A Light Weight Approach for Ontology Generation and Change Synchronization between Ontologies and Source Relational Databases

Waqas Ahmed[1,2], Muhammad Ahtisham Aslam[1,3], Jun Shen[4], Jianming Yong[5]
[1]Department of Computer Science, COMSATS Institute of Information Technology, Lahore, Pakistan
[2]Department of Computer Forensics, Punjab Forensic Science Agency, Lahore, Pakistan
[3]Faculty of Computing and Information Technology, King Abdul Aziz University, Jeddah, Saudi Arabia
[4]School of Information Systems and Technology, University of Wollongong, Wollongong, NSW, Australia
[5] School of Information Systems, Faculty of Business, University of Southern Queensland, Toowoomba, QLD, Australia

Abstract- *Ontology is specification of shared conceptualization and is building block of the semantic Web. Ontology building requires a detailed domain analysis that in turn requires financial resources, intensive domain knowledge and time. Most of industrial data is present in relational databases and a relational database schema represents a domain model. An ontology built from this schema can represent concepts and relationships that are present in domain of discourse. However, databases are not static and their schema evolves over time. Once a database schema is changed, these changes in schema should also be incorporated in ontology, generated from this database. The possible solution of regenerating a new ontology from changed database schema is not feasible because this will result in loss of manual changes of ontology. In this paper we present an approach that can be used to generate ontology from RDBs and to synchronize the generated ontology with changes occurred in the same database. We also present the prototypical implementation of the proposed approach as Protégé plug-in (i.e. DATAONTO) that can be used to generate ontology from database and to synchronize the ontology with the original database.*

Keywords: Semantic Web, Ontology Generation, Ontology Mapping, Change Synchronization, Relational Database Schema

1 Introduction

Web semantics are the representation of contents in machine interpretable and manipulatble form. These semantics and meanings are required to be expressed in a specific way to make them useable by machines. Traditional World Wide Web (WWW) presents contents for human beings whereas focus of the semantic Web is to mange and present contents for machines [1].

Ontology is a representation of shared concepts present in domain of discourse [2] and is usable in many knowledge-processing systems. These ontologies are building block of the semantic web and have enriched inference capabilities. Ontology building process begins by identifying concepts and relationship among these concepts. After identifying concepts and relationships among these identified concepts, knowledge of ontology describing languages is also mandatory to formally transform these identified concepts and relationships into a working ontology. Ontology building from scratch definitely requires domain analysis that cannot be done without expenditure of valuable financial resources and time.

Currently most of industrial and research data is present in relational databases. Usually a detailed domain analysis is performed at the time when these relational databases are built. An automated approach to having capability of transforming this relational database domain information into ontology can surely speed up the process of ontology building. Once generated, the ontology user may populate it with instances or add more domain details in it according to his or her needs.

If we consider the fact that a database's metadata contains information about concepts and relationships present in domain of discourse then we can use this information to extract ontology out of it. Once ontology is automatically built from relational database schema, the database metadata may change, which is a more common case for research related databases [3]. This change in metadata means a change in domain model. To have ontology that defines domain more accurately these changes in domain model should also be implemented into ontology that was previously generated from the same database [4]. One consideration at this point should be that the manual changes in generated ontology should not be lost. This paper presents an approach and its implementation to generate and then synchronize generated ontology with source database schema.

This paper is organized as follows: related work and shortcomings of existing work have been discussed in the Section 2. Section 3 and 4 show that how ontology can be constructed from relational database metadata and how changes can be detected and implemented in same ontology. The Section 5 describes the DATAONTO, an automated ontology generation and change synchronization plug-in for Protégé.

978-1-4577-0387-4/11/$26.00 ©2011 IEEE

2 Related work

Mukhopadhyay D. Banik and A. Mukherjee have presented an approach [16] to construct RDF ontologies from existing databases. The problem with this approach is that it uses RDF as ontology language that is less expressive as compared to OWL. This approach uses database physical schema to construct corresponding classes. This physical schema doesn't have associated semantics with it. For example, if there is a bridge table in relational database then it will also transform it into RDF class. This transformation is not a true representation of the domain knowledge.

DB2OWL [17] is a local automatic database-to-ontology mapping tool that focuses on local ontology generation from relational database. One major issue of this approach is that it uses predefined table cases and maps physical schema directly into ontology. Another drawback of this approach is that it supports object properties but these object properties are not added to classes as restrictions. This implies that only the *sufficient criteria* for class is supported but *necessary and sufficient criteria* is not supported.

Justas Trinkunas and Olegas Vasilecas presented an approach [7] to construct ontologies form relational databases using reverse engineering. Proposed approach converts a conceptual graph model into ontology but it requires an existing conceptual graph model as input. IN most times conceptual graph models of operational databases are not available, therefore, ontology generation will not be possible in that case.

The approach presented in [15] takes extended ER model as input to construct ontology out of it. As ontology is relatively new construct as compared to relational databases, the relational databases that are in operation often don't have associated models or documentations along with. That is why it is difficult to find the extended ER model and then transform it into ontology.

In our approach we address above discussed limitations to generate more consistent ontologies from relational databases. Our approach consists of two major steps: in the first step we generate conceptual graph model (shortly conceptual model) from database schema, in the second step we generate ontology from this conceptual graph model. We also present an approach that manges the synchronization between generated ontology and source database. As much as we know, till now no approach has been presented that supports the change synchronization between source database and generated ontology. Details of ontology generation from relational databases are given in the following sections.

3 Ontology Generation From Database

Ontology generation from database requires knowledge of relational database schema. This section discusses that how and what schema information is read from database then how this schema information is mapped as ontology constructs. Before transforming database schema into ontology, a mapping specification is required to perform transformations of database schema into ontology elements. This section also describes these mapping specifications.

3.1 Database Schema to Conceptual Graph Model Transformation

Database schema contains information of *tables, columns, columns datatype, primary* and *foreign keys* and other constructs such as *stored procedures, views* and *triggers*. All these details are dependent on the RDBMS used to implement the database [5]. The conceptual graph model of a database provides insight about different entities present in domain of discourse, their attributes and relationships [6]. When a conceptual graph model is transformed into a physical implementation schema, the semantics that are associated with conceptual model are lost [7].

These semantics are very important from ontology's point of view. An example of these lost semantics is transformation of bridge table that is a result of many-to-many relationship between two entities. At the time of converting conceptual graph model into a relational database schema, a many-to-many relationship results in three tables. The third table that represents this many-to-many relation consists of primary keys of two participating entities as its attributes. These primary keys of individual participating entities constitute a composite primary key in bridge table [8]. This table did not exist in the original domain model that was captured using conceptual graph model. If ontology is generated directly from this schema then it will certainly lose domain information like this, therefore, it is required to transform physical schema into corresponding conceptual representation. In the next section we describe our approach to transform physical schema into conceptual graph model.

3.2 Transformation of Physical Schema into Conceptual Graph Model

The conceptual graph is a data model that presents higher-level details of how data is stored in database [9]. In our prototypical implementation this conceptual data model is represented in the form of a directed label graph. Formally this graph G is defined as $G = \{N, E\}$, stating that graph G is a finite set of labelled nodes N and labelled edges E. Each node represents an entity of database and edge represents the relationship between entities [7]. The direction of edge represents the direction of original constraint present in database. The label of the edge is the relationship name. Each edge is written as a triplet $E = [N_1, \alpha, N_2]$ where N_1 and N_2 are nodes that are connected by edge and α is the label of edge. The Figure 1 represents a node and directed edge. Every node

represents an entity in database. A node has its *unique name* and *attributes* with their *datatype*. These attributes are actually the columns of a database table with their datatype values. An edge contains the *node names* that it is connecting and a *label* that represents the relationship between two nodes.

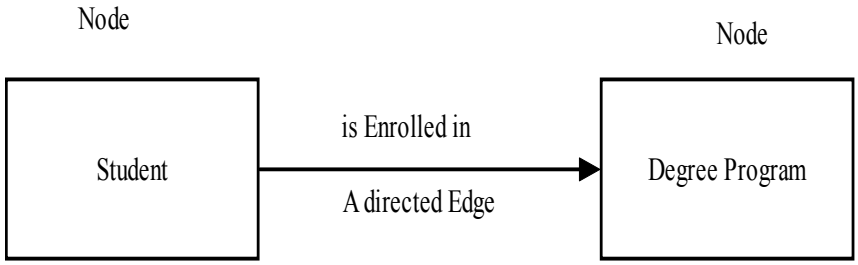

Fig. 1. A representation of graph instance. *Student* and *Degree Program* are nodes, a directed edge having a label "*is Enrolled in*" points from *Student* to *Degree Progra*

Before a model can be constructed, the information of database schema (database metadata) is required. Following information about each table is read before converting it into a model:

1. Name of the table
2. List of columns in the table along with their datatype
3. List of primary keys
4. List of foreign keys along with the names of tables in which these are primary keys

Once this information is available, a conceptual graph model can be constructed using mapping rules specified in the Table 1.

Table 1. Mapping rules for databaes metadata to conceptual graph transformation.

	Determines	**Rule**	**Mapping in Graph**
MR1	Bridge table	A table *T* is said to be a bridge table if it connects tables *T1* and *T2* in such a way that it has two distinct subset of columns *A1* and *A2*, each set belonging to the table *T1* and *T2* respectively. These attributes of table *T* are foreign keys as well as primary keys in *T*.	Two edges, one directed from table T_1 to T_2 and other from *T2* to T_1 are added in the graph. Label of graph is set to "*has a"*.
MR2	Class/subclass	If table *T* has a referential integrity constraint with table T_1 such that the primary key of table T_1 is a foreign key in table *T* and it is also a primary key in table *T* then *T* is a subclass of T_1	An edge from table T_1 to *T* is added with label *"is a"*.
MR3	Relationship	If a table *T* has a foreign key *K* that is a primary key in another table *T1* then a single instance of table T_1 is related to multiple instances of table *T*.	An edge from table *T* to T_1 is added with label *"has a"*.
MR4	Node	Each table in database is qualified as node.	Every table is added as node in the graph. Columns with their associated datatype are added as attributes. Columns that are foreign keys are not added. If a primary key is also a foreign key then it is added as node attribute.

3.3 Building Ontology from Conceptual Graph Model

Once database schema is read and transformed into conceptual graph model, this graph model can be used to construct ontology. Ontology is constructed by using *classes* and *properties*. These properties can be *object properties* and *datatype properties*. *Object properties* are used to represent relationship between two classes. *Data type* properties describe a relationship between an individual and data value [10]. We have defined some specifications (i.e. mapping specifications) to map and construct ontology from relational database conceptual graph model. Our mapping specifications define that which relational database conceptual graph model element will be transformed into which ontology construct. A higher-level description of these mapping specifications is as under.

3.3.1 Node to Ontology Class

An entity (also referred as entity type) is a basic object of conceptual database model that represents a "thing" which has independent physical or logical existence [5]. In [8] an entity type is defined as the group of objects that exists in the real world and has same properties. An ontology class defines a group of individuals that share some common characteristics [10]. It is actually a set that contains individuals those fulfil certain criteria [11].
It is clear from the definitions of entity type and ontology class that both contain individuals sharing some common properties or characteristics. Since a node in conceptual graph model represents an entity in domain of discourse, therefore, it may become a class in ontology.

3.3.2 Node Attributes to Ontology Datatype Properties

Sets of properties, which completely define entity type, are called its attributes [5]. Attributes have values that are main part of data stored in database [8]. OWL datatype properties represent a relationship of an individual with a data value. Keeping this similarity in mind, we can replace node attributes with OWL datatype properties in ontology.

3.3.3 Node relationship to Object Property

Whenever an attribute of an entity type refers to another attribute of some other entity then there exists a relationship between these two entities. If an entity type can be mapped as ontology class then we also need to map relationships between entity types. These relationships can be mapped by using object properties of ontology. In OWL the object properties are used to represent some relationships between classes. So relationships that exist in conceptual database model are transformed into OWL *object properties*. An edge with label *"has a*" represents a one-to-many relationship between nodes it is connecting. This relationship can be mapped by adding an *object property* in OWL ontology.

3.3.4 Class/Subclass relationship between Nodes to Class Hierarchy

Entities can be specialized in the form of a specialization hierarchy. This hierarchy is very important if these entities are to be transformed into ontology. Each sub entity can be a subclass of its corresponding specialized entity class. An edge with label *"is a"* is mapped as class/subclass relation in ontology.

3.4 Conceptual Graph Model to Ontology Mapping Rules

The Table 2 lists Ontology Mapping Rules (OMR) used for generating ontology from conceptual graph model.

Table 2. Conceptual graph to ontlogy mapping rules.

	Rule
OMR1	Each node in graph is mapped as a class in ontology. To define *necessary and sufficient* criteria for being a member of class, class will be an intersection of all restrictions on its *datatype* and *object properties*.
OMR2	All *node attributes* are mapped as *datatype properties* in ontology. The range of *datatype property* is set to XML schema datatype (XSD) equivalent of attribute's datatype in relational database. An attribute with similar name may belong to more than one node therefore node name is appended before the attribute name.
OMR3	If an *edge* with label *"has a"* connects node N_1 with node N_2 then an object property *OP* named as $hasN_2$ will be created in the ontology. Domain and range of property *OP* will be set to N_1 and N_2 respectively. The property will also be made functional to make it sure that it connects only one instance at a time.
OMR4	If an edge with label *"is a"* connects two nodes N_1 and N_2 then in ontology, N_1 will be mapped as subclass of N_2.

4 Change Detection and Synchronization between Database and Ontology

Public and research databases are not static and their schema keeps changing [3]. New tables are added, existing tables are deleted and new columns are added or deleted from existing tables. If ontology is generated from database schema considering the fact that this schema contains domain information then this change in schema will directly imply a change in domain model. Once domain model is changed, the ontology representing domain of discourse should also be changed.

Once ontology is generated from a database schema using the technique discussed in the Section 3, this database schema may go through changes. A database schema may go through following changes:

1. Addition of new table(s)
2. Deletion of table(s)
3. Addition of column(s) in table(s)
4. Deletion of column(s) in table(s)
5. Change of datatype of column(s)
6. Addition of relationship between tables
7. Deletion of relationship between tables

The process of change detection begins by constructing a conceptual graph model from changed database schema. The construction of conceptual graph is same as discussed in the Section 3.2. Once we have the new conceptual graph model, it is compared with previous version of conceptual graph model that was used to generate ontology. The difference between two models represents the changes in schema. These changes are maintained in a change set. This change set is then used to map these changes into ontology. Table 3 lists Change Detection Rules (CDR). Keywords G_N and G_O symbolize newly generated and old graph models.

4.1 Change Set

Rules mentioned in the Table 3 can be used to detect changes in database schema. Once changes are detected they are stored in the form of change set. This change set is then used to implement these changes into ontology. Change set consists of following items:

1. A conceptual graph model
2. Names of nodes to be deleted
3. Names of edges to be deleted
4. Names of attributes to be deleted along with their node names
5. Name of attributes whose datatype is to be changed along with class name and new datatype value

One thing to be noted here is that the above-mentioned list does not contain the names of newly added nodes and edges but it contains a graph model. This graph model is actually a subset of changes that represents newly added nodes and edges. The Figure 2 shows the process of construction of change set.

4.2 Change Mapping Rules

Once, changes in database schema are identified they can be implemented in the ontology that has already been generated from database. Before a change is made into ontology it is necessary to know which conceptual graph model element was mapped to what ontology construct. For this purpose we may use the rules specified in the Table1. The Table 4 presents Change Mapping Rules (CMR) to implement changes into ontology. After changes are implemented in ontology, newly read graph model G_N is saved for future change detection and implementation.

Table 3. Change detection rules.

	Detects	Rule
CDR1	Deleted table	If a node N does not exists in G_N but is present in G_O then it is deleted from database
CDR2	Added table	If a node N exists in G_N but is not present in G_O then this is a newly added table in database
CDR3	Added column	If a node attribute A is part of node N in G_N but same node N does not have any attribute named as A in G_O then this is a newly added attribute
CDR4	Deleted column	If a node attribute A is part of node N in G_O but same node N does not have any attribute named as A in G_N then this attribute is deleted from database
CDR5	Change of datatype	If a Node N in G_N has an attribute named as A with datatype D and G_O also has the node named as N and that N has an attribute named as A but datatype of A is not D then its datatype is changed
CDR6	Addition of relationship	If an edge E having label L connects two nodes N_1 and N_2 in G_N but same edge does not exist in G_O then E represents a newly added relation between node N_1 and N_2
CDR7	Deletion of relationship	If an edge E having label L connects two nodes N_1 and N_2 in G_O but same edge does not exist in G_N then E represents a deleted relation between node N_1 and N_2

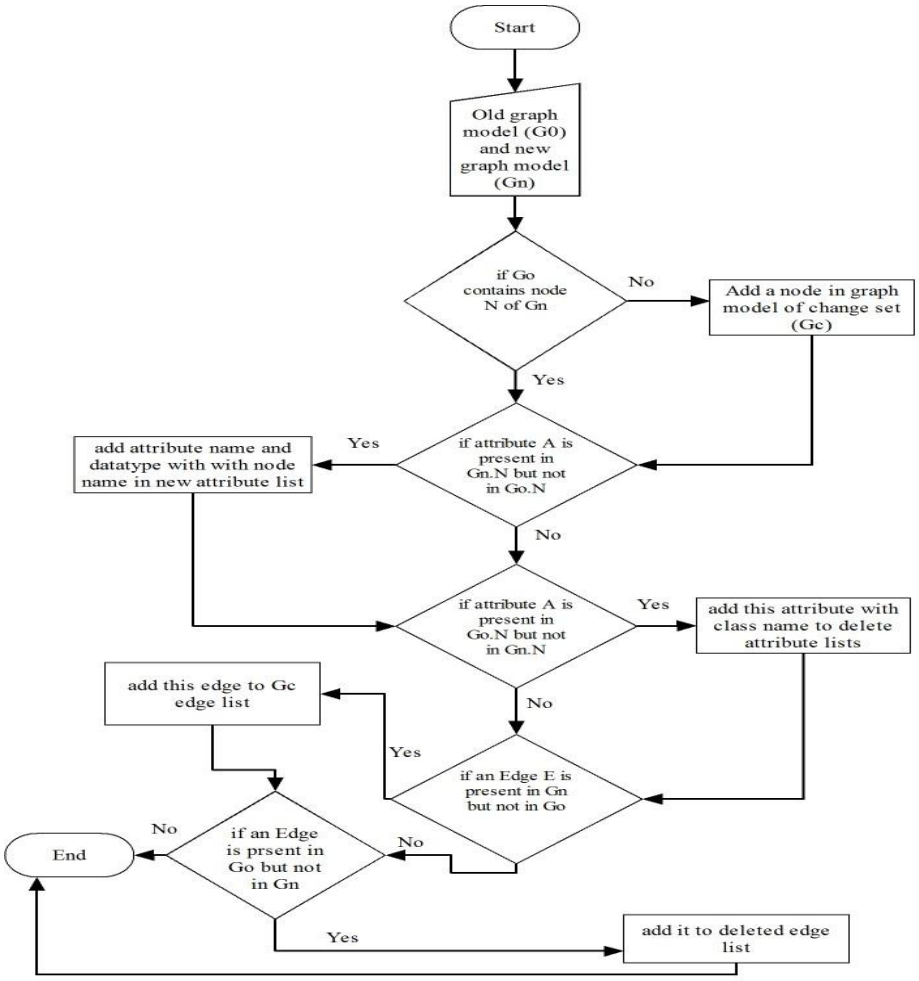

Fig. 2. Process of change set construction.

5 DATAONTO: A Data-to-OWL Conversion and Synchronization Plug-in for Protégé

We have developed a plug-in (named as DATAONTO[1]) for famous and widely used ontology editor, protégé [12]. This plug-in reads database schema as input and transforms it into OWL-DL ontology using the presented approach. Once ontology is generated, DATAONTO also looks for database schema changes in source database and if changes are found, they are implemented in existing ontology. JAVA JDBC API [13] is used for reading database schema and Jena 2.1 [14] is utilized for writing or reading ontology files.

DATAONTO is implemented in Java, a platform independent language, widely used by research community and due to availability of online support. For the purpose of manipulating OWL-DL ontologies, Jena 2.1 API is used. Jena is an open source, well maintained Java framework that can be used to build semantic web applications. It supports RDF, RDFS, OWL and SPARQL along with a rule based inference engine. Some important features of DATAONTO are as follow:

1. Automatic schema Reading
2. Acquisition of implicit semantics from relational database
3. Automatic change detection and mapping of these changes into existing ontology

6 Conclusion and Future Work

In this paper we presented a generic approach and its prototypical implementation for ontology generation and change synchronization from relational databases. Our work has two fold contributions: 1) the ontology generated by using our approach is more consistent because it supports classes and properties, and defines relations

[1] https://sourceforge.net/projects/dataonto/

Table 4. Change mapping rules.

	Rule
CMR1	The conceptual graph present in change set will be mapped similar to the graph transformation specified in the Section 3.4. This will add new classes and relationships between these new classes into ontology
CMR2	For each node present in the *change set delete node list*, corresponding class in ontology will be deleted. All datatype properties of this class will also be deleted
CMR3	For each edge *E* labelled as *"has a"* connecting Nodes N_1 and N_2, in *change set new edge list*, an object property with domain N_1 and range N_2 will be added in N_1. If edge label is *"is a"* then class corresponding node N_1 will be made subclass of class represented by N_2
CMR4	For each edge *E* labelled as *"has a"* connecting Nodes N_1 and N_2, in the *change set delete edge list*, the object property corresponding this edge in ontology will be deleted. If edge label is *"is a"* then the class/subclass relation between N_2 and N_1 will be removed
CMR5	For each element in the *change set add attribute list*, a new datatype property *P* will be added to the class corresponding to node to which the attribute belongs. The range of datatype property will be set as the XSD equivalent of SQL data type
CMR6	For each element in the *change set change attribute data type list*, the range of data type property *P* that corresponds to that attribute in ontology, will be changed to new XSD equivalent of SQL data type
CMR7	For each element in the *change set add attribute list*, data type property *P* that represents attribute in ontology will be deleted

between classes by using properties. These relations result in more powerful reasoning capabilities by semantic Web agents. 2) Our work supports the change synchronization between generated ontology and source database. As far as we know till now no approach has been presented that supports the change synchronization between source database and ontology.

In future, we plan to extend this approach and identify mapping rules for XML and object oriented database schema so that ontology can also be generated from them. At present we are working on more stable release of DATAONTO as a plug-in for widely used ontology editor protégé. Our aim is to present DATAONTO as a tool that can generate accurate and closer to domain ontology skeleton from any data source.

References

[1] Berners-Lee T., Hendler J., and Lassila O.: The Semantic Web. May 2001, Scientific American.

[2] Studer R., Benjamins V. R., Fensel D.: Knowledge Engineering: Principles and Methods. In: IEEE Transactions on Data and Knowledge Engineering, pp. 161-197. (1998)

[3] Kupfer, S. Eckstein, K. Neumann, B. Mathiak : A Coevolution Approach for Database Schemas and Related Ontologies. In: 19th IEEE International Symposium on Computer-Based Medical Systems, CBMS, pp. 605-610. (2006)

[4] Natalya F. Noy, Michel Klein.: Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems. 428-440. (2004)

[5] Ramez Elmasri, S. B. Navathe: Fundamentals of Database Systems. International Edition : Addison and Wesely, (2000)

[6] Felden C., Kilimann D.:Deployment of Ontologies in Business Intelligence Systems. In: Eighth International Conference on Enterprise Information Systems. Paphos Cyprus. (2006)

[7] Trinkunas J., Vasilecas O.:Building Ontologies from Relational Databases Using Reverse Engineering Methods. In: International Conference on Computer Systems and Technologies.Bulgaria. June 14 – 15.(2007)

[8] Thomas M. Connolly, Carolyn E. Begg.: Database Systems: A Practical Approach to Design, Implimentation and Management. Third Edition. ABC. (2000).

[9] Chen P. P,: The entity-relationship model: Towards a unified view of data. ACM Transactions on Database Systems. 1(1)., pp. 471-522. (1976)

[10] OWL Specifications, http://www.w3.org/TR/2004/REC-owl-features-20040210/

[11] Matthew Horridge, Simon Jupp, Georgina Moulton, Alan Rector, Robert Stevens, Chris Wroe.: A Practical Guide To Building OWL Ontologies Using Protege 4. The University Of Manchester, (2007).

[12] protégé ,protégé, http://protege.stanford.edu/

[13] JDBC, http://java.sun.com/javase/technologies/database/

[14] Jena, http://jena.sourceforge.net

[15] Sujatha R Upadhyaya, P Sreenivasa Kumar.: ERONTO: A Tool for Extracting Ontologies from Extended ER Diagrams. In: ACM Symposium on Applied Computing. (2005)

[16] Mukhopadhyay D. Banik A., Mukherjee,: A Technique for Automatic Construction of Ontology from Existing Database to Facilitate Semantic Web. In: 10th International Conference on Information Technology (ICIT 2007), pp. 246-251. (2007).

[17] Cullot N., Ghawi R., Yétongnon K.: DB2OWL: A Tool for Automatic Database-to-Ontology. In: Proceedings of the 15th Italian Symposium on Advanced Database Systems. (2007).