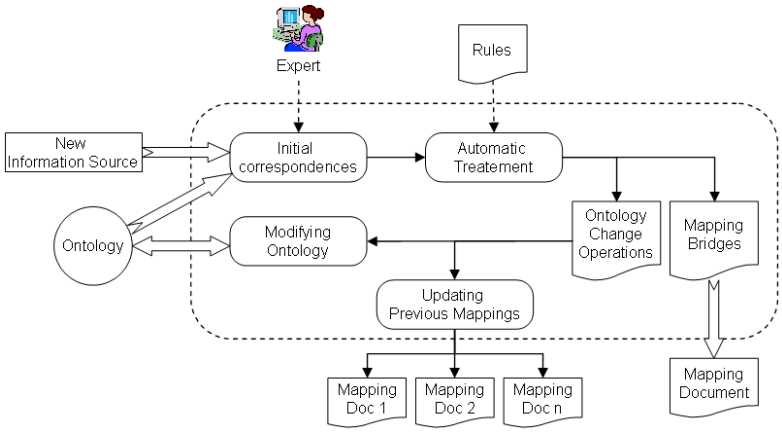

Figure 1. Overview of an evolution step

In order to keep all the mapping documents (the current and the previous) consistent with the ontology, the ontology change operations are executed in a transactional mode. A transaction is composed of an ontology change operation, a set of related mapping bridge definitions and a set of depending previous-mappings updates. Thus, when a transaction is committed, the ontology change operation is executed, its related mapping bridges are saved to the mapping document, and its depending previous-mappings updates are applied on the concerned previous-mappings documents.

In our approach, the evolved ontology is expressed using OWL language, which is a W3C recommendation for publishing and sharing ontologies on the web. We use a mapping specification similar to D2RQ [3]. We extended it with additional primitives to express transformations, compositions and conditions. Mapping documents are written in XML, and they are used for query resolution purposes (this topic is beyond the scope of the paper).

In the following section, we present the set of ontology change operations that we use in our approach, as well as their effects on the ontology and on the previously established mappings.

3.2 Ontology Change Operations

In our approach to build ontologies by involving information sources, the target ontology is instance-free, that is, data instances still reside in their sources and are not exported to the target ontology. Instead, the relationships between the entities of the ontology and the entities of each information source are expressed via mapping bridges. Therefore, no ontology population by instances is performed, but a query-driven approach is used where data instances are retrieved from their sources only for query answering purposes using the elaborated mapping bridges.

Consequently, we replace the information-preservation dimension of compatibility by a similar dimension which is the mapping-preservation dimension of compatibility. We say that two versions of ontology are compatible if the mappings between the old version and information sources are still valid for the new version. According to this dimension, we distinguish three types of ontology change operations;

1. Mapping-preservation change operations; new mapping bridges may be added but existing bridges are not changed.
2. Translatable change operations; existing bridges are changed but not lost.
3. Mapping-loss change operations; existing bridges may be lost.

The ontology change operations that we will need in our approach to build ontologies by involving information sources are seven atomic operations and two complex ones. These few operations are sufficient to handle heterogeneity situations that may be encountered between the ontology and the information sources (see section 4). The atomic operations are the following:

- The addition of a new concept to the ontology (*Add_Concept*).
- The addition of a new datatype property to an existing concept (*Add_DatatypeProperty*).
- The addition of a new object property to an existing concept (*Add_ObjectProperty*).
- The setting of a concept as a subconcept of another one (*Set_SubConceptOf*).
- The setting of a property as a subproperty of another one (*Set_SubPropertyOf*).
- The setting of two ‘object’ properties as inverse of each another (*Set_InverseOf*).
- The removal of an existing property from the ontology (*Remove_Property*).

Except for the last one, all these operations are mapping-preservation, since they may add new mapping bridges but do not affect existing ones. The *Remove_Property* operation is mapping-loss because it causes the loss of any mapping bridges concerning the removed property.

The complex operations are :

- The replacement of an existing datatype property by two (or more) ones (*Split_Property*).
- The conversion of a datatype property into an object property (*Convert_Property*).

These complex operations can be replaced by a series of atomic changes. The *Split_Property* operation can be replaced by a set of *Add_DatatypeProperty* and one *Remove_Property* operations. The *Convert_Property* operation can also be replaced by one *Add_ObjectProperty* and one *Remove_Property* operations.

However, since both replacements include the *Remove_Property* operation which is mapping-loss, we do not do these replacements because both complex operations would become mapping-loss, too. Instead, we use an approach that let both *Split_Property* and *Convert_Property* become translatable change operations. In fact, mapping update transactions are embedded into the complex operations in order to modify the existing mapping bridges instead of remove them. The update of previous mappings is beyond the scope of this paper.

In the next section we present in details the process of involving a relational database in an ontology, as well as, the different cases that arise from the heterogeneity between them.

4 Ontology Evolution by Involving a Database

A relational database schema consists of a set of tables, each of them contains a set of columns. Tables may relate to each other using primary – foreign key relationships. The method we present here combines several techniques from several research fields:

1. Building an ontology from a database (DB2OWL [4,10])
2. Mapping an existing ontology to a database (D2RQ [3])
3. Ontology change management ([15])

In this section, we present a semi-automatic process to involve a database schema in an ontology. This process is supervised by a human expert, who firstly suggest initial correspondences between the ontology and the database entities. There are different cases of initial correspondences that can be suggested by the human expert. We defined a set of rules that associates each correspondence case with its possible solutions. Each solution consists of one or more ontology change operations and one or more mapping bridges.

The initial correspondences are treated automatically to apply needed changes on the ontology and to elaborate the suitable mappings. The automatic treatment of initial correspondences is based on pre-defined rules that determine which are the possible solutions for each correspondence case. Different possible solutions lead to different evolution strategies. Thus an intervention of the expert is needed again in order to validate one of the possible evolution strategies. The whole process consists of two main phases :

1. Evolving ontology concepts using database tables .
2. Evolving ontology properties using database columns.

4.1 Phase 1: Evolving Ontology Concepts Using Database Tables

In this phase, the ontology engineer identifies initial correspondences between the ontology concepts and the database tables. Then, these correspondences are automatically treated according to the rules presented below. The result of this treatment is a set of ontology change operations and a set of mapping bridges between ontology concepts and database tables. The validation of these operations and mappings is required to go on with the next phase.

In the following, we present some rules to solve different possible cases of correspondences between concepts and tables. For each case, the solution is some possible ontology change as well as some mapping bridges definitions that should be added to the mappings document. The different main cases that may be encountered are the following:

1. A concept C has no corresponding table T.
2. A concept C corresponds to exactly one table T.
3. A table T has no corresponding concept C.
4. A concept C corresponds to several tables T_1, T_2, …, T_m.
5. A table T corresponds to several concepts C_1, C_2, …, C_m.
6. Several concepts C_1, C_2, …, C_n correspond to several tables T_1, T_2, …, T_m.

Those cases are presented below with thier solutions:

Case 1 : A concept C has no correspondent table in the database. In this case, nothing should happen, the ontology does not change, and nothing is added to the mappings document.

Case 2 : A concept C corresponds to exactly one table T. In this case, the ontology does not change, but a concept bridge between C and T is added to the mappings.

Case 3 : A table T has no corresponding concept C. For this case, we will use the approach followed in our DB2OWL tool [4,10] where several cases of tables are distinguished: 1) a table is used to relate two other tables in many-to-many relationship, 2) a table is related to another table by a foreign key which is also a primary key (inheritance), and 3) other tables (default case). According to this distinction, we present three solutions for these three sub-cases (see Figure 2).

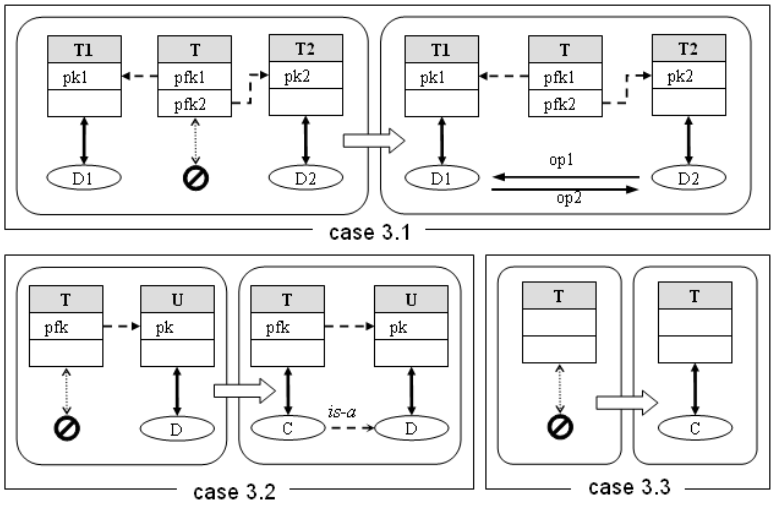

Figure 2. Phase 1: Case 3: A table T has no corresponding concept C

Case 3.1 : A table T is used only to relate two other tables T_1 and T_2 in a many-to-many relationship. T_1 and T_2 are supposed to be already mapped to two concepts D_1 and D_2. In this case, no concept is created, but two inverse object properties between D_1 and D_2 are added to the ontology (see the case 3.1 in Figure 2).

Case 3.2 : A table T is related to another table U by a foreign key which is also a primary key. In this case, a new concept C is added to the ontology and a concept bridge between T and C is added to the mappings. In addition, U is supposed to be already mapped to some concept D, therefore, C is set to be a sub-concept of D (see the case 3.2 in Figure 2).

Case 3.3 : A table T is in the default case. A new concept C is added to the ontology and a concept bridge between T and C is added to the mappings (see the case 3.3 in Figure 2).

From an algorithmic point of view, the treatment should start by the default case (case 3.3) because it is needed for other cases, then the case of inheritance (case 3.2) is treated, and finally the case of many-to-many relationship (case 3.1). In case 3.2 and case 3.3, when a concept is created from a table, the suitable properties of the concept are created from the table columns in the phase 2.

Case 4 : A concept C corresponds to several tables T_1, T_2, …, T_m. According to previous cases, tables T_1, T_2, …, T_m should be mapped to some (existing or created) concepts C_1, C_2, …, C_m. However, two sub-cases are distinguished according to the type of correspondence: union and join (see Figure 3).

Case 4.1 (union): One concept C corresponds to the union of several tables T_1, T_2, …, T_m. The solution consists in setting these concepts as sub-concepts of C. The common properties are moved to the super-concept C (see the case 4.1 in Figure 3).

Case 4.2 (join): One concept C corresponds to the join of several tables T_1, T_2, …, T_m. In general, the join of tables does not correspond to a concept, except if the join is based on primary keys on both tables. The solution of this case consists in setting these concepts to be super-concepts of C (multiple inheritance) (see the case 4.2 in Figure 3).

Case 5: A table T corresponds to several concepts C_1, C_2, ..., C_m. In this case, a new concept C is added to the ontology and a concept bridge between T and C is added to the mappings. In addition, concerning concepts are set to be subconcepts of C, and common properties are moved to the super-concept C (see Figure 4).

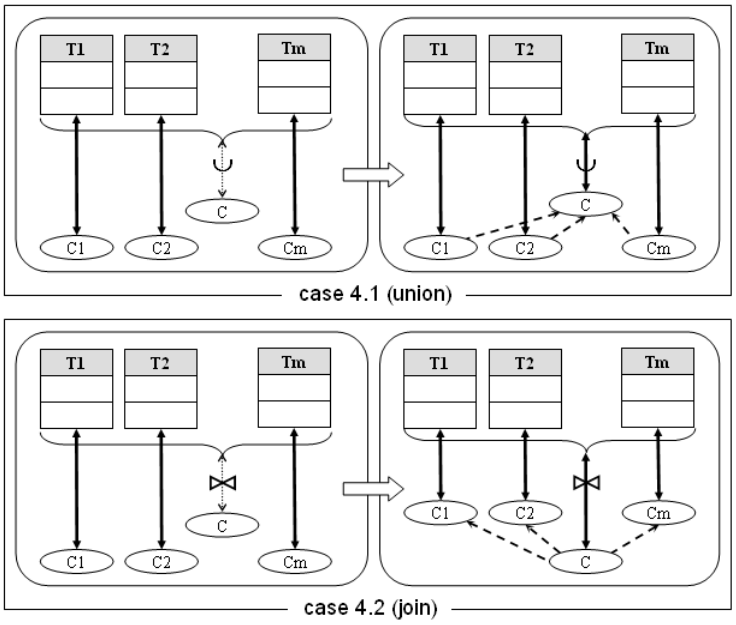

Figure 3. Phase 1: Case 4: A concept C corresponds to several tables T_1, T_2, …, T_m

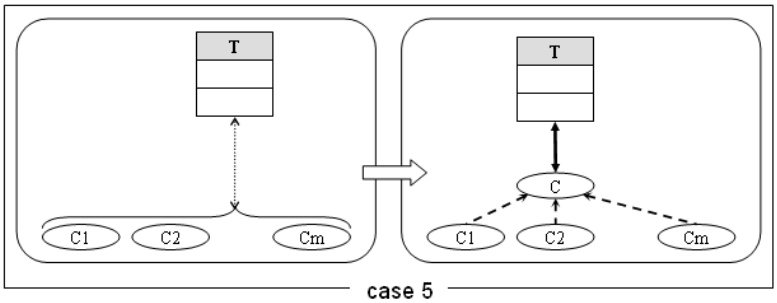

Figure 4. Phase 1: Case 5: A table T corresponds to several concepts C_1, C_2, …, C_m

Case 6 : Several Concepts C_1, C_2, …, C_n correspond to several tables T_1, T_2, …, T_m. Although this case could not happen in real world, but if it occurs, it can be decomposed into simpler cases. Either we consider that each concept C_i corresponds to several tables T_1, T_2, …, T_m, so we have *n* occurences of the case 4. Or we consider that each table T_j corresponds to several concepts C_1, C_2, …, C_n , so we have *m* occurencs of the case 5.

At the end of this first phase, each table is mapped to one or more concepts (except the particular case 3.3). These mappings are expressed using concept bridges.

4.2 Phase 2: Evolving Ontology Properties Using Columns

After evolving ontology concepts using database tables, the next phase can start, where the ontology engineer identifies initial correspondences between ontology properties and the database columns. As in the previous phase, the initial correspondences are automatically treated to generate a set of ontology change operations and a set of mapping bridges.

There are different cases of initial correspondences between properties and columns. The distinction between these different cases is based on several criteria such as, the kind of the property (datatype or object property), the kind of the column (primary, foreign key or non-key column), the number (cardinality) of corresponding entities (properties/columns), and whether the correspondence is direct or using transformations.

We use the concept bridges established in the first phase to classify the initial correspondences. When a concept C corresponds to a table T (a mapping bridge exists between them), we can distinguish three categories of correspondence cases:

1. Correspondences between C properties and T columns.
2. Correspondences between C properties and columns in other tables.
3. Correspondences between T columns and properties of other concepts.

Category 1

In this category, we have a concept bridge between a concept C and a table T. We can distinguish the following seven cases of correspondences between C properties and T columns:

Case 1 : A (datatype or object) property *prop* has no corresponding column (neither in T nor in any other table). In this case, nothing should happen, the ontology does not change, and nothing is added to the mappings document.

Case 2 : A column *col* has no corresponding property in C. We distinguish two sub-cases. Firstly, if the column *col* is not a foreign key (case 2.1), then a datatype property *dp* corresponding to *col* should be added to the ontology. A property bridge between *dp* and *col* is thus added to the mappings (see case 2.1 of Figure 5).

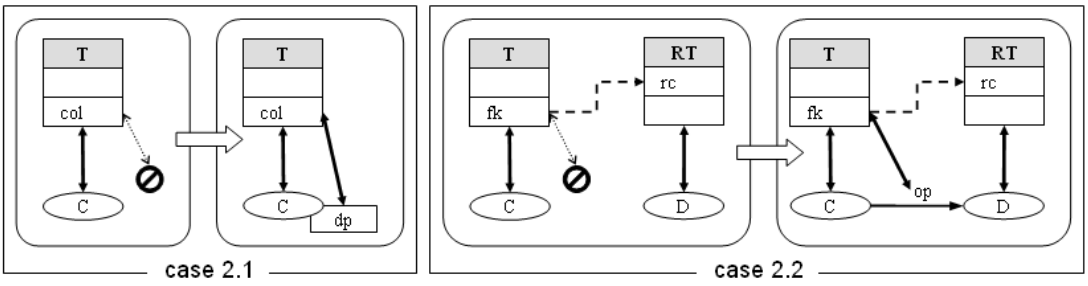

Figure 5. Phase 2: Case 2: A column has no corresponding property

The second case (case 2.2) occurs when the column *col* is a foreign key *fk* referring to a column *rc* in another table *RT*. We assume that the table *RT* is already mapped (in phase 1) to some (existing or created) concept *D* (there is a concept bridge between *D* and *RT*). The solution here is to add to the ontology an object property *op* whose domain is *C*, and whose range is *D*. This new object property corresponds to the foreign key *fk*, therefore, a property bridge between *op* and *fk* is added to the mappings (see case 2.2 of Figure 5). This property bridge should :

- belong to the concept bridge between *C* and *T*,
- refer to the related concept bridge which is between *D* and *RT*,
- refer to the join expression that indicates how *T* and *RT* tables have to be joined together (*T.fk* = *RT.rc*).

Case 3 : A datatype property *dp* corresponds directly to one column *col*. In this case, the ontology does not change, and a property bridge between the property *dp* and the column *col* is added to the mappings.

Case 4 : An object property *op* (from C to another concept D) corresponds directly to a foreign key *fk* referring to a column *rc* in another table *RT*. We assume that the table *RT* is already mapped (in phase 1) to some (existing or created) concept *F*. We distinguish two sub-cases. The first case (case 4.1) occurs if, fortunately, this concept *F* is the concept *D* (the range of *op*), then the object property *op* is mapped to the foreign key *fk*, therefore, a property bridge between *op* and *fk* is added to the mappings (see case 4.1 of Figure 6). This property bridge should :

- belong to the concept bridge between *C* and *T*,
- refer to the related concept bridge which is between *D* (which is *F*) and *RT*,
- refer to the join expression that indicates how *T* and *RT* tables have to be joined together (*T.fk* = *RT.rc*).

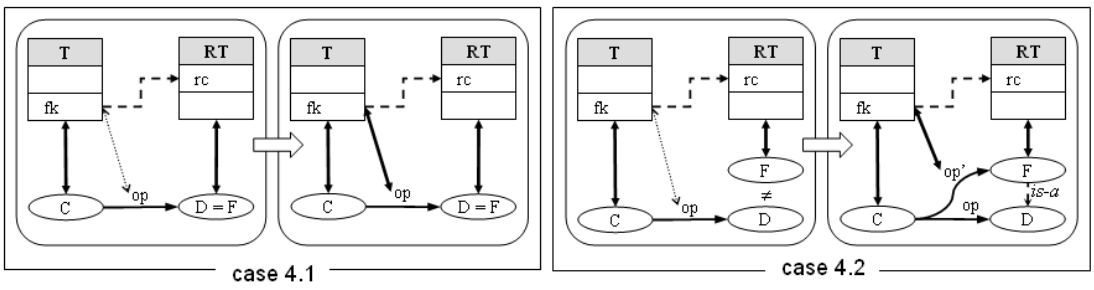

Figure 6. Phase 2: Case 4: An object property corresponds directly to a foreign key

The second case (case 4.2) occurs if the concept *F* (mapped to *RT*) is not *D* (the range of *op*), then *F* is set to be a sub-concept of *D*, and an object property *op'* is created from *C* to *F*, and it is set as sub-property of *op*.

The new object property *op'* is mapped to the foreign key *fk*, therefore, a property bridge between *op'* and *fk* is added to the mappings (see case 4.2 of Figure 6). This property bridge should :

- belong to the concept bridge between *C* and *T*,
- refer to the related concept bridge which is between *F* and *RT*,
- refer to the join expression that indicates how *T* and *RT* tables have to be joined together (*T.fk = RT.rc*).

For example, let us consider a concept *Conference* that corresponds to a table *conference*. The table *conference* is related to another table *city* via the foreign key *venue*. The table *city* is mapped to a concept *City*. An object property *hasLocation* relates the concept *Conference* with another concept *Location*. The ontology engineer determined that the object property *hasLocation* corresponds to the foreign key *venue*. The solution is to set *City* as a sub-concept of *Location*, create an object property *hasCity* from *Conference* to *City*, set *hasCity* as sub-property of *hasLocation*, and make a property bridge between *hasCity* and *venue*.

Case 5 : A datatype property *dp* corresponds to a transformation *Trans* of one column *col*. In this case, the ontology does not change, and a single transformation property bridge between the property *dp* and the transformation *Trans* of the column *col* `propertyBridge(dp, Trans(col))` is added to the mappings. For example, a datatype property *fahrenheitTemp* corresponds to a column *celsiusTemp*. The transformation could be an external function *fahrenheit2Celsius* that converts temperature from Fahrenheit to Celsius measures. The property bridge looks like : propertyBridge (*fahrenheitTemp*, fahrenheit2Celsius(*celsiusTemp*)).

Case 6 : A datatype property *dp* corresponds to a combination/transformation *Trans* of multiple columns col_1, col_2, … col_n. For example, a datatype property *name* (single string) corresponds to the combination of two columns *firstName* and *lastName*. This combination is expressed using a transformation *concatSpace* that concatenate its first argument, a space and its second argument. For this case, there are two possible solutions:

1. We do not change the ontology, but we add a multiple transformation property bridge propertyBridge(*dp*, *Trans*(col_1, col_2, … col_n)) to the mappings. For example : propertyBridge(*name*, concatSpace(*firstName*, *lastName*))
2. We split the datatype property into *n* datatype properties dp_i each of them corresponds directly to one column col_i . The *Split_Property* operation has to determine how each resulting datatype property is related to the old one. For example, when we split the *name* datatype property into *firstName* and *lastName* datatype properties, we may say :

```
Split_Property(name AS
     firstName = name2FN(name),
     lastName = name2LN(name))
```

where *name2FN* (respectively, *name2LN*) is a transformation giving the first (respectively, the last) name from a full name.

Case 7 : A column *col* corresponds to a combination/transformation of multiple datatype properties. In this case, the ontology is not changed, but a suitable single transformation property bridge is added to the mappings for each datatype property. For example, the column *startingTime* has the datatype datetime and it indicates the date and the time of the start of an event, so it corresponds to the combination of two datatype properties *hasStartDate* and *hasStartTime*. Consequently, the following property bridges is added to the mappings :

```
propertyBridge(hasStartDate, datetime2date(startingTime))
propertyBridge(hasStartTime, datetime2time(startingTime))
```

Category 2

As mentioned above, we identify correspondence cases according to the concept bridges established in the first phase to classify the initial correspondences. When we have a concept bridge between a concept C and a table T, and when a C property corresponds to a column in another table U, then we are in the second category. In this category, we can distinguish the follwing cases concerning datatype properties:

1. A datatype property *dp* corresponds directly to one column *col* in another table *U*.
2. A datatype property *dp* corresponds to a transformation *Trans* on one column *col* in another table *U*
3. A datatype property *dp* corresponds to a combination/transformation *Trans* of multiple columns col_1, col_2, …, col_n in the same table *U* other than *T*.
4. A datatype property *dp* corresponds to one column *colT* in *T* and to another column *colU* in another table *U*.

Similar cases can also be distinguished for object properties.

Category 3

When we have a concept bridge between a concept C and a table T, and when a property of another concept D corresponds to a column of T, then we are in the third category. Some of this category cases are:

1. A column *col* corresponds directly to one datatype property *dp* of another concept *D.*
2. A column *col* corresponds to a combination/transformation of multiple datatype properties.
3. A column *col* corresponds to an inherited datatype property.

However, because of the paper limits, the solutions of second and third categories are not presented.

5 Conclusion and Future works

We have proposed an ontology-evolution-based method to build ontologies from several heterogeneous information sources. In this method, an initial ontology is evolved progressively by involving an information source in each step. An evolution step includes a set of ontology change operations as well as a set of mapping bridges between the entities of the ontology being evolved and the entities of the source being involved. Two types of information source are considered, relational databases and XML documents. However, we have presented the case of relational databases only.

Involving a relational database consists of two main phases: evolving ontology concepts from database tables, and evolving ontology properties from database columns. In the second phase, several categories of correspondences are distinguished according to the established concept bridges. For instance, the currently used mappings include direct bridges, single transformation and multiple transformation bridges. We are currently working on identifying conditional correspondence cases. For example, we may say that a table *student* corresponds to a concept *Person* when the datatype property *'job'* has the value *'student'*. This will add to the used specification the notion of conditional mapping bridges. This kind of bridges applies only when its associated condition holds.

An implementation of the method presented in this paper is currently under development.

References

[1] **Barrasa J., Corcho O., Gomez-Perez A.** R2O, an Extensible and Semantically Based Database-to-Ontology Mapping Language. *Second Workshop on Semantic Web and Databases (SWDB2004).* Toronto, Canada. August 2004.

[2] **Benslimane D., Arara A., Yetongnon K., Gargouri F., Ben Abdallah H.** Two Approaches for Ontologies Building: From-Scratch and From Existing Data Sources. *The 2003 International Conference on Information Systems and Engineering (ISE 2003)*. Quebec, Canada. July 2003.

[3] **Bizer C.** The D2RQ Platform - Treating Non-RDF Databases as Virtual RDF Graphs. *http://www4.wiwiss.fu-berlin.de/bizer/d2rq/*. 2004.

[4] **Cullot N., Ghawi R., Yetongnon K.** DB2OWL: A Tool for Automatic Database-to-Ontology Mapping. *In Proceedings of the 15th Italian Symposium on Advanced Database Systems (SEBD 2007), Torre Canne di Fasano (BR), Italy.* June 2007, pp. 491-494.

[5] **de Laborda C. P., Conrad S.** Relational.OWL A Data and Schema Representation Format Based on OWL. *In Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), Newcastle, Australia*. volume 43 of CRPIT, 2005, pp. 89-96.

[6] **Fernandez-Lopez M., Gomez-Perez A., Juristo N.** METHONTOLOGY: From Ontological Art Towards Ontological Engineering. *Workshop on Ontological Engineering, Spring Symposium on Ontological Engineering, Stanford University, USA.* March 1997.

[7] **Flouris G., Plexousakis D.** Handling Ontology Change: Survey and Proposal for a Future Research Direction. *Technical Report FORTH-ICS/TR-362*. 2005.

[8] **Flouris G., Plexousakis D., Antoniou G.** A Classification of Ontology Change. *In Proceedings of SWAP 2006, the 3rd Italian Semantic Web Workshop, Pisa, Italy*. December 18-20, 2006. *CEUR Workshop Proceedings, ISSN 1613-0073, online http://ceur-ws.org/Vol-201/03.pdf.*

[9] **Gennari J., Nguyen M., Silberfein A.** DataGenie. *http://protege.cim3.net/cgi-bin/wiki.pl?DataGenie.*

[10] **Ghawi R., Cullot N.** Database-to-Ontology Mapping Generation for Semantic Interoperability. *Third International Workshop on Database Interoperability (InterDB 2007), held in conjunction with VLDB 2007*. Vienna, Austria. 2007.

[11] **Gruber T. R.** A Translation Approach to Portable Ontologies. *Knowledge Acquisition.* 1993, volume 5, no. 2, pp. 199-220.

[12] **Konstantinou N., Spanos D., Chalas M., Solidakis E., Mitrou N.** VisAVis: An Approach to an Intermediate Layer between Ontologies and Relational Database Contents. *International Workshop on Web Information Systems Modeling (WISM 2006)*. Luxembourg. 2006.

[13] **Noy N. F., Klein M.** Ontology Evolution: Not the Same as Schema Evolution. *In SMI technical report SMI-2002-0926.* 2002.

[14] **Predoiu L., Feier C., Scharffe F., de Bruijn J., Martan-Recuerda F., Manov D., Ehrig M.** State-of-the-Art Survey on Ontology Merging and Aligning V2. *Digital Enterprise Research Institute, University of Innsbruck*. 2005.

[15] **Stojanovic L., Maedche A., Motik B., Stojanovic N.** User-driven Ontology Evolution Management. *In Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW*. Madrid, Spain. 2002.

[16] **Stojanovic L., Stojanovic N., Volz R.** Migrating Data-Intensive Web Sites into the Semantic Web. *ACM Symposium on Applied Computing (SAC 2002).* Madrid, Spain. March 2002.

[17] **Uschold M., King M.** Towards A Methodology for Building Ontologies. *IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*. Montreal, Canada. 1995.

[18] **Xu J., Li W.** Using Relational Database to Build OWL Ontology from XML Data Sources. *Computational Intelligence and Security Workshops (CISW2007)*. December 2007.