A Building Method of Virtual Knowledge Base Based on Ontology Mapping

Huayu Li[1] and Xiaoming Zhang[2]

[1] College of Computer & Communication Engineering, China University of Petroleum, Qingdao 266555, China
[2] School of Information Science & Engineering Hebei University of Science and Technology, Shijiazhuang 050018, China
lhyzj@upc.edu.cn, zxm1975@gmail.com

Abstract. For distributed relational data sources in oilfield domain, we propose a semantic integration method of building virtual knowledge base using ontology mapping technology. In this method, two kinds of mappings are created and two ontologies designed specially to record these mappings, by which semantic query can be rewritten into SQL statements to be executed on data sources, and results will be returned as knowledge instances. Moreover, a knowledge exchange center is provided to cache some results to improve query efficiency. Since query results directly come from data sources rather than by instances transformation, this query-driven integration system can provide a unified, virtual knowledge base which can be served to achieve decision support for oilfield production.

Keywords: semantic integration, virtual knowledge base, ontology mapping.

1 Introduction

At present, research of semantic integration mostly focuses on specific domains, and ontology mapping is used as main technology which can describe relationships between global and local concepts clearly, and they will be served as important reference for instances transformation or query rewriting. Instances transformation method includes two procedures: first, extraction rules are defined to map data source model into a local ontology and mappings are recorded; second, records of data sources are transformed into ontology instances to build Knowledge Base to support semantic query. This method can achieve complex query requirement and is applicable to those data sources with low update frequency, SOAM [1] and Stojanovic [2] belong to this kind of method. Another is query rewriting method and it can obtain results directly from data sources by rewriting algorithm, which can convert a semantic query into a group of SQL statements to be executed at data sources. This query-driven method is suitable for those data sources with such features as real-time update and moderate scale. For example, Dart Grid [3] and INDUS [4] all make use of this method.

For distributed relational data sources of Well-Engineering, we propose a semantic integration solution of building virtual KB by ontology mapping technology. This solution first establishes global and local ontologies to describe domain-concept models.

Second, a mapping ontology is created to record two kinds of mappings: one is between global and local ontologies; the other is between local ontology and data sources model. Finally, a semantic query rewriting process is executed to get results from data sources, which will be converted into ontology instances to populate KEC (Knowledge Exchange Center). KEC is used to cache part of instances to reduce consumption caused by data conversion and loading.

Reminder of this paper is organized as follows: Section 2 illustrates implementation framework; Section 3 describes building method of global and local ontologies; Section 4 describes realization mechanism and expression method of ontology mapping; Section 5 introduces execution process of semantic query and rewriting algorithm; Section 6 gives conclusions.

2 Implementation Framework

According to proposed solutions, we implement WeVKB (Well-Engineering Virtual Knowledge Base) system. As shown in Fig. 1, WeVKB involves three layers: User Layer, Virtual-KB Layer and Wrapper Layer.

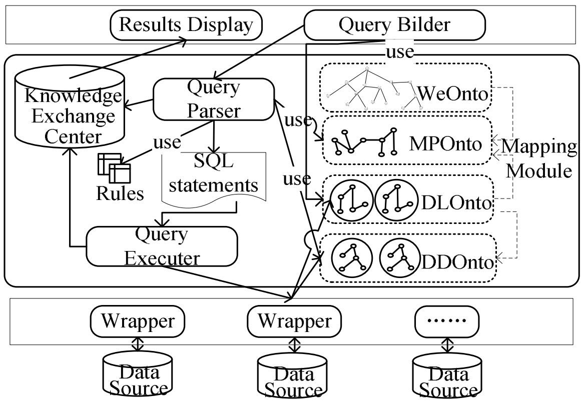

Fig. 1. Framework of WeVKB is consisted of three layers. By their respective functions, this system can provide an effective semantic integration platform.

(1) User Layer: it provides users query interfaces and contains two components: Query-Builder and Query-Display. The former takes charge of creating SPARQL query Q_S according to requirements set by users. The function of Query-Display is to reorganize the results sent by KEC and display them in certain format.

(2) Virtual-KB Layer: it is the core of WeVKB and completes two functions: receive Q_S from Query-Builder and rewrite it into a group of SQL statements to be executed by Wrapper-Layer; receive query results and returned them to KEC.

Four key components are included in Ontology-Mapping Module: WeOnto (Global Well-Engineering Ontology), DLOnto (Data Source Local Ontology), DDOnto (Data Source Description Ontology) and MPOnto (Mapping Ontology). WeOnto and DLOnto describe global and local domain concepts model, which are displayed by

Query-Builder as hierarchical structures for user to set query requirements. Two kinds of mappings need to be created for query rewriting: one is between WeOnto and DLOnto; the other is between DLOnto and data source model. MPOnto and DDOnto are defined to describe them respectively in standard format. KEC saves the latest query results. When a semantic query is presented, it is first delivered to KEC, if no results or insufficient results returned, the whole query process will be executed, and the returned results will firstly populate KEC before submitted to Query-Display.

(3) Wrapper Layer: this layer includes all distributed data sources and corresponding wrappers. Data sources involve relational database such as SQL Server, Oracle and Access. Wrappers are responsible for executing the dispatched SQL statements and submit the results to Query-Executer.

3 Ontology Construction

By referring to methods proposed in [5] and [6], we construct WeOnto under guidance of domain expert. According to application requirements, concepts range of WeOnto is determined in accordance with data model of Exploration-Production database which is widely used in most oilfields. At present, we define 52 classes and 246 properties relating to production, operation, measurement and cost. Part structure of WeOnto is shown as Fig.2.

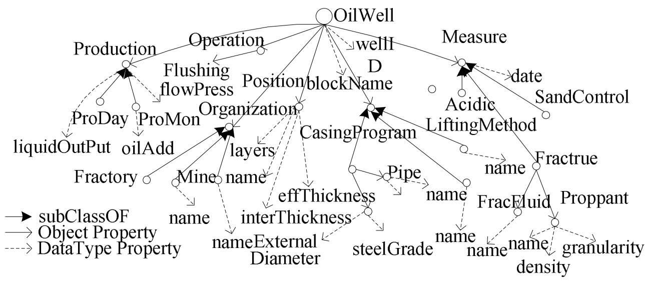

Fig. 2. *OilWell* is the root class of WeOnto. For each class, its subclass is denoted as a start node of a solid arrow line; its object property and data type property is denoted respective as an end node of a solid arrow line and a dotted arrow line.

With referential integrity constraints extracted from database schema, four procedures are defined to create DLOnto from data source by mapping table and attributes to ontology classes and properties: the first is to classify all user tables into three table Sets; the second is the definition of three extraction rules for each table Set; the third is to start a program to create initial local ontology; the fourth is the perfection process by ontology edit tool to construct the final local ontology.

4 Ontology Mapping

Two kinds of mappings exist in WeVKB. One is between DLOnto and data source model, named $M_{L\text{-}DS}$; the other is between WeOnto and DLOnto, named $M_{G\text{-}L}$.

We adopt a similar method with Relational.OWL [7] to express $M_{L\text{-}DS}$ by creating a data source discription ontology named DDOnto. Some classes and properties are defined in DDOnto to describe data source model and mappings of $M_{L\text{-}DS}$. Four objects of data source model including *database*, *table*, *column* and *primary key*, are expressed by *ddonto*:*Database*, *ddonto*:*Table, ddonto*:*Column and ddonto*:*PrimaryKey*; relationships among these classes are denoted by three object properties: *ddonto*:*hasTable, ddonto*:*hasColumn,* and *ddonto*:*hasPK*. Mappings of inter-classes and inter-properties are key information needed to be recorded in DDOnto, and they are described by two object properties: *ddonto*:*relatedClass* and *ddonto*:*relatedProperty*.

$M_{G\text{-}L}$ is expressed by MPOnto with which two kinds of mappings are expressed:

(1) Equivalence and sub-class mapping among classes: *EquivalentClass* and *subClassOf* are used as descriptors to denote this mapping.

(2) Equivalence and sub-property mapping among properties: two kinds of express forms need to be considered. For simple case, *subPropertyOf* and *equivalentProperty* can be directly used. While in complex case, a series of logical rules need to be used altogether to describe every semantic expression in RDF triples format. For instances, *oilIncrMon*, a property of $DLOnto_{DS1}$, can be defined equivalent to the property *oilAdd* in WeOnto by following rules:

Following rules gives a mapping example between properties

```
(?x rdf:type dlonto1:WellInfo)∧(?x dlonto1:meaEffect ?a)∧
(?a dlonto1:effIndex ?b)∧(?b dlonto1:oilIncrMon ?y)→
(?x rdf:type weonto:OilWell)∧(?x weonto:production ?c)∧
(?d rdfs:subClassOf weonto:?c)∧(?d rdf:type weon-
to:ProMon)∧(?d weonto:oilAdd ?y)
```

With above expressions method and *owl:imports* statement which is used to import WeOnto and DLOnto, MPOnto can distinctly describe mappings of $M_{G\text{-}L}$.

5 Implementation of Semantic Query

As a query-driven semantic integration system, WeVKB takes advantage of ontology mappings and query rewriting algorithm to achieve query process, and this process can be divided in four procedures:

(1) Q_S is first delivered to KEC to query cached instances. If returned results exists *null* values, proceeding to execute next procedure; otherwise, Query-Display presents query results to users in certain format.

(2) According to query scope of Q_S, Query-Parser reads mappings from MPOnto or DDOnto and executes query rewriting algorithm. Consequently, Q_S is rewritten into a group of SQL statements: Q_{DS1}, Q_{DS2}, … Q_{DSn}.

(3) These SQL statements are dispatched to corresponding wrappers to execute, and their returned results are denoted as R_{DS1}, R_{DS2}, … R_{Dn}.

(4) Query-Executer reorganizes all results and transforms them into ontology instances which will be submitted to KEC and Query-Display.

By constructing semantic query tree, we realize a rewriting algorithm and it involves five realization steps:

(1) Q_S is read into Query-Parser and analyzed to extract all classes, properties, operators and constants included in *Select*, *Where*, and *Filter* sub-statements. Meanwhile, relationships contained in every RDF triple of Q_S are also recorded.

(2) Basic information recorded in Step 1 are packaged into three Java objects named *NodeC*, *NodeP* and *NodeR*, with which a query tree named Q_S-Tree is build and its leaf nodes are *NodeP* or *NodeR*, branch nodes are *NodeC*.

(3) A traversal process is launched to access every node of Q_S-Tree, by mappings information, each node is mapped to one or several table object, column object or condition object named *NTab*, *NCol*, and *NCon*. According to their pre-mapped relationships in Q_S-Tree, these objects are then interconnected each other to create a SQL query graph called SQL-Graph.

(4) A search process is executed to find all sub-graphs of SQL-Graph which meets two conditions: each sub-graph is a tree including all *NCol* objects; each node in sub-graph has valid relationship conforming to the logical model of any data source. Returned sub-graphs are marked as SQL-Tree$_1$, SQL-Tree$_2$, …, SQL-Tree$_n$.

(5) For these SQL-Tree, a bottom-to-top traversal process is executed to generate a group of SQL statements named Q_{DS1}, Q_{DS2}, …, Q_{DSn}, each of which will be dispatched to its corresponding data source.

The following statement is an example of SPARQL query Q_S

```
Select ?x ?y Where { ?a rdf:type OilWell ?a weonto:
wellID ?x ?a weonto:production ?b ?b rdf:type Production
?b weonto:proMon ?y ?a weonto:blockName "Guan15-6" ?a
weonto:measure ?c ?c rdf:type Fracture ?c weonto:name
"yl" Filter (?y > 10)}
```

By this algorithm, Q_S-Tree, SQL-Graph and two SQL-Trees are build as Fig. 3. Two SQL statements are generated, and they are expressed as following statements.

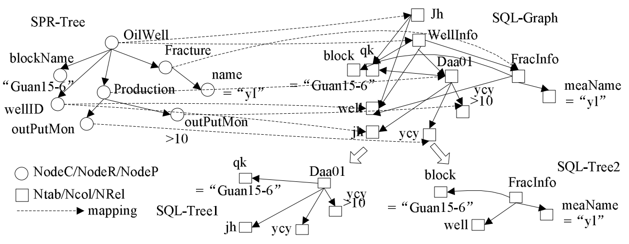

Fig. 3. The top part illustrates a process of step 3, which converts *SPR-Tree* into a *SQL-Graph*. Based on *SQL-Graph* and step 4, two *SQL-Trees* are generated and displayed at the under part.

The following two statements are the final SQL queries generated from Q_S

```
For DS1: Select fracInfo.well
         from fracInfo where fracInfo.block = "Guan15-6"
         and fracInfo.meaname = 'yl'
For DS2: Select daa01.jh, daa01.ycy from daa01
         where daa01.qk= "Guan15-6" and daa01.ycy>10.
```

6 Conclusions

In order to test proposed methods, we implement WeVKB system using Java-Struct development framework. By practical application, it can provide effective semantic integration services for oilfield production and decision makings. The future work of WeVKB is to realize automation discovery of mappings and increase efficiency of semantic query rewriting.

Acknowledgments. This work is supported by the "Fundamental Research Funds for the Central Universities": No. 27R1007007A.

References

1. Li, M., Du, X.-Y., Wang, S.: A semi-automatic ontology acquisition method for the semantic web. In: Fan, W., Wu, Z., Yang, J. (eds.) WAIM 2005. LNCS, vol. 3739, pp. 209–220. Springer, Heidelberg (2005)
2. Stojanovic, L., Stojanovic, N., Volz, R.: Migrating data-intensive web sites into the Semantic Web. In: Proceedings of the 2002 ACM Symposium on Applied Computing, pp. 1100–1107. ACM (2002)
3. Chen, H., Wang, Y., Wang, H.: Towards a Semantic Web of Relational Databases: a Practical Semantic Toolkit and an In-Use Case from Traditional Chinese Medicine. In: Proceedings of the 5th International Semantic Web Conference, pp. 750–763. Springer, Heidelberg (2006)
4. Caragea, D., Pathak, J., Bao, J.: Information Integration and Knowledge Acquisition from Semantically Heterogeneous Biological Data Sources. In: Proceedings of the Data Integration in Life Sciences, pp. 175–190. Springer, Heidelberg (2005)
5. Noy, N.F., Mcguinness, D.L.: Stanford Medical Informatics Technical Report SMI-2001-0880. Stanford: Stanford Knowledge Systems Laboratory (2001)
6. Villanueva-Rosales, N., Dumontier, M.Y.: OWL: An ontology-driven knowledge base for yeast biologists. Journal of Biomedical Informatics 41(5), 779–789 (2008)
7. De Laborda, C.P., Conrad, S.: Relational.OWL: a data and schema representation format based on OWL. In: Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modeling, pp. 89–96. Australian Computer Society (2005)