A Model Driven Approach to Constructing Knowledge Graph from Relational Database

A Model Driven Approach to Constructing Knowledge Graph from Relational Database

Shanxin Sun[a], Fanchao Meng[b] and Dianhui Chu

School of Computer Science and Technology, Harbin Institute of Technology at Weihai, Weihai, 264200, China

E-mail: [a]17862703598@163.com; [b]fcmeng@hit.edu.cn

Abstract. Relational database is one of the main data sources in the construction of vertical domain knowledge graph. How to establish the mapping relationship between the concepts in relational database and the corresponding concepts in knowledge graph is the key to improve the efficiency of the construction of vertical domain knowledge graph. To solve this problem, this paper proposes a model driven approach to automatically constructing knowledge graph from relational database based on the Model Driven Architecture (MDA). First of all, a model driven framework is proposed to describe the relational database and knowledge graph using standard XML Schema. Then, a model driven approach is proposed, which extracts database models and data to XML, defines the conversion functions, transforms models and data from database to knowledge graph, and builds knowledge graph according to the expression forms. Finally, using a case to validate the approach. This paper uses the idea of model driven to realize the high-quality transformation from relational database to knowledge graph.

1. Introduction

Knowledge Graph is a technical approach that uses graph model to describe the relationship between knowledge and model the world. It consists of nodes and edges[1]. Nodes represent all things in the world and edges represent the relationship between all things. In recent years, knowledge graph has played an increasingly important role in semantic search, intelligent Q&A, language understanding and other fields. The vertical domain knowledge graph is oriented to a specific domain, which is usually constructed by top-down construction. Domain experts use the ontology editing tools such as protégé and WebOnto to build domain knowledge and then fill in specific data by understanding the domain. On the one hand, it will waste a lot of time and energy by manual operation. On the other hand, it is difficult to ensure the completeness and correctness of domain expert knowledge. How to acquire domain knowledge from data automatically and with high-quality becomes a difficult problem.

The source data of vertical domain knowledge graph mostly comes from unstructured data, semi-structured data and structured data. There is a lot of domain independent information in unstructured data, which interferes with domain knowledge extraction. If you want to build a complete domain knowledge, you need a lot of text support, and the construction efficiency is low[2]. Semi-structured data is mostly encyclopedia data or web data, which contains a certain structural information. It is helpful for knowledge extraction. Structured data has strict structural information, and is mostly stored in relational databases. Generally, each domain or enterprise has its own relational database. Domain knowledge is relatively complete. Extracting domain knowledge from relational database can

effectively improve the correctness of building vertical domain knowledge graph. Currently, relevant standards and tools support the transformation of relational database data into RDF Data, OWL ontology, etc. The RDB2RDF working group of W3C has published two mapping languages: DM[3] (Direct Mapping) and R2RML[4]. Many OBDA (Ontology Based Database Access) systems support direct access to relational databases in the form of knowledge graph, such as D2RQ, Mastro, etc. In this paper, an approach of constructing vertical domain knowledge graph from relational database is proposed, and knowledge is acquired from enterprise relational database.

2. Related work

W3C's direct mapping specification defines a simple transformation from relational database to RDF graph data, and provides a basis for defining and comparing more complex transformations[5]. The R2RML mapping language is a language for representing custom mapping from relational databases to RDF datasets[6]. The difference between the two is that direct mapping can only use the names of database schema elements, while R2RML can flexibly use custom target views. In addition to simply extracting data from relational databases, some scholars have also focused on extracting semantic relationships from relational databases. The relational database based on educational information in reference[7] used D2RQ mapping tools to extract semantic metadata from the data. Reference[8] proposed an approach for discovering and transforming semantic relationships between database entities based on random forests to transform relational data into RDF.

OWL, as another language used to describe the knowledge graph, has also been studied at home and abroad. Reference[9] formed ontology and corresponding instance data by extracting relation mode and data instance, and constructs ontology with OWL language. Reference[2] proposed an automatic construction approach of OWL ontology, which realized the automatic construction from relational database to OWL ontology.

In addition to the transformation of relational database, reference[10] introduced an approach that can transform ER model into OWL ontology, and reference[11] introduced a transformation approach from XML Schema to OWL ontology model, and reference[12] introduced the transformation rules from XML schema to RDF.

3. The framework of knowledge graph construction based on model driven

OMG has published model driven architecture (MDA) to solve the non-standard problems in the process of software development. MDA is an approach to constructing abstract model of business logic and generate complete application program automatically. MDA is divided into four layers, namely meta-meta-model layer, meta-model layer, model layer and instance layer. The approach of knowledge graph construction based on model driven is suitable for the vertical domain where the data source is relational database. The characteristic of data is that it has strict structural information and can well realize the extraction and transformation of model and data. Knowledge graph can be regarded as a kind of semi-structured data. We can shield the differences of different types of relational database and knowledge graph by using the unified description language, and then use the established rules to realize the transformation. The transformation from data source to knowledge graph requires detailed description of data source and knowledge graph based on model driven and formulation of transformation rules. The transformation is shown in figure 1.

The data source includes all kinds of relational databases, such as MySQL, Oracle. These databases have their own definition languages and corresponding meta-model element. Users use the meta-model elements to build different scenarios to solve the industry needs. The instance layer is the specific data in a specific scenario, which is filled by the user according to the model design.

The knowledge graph is logically divided into two layers: schema layer and data layer, which corresponds to the MDA's model layer and the instance layer. In the data layer, knowledge is stored in units of facts, and its basic forms are 'entity-relation-entity' and 'entity-key-value' triples. Entities are interconnected through relationships to form a networked knowledge structure. The schema layer highly refines the knowledge of the data layer. Knowledge graph also has many storage forms. These

storage forms have their own definition language, and the defined meta-model elements are expanded around triples. The knowledge graph mentioned in this paper also refers to different types of knowledge graph storage forms.

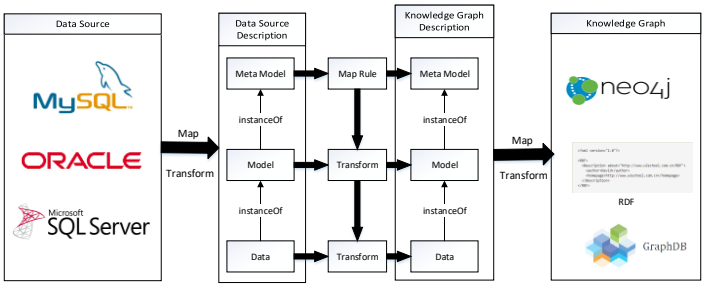


Figure 1. The transformation from data source to knowledge graph

4. Modelling method based on XML

4.1. Meta-meta-model layer description

In this paper, we use XML Schema as the description language of meta-meta-model layer. We use XML Schema to describe the elements and relationships needed in meta-model layer. We use the elements and relationships of meta-model layer to describe the model layer and instance layer to form XML file. To better define the meta-model layer, we define the XML Schema abstract model as a five-tuple[13] shown in equation (1).

$$(E,A,T,P,C) \tag{1}$$

In the equation (1):

- E is a collection which contains all element names in the XML Schema.
- A is a collection which contains all attribute names in the XML Schema. We use @ as the attribute prefix.
- T is a collection of data types which specifies the types of all data in the XML Schema. T is divided into simple data type (ST), composite data type (CT) and empty type (ET). We use '::=' to mean 'defined as'. ST is defined as $ST::=int|string$, which is the built-in data type in the XML Schema. In the subsequent description, the ST will not be described any more. CT consists of simple data types, defined as $CT::=order[e_1:t_1,...,e_n:t_n]$, where *order* represents *Sequence*, *Choice*, and *All*. If e appears more than once, it is marked with '*'. ET indicates that the type does not contain any simple data type.
- Type function P, $P(A)=T$ specifies the data type of attributes in A; $P(E)=T$ specifies the data type of elements in E.
- Attribute contains function C, $C(E)=[a_1,...,a_n]$. Every element in E contains a set of attributes, and the attributes are not repeated.

4.2. Database meta-model layer description

We describe the elements and relationships used in various relational databases as follows:

- Data source is relational database, and root element is defined as element 'database'.
- For 'database', it uses the attributes 'source', 'version', 'dbName' and 'dbType' to identify the source, version, name and type. If 'database' represents the model layer, the 'dbType' value is 'Model', and if 'database' represents the instance layer, the 'dbType' is 'Data'.
- Element 'database' contains multiple sub-elements 'table'.
- The 'table' includes the entity table that represents specific things and the relationship table that represents the relationship between things. For 'table', we use the attribute 'tableName' to identify its name. Element 'table' contains multiple sub-elements 'row' and 'foreignKey'.

- We use element ‘row’ to represent the definition in the model layer or the data in the instance layer. Element ‘row’ contains one or more sub-elements ‘column’.
- Element ‘column’ represents a field.We use the attributes ‘colName’ and ‘colType’ to identify the name and type. If in the model layer, the ‘column’ can be filled with default values.
- Element ‘foreignKey’ reflects the relationship between various tables, and it contains the sub-elements ‘fkName’, ‘fkField’, ‘fkCitedTable’ and ‘fkCitedField’.
- Element ‘fkName’ represents the name of the relationship. Element ‘fkField’ represents the field specified by the relationship in the table. Element ‘fkCitedTable’ represents the table associated with the relationship. Element ‘fkCitedField’ represents the field in the table associated with the relationship. We can use these four elements to determine the relationship between the specific data in the two tables.

According to the above description, we use five-tuple to describe the database.

- Element definition:

$$E=[database,table,row,column,foreignKey,fkName,fkField,fkCitedTable,fkCitedField]$$

- Attribute definition:

$$A=[@dbType,@source,@version,@dbName,@tableName,@colName,@colType]$$

- Data type definition:

$$t_{COLUMN}::=ET,\ t_{ROW}::=sequence[(column)*:t_{COLUMN}]$$
$$t_{FOREIGNKEY}::=sequence[fkName:string,fkField:string,fkCitedTable:string,fkCitedField:string]$$
$$t_{TABLE}::=sequence[(row)*:t_{ROW},(foreignKey)*:t_{FOREIGNKEY}],\ t_{DATABASE}::=sequence[(table)*:t_{TABLE}]$$

- Type function definition:

$$P(column)=t_{COLUMN},\ P(foreignKey)=t_{FOREINGNKEY},\ P(row)=t_{ROW},\ P(table)=t_{TABLE}$$
$$P(database)=t_{DATABASE},\ P(fkName)=string,\ P(fkField)=string,\ P(fkCitedTable)=string$$
$$P(fkCitedField)=string,\ P(@dbType)=string,\ P(@source)=string,\ P(@version)=string$$
$$P(@dbName)=string,\ P(@tableName)=string,\ P(@colName)=string,\ P(@colType)=string$$

- Attribute contains function definition:

$$C(column)=[@colName,@colType],\ C(table)=[@tableName]$$
$$C(database)=[@dbType,@source,@version,@dbName]$$

Use Altova XMLSpy to draw elements and relationships, as shown in figure 2.

4.3. Knowledge Graph meta-model layer description

Because different knowledge graphs use different representations, for example, Neo4J uses the concepts of nodes, relationships and attributes, while RDF uses the concepts of resources, attributes and relationships. Through the analysis, we find their basic concepts are the triple, so we use the triple when we define the elements of knowledge graph’s meta-model. We describe them as follows:

- Root element is ‘knowledgeGraph’.
- We use the attribute ‘kgType’ to distinguish whether element ‘knowledgeGraph’ is a schema layer or a data layer. There are two optional values of ‘kgType’, namely ‘Model’ and ‘Data’. We use the attribute ‘kgName’ to identify the name.
- Element ‘knowledgeGraph’ contains multiple sub-elements ‘entity’ and ‘relationship’.
- Element ‘entity’ refers to the entity concept mentioned above. It uses the attribute ‘entityType’ to determine its type. If it is specific data of data layer, it needs to use the attribute ‘entityName’ to name it. The ‘entity’ contains multiple sub-elements ‘property’.
- Element ‘relation’ refers to the relationship between entities. The attribute ‘relationName’ is used to determine the relationship name, and the sub-elements ‘head’ and ‘tail’ are used to determine the two entities of the relationship.
- Element ‘property’ represents the property of the entity, which is the concept of ‘key-value’ mentioned above, and the key is specified using the attribute ‘key’.

According to the above description, we use five-tuple to describe the knowledge graph.

- Element definition:

$$E=[knowledgeGraph,entity,relation,property,head,tail]$$

- Attribute definition:

$$A=[@kgType,@kgName,@entityType,@entityName,@relationName,@key]$$

- Data type definition:

$$t_{PROPERTY}::=ET,\ t_{ENTITY}::=sequence[(property)*:t_{PROPERTY}]$$
$$t_{RELATION}::=sequence[head:t_{ENTITY},tail:t_{ENTITY}]$$
$$t_{KNOWLEDGEGRAPH}::=sequence[(entity)*:t_{ENTITY},(relation)*:t_{RELATION}]$$

- Type function definition:

$$P(knowledgeGraph)=t_{KNOWLEDGEGRAPH},\ P(entity)=t_{ENTITY},\ P(relation)=t_{RELATION}$$
$$P(head)=t_{ENTITY},\ P(tail)=t_{ENTITY},\ P(@kgType)=string,\ P(@kgName)=string$$
$$P(@entityType)=string,\ P(@entityName)=string,\ P(@relationName)=string,\ P(@key)=string$$

- Attribute contains function definition:

$$C(knowledgeGraph)=[@kgType,@kgName],\ C(entity)=[@entityType,@entityName]$$
$$C(relation)=[@relationName],\ C(property)=[@key]$$

Use Altova XMLSpy to draw elements and relationships, as shown in figure 3.

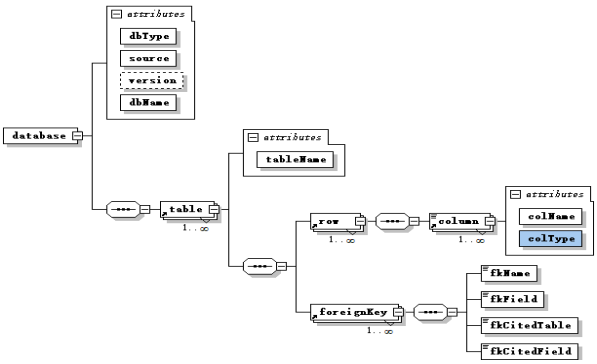


Figure 2. Database's meta-model layer

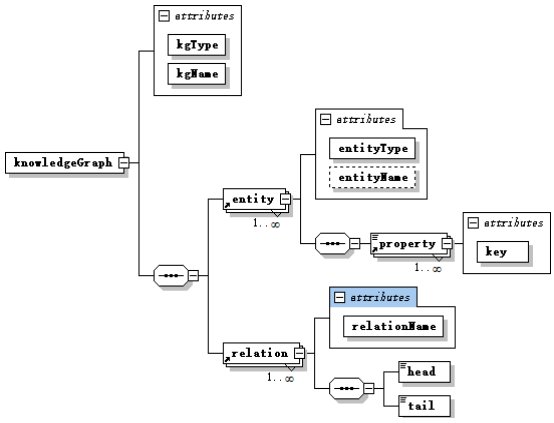


Figure 3. Knowledge graph's meta-model layer

5. Transformation rules from database to knowledge graph

After defining the elements and relationships of the meta-model layer, it is necessary to transform the models and instances in specific scenarios. The transformation rules of the model layer are determined by the element correspondence of the meta-model layer, while the transformation of the model layer determines the transformation of the instance layer.

From the database to the description file, there is basically a one-to-one correspondence, which is described in section 4.2. If specific to a certain database, it is different to describe the way of extracting model or data, so we will not describe here.

The corresponding relationship between knowledge graph description and knowledge graph is described in section 4.3. In order to convert the description file to knowledge graph, it is necessary to take out the model and data in the description file and combine the characteristics of different knowledge graph to form the corresponding model and data.

The key point of this chapter is the corresponding relationship between database description and knowledge graph description. We define polymorphic function β as shown in equation (2), where D , E_D and A_D represent data source description, meta-model elements and attributes, and K, E_K and A_K represent knowledge graph description, meta-model elements and attributes.

$$\beta(D)=K\ ,D=\{E_D,A_D\},K=\{E_K,A_K\} \tag{2}$$

Table 1. Correspondence between database and knowledge graph

	Database	Knowledge graph	Function
In common	*database* $\in E_D$	*knowledgeGraph* $\in E_K$	β(*database*)=*knowledgeGraph*
	@*dbType* $\in A_D$	@*kgType* $\in A_K$	β(@*dbType*)=@*kgType*
	@*dbName* $\in A_D$	@*kgName* $\in A_K$	β(@*dbName*)=@*kgName*
	@*tableName* $\in A_D$	@*entityType* $\in A_K$	β(@*tableName*)=@*entityType*
	column $\in E_D$	*property* $\in E_K$	β(*column*)=*property*
	@*colName* $\in A_D$	@*key* $\in A_K$	β(@*colName*)=@*key*
	fkName $\in E_D$	@*relationName* $\in A_K$	β(*fkName*)=*relationName*
In model	*table* $\in E_D$	*entity* $\in E_K$	β(*table*)=*entity*
	@*tableName* $\in A_D$	@*entityName* $\in A_K$	β(@*tableName*)=@*entityName*
In instance	*row* $\in E_D$	*entity* $\in E_K$	β(row)=*entity*

The corresponding relationship between them needs to distinguish the model layer and the instance layer. First, we describe the same part, and then describe the characteristic part. The same corresponding relationship between model layer and instance layer is as follows.

- Element *database* $\in E_D$ corresponds to element *knowledgeGraph* $\in E_K$, @*dbType* $\in A_D$ corresponds to @*kgType* $\in A_K$, and @*dbName* $\in A_D$ corresponds to @kgName $\in A_K$.
- Attribute @*tableName* $\in A_D$ corresponds to @*entityType* $\in A_K$.
- Element *column* $\in E_D$ corresponds to element *property* $\in E_K$,and @*colName* $\in A_D$ corresponds to @*key* $\in A_K$.
- Foreign key's *fkName* $\in E_D$ corresponds to relationship's @*relationName* $\in A_K$.

The characteristics corresponding relationship of model layer is as follows.

- Element *table* $\in E_D$ corresponds to element *entity* $\in E_K$, @*tableName* $\in A_D$ corresponds to @*entityName* $\in A_K$,and element *row* $\in E_D$ doesn't participate in correspondence.
- An element *foreignKey* $\in E_D$ of an element *table* $\in E_D$ corresponds to an element *relation* $\in E_K$, the *head* of the *relation* is the *table*, and the *tail* is the *entity* of the *table* specified by *fkCitedTable* of the *foreignKey*.

The characteristics corresponding relationship of instance layer is as follows.

- An element *row* $\in E_D$ corresponds to an element *entity* $\in E_K$. For the @*entityName* of element *entity*, first we establish a synonym dictionary of 'name', and then look for the *column* whose value of the @*colName* in the synonym dictionary. If it can be found, the value of @*entityName* is the value of *column*, and we delete the corresponding item of the *column* in the *property*. If it can't be found, we use the value of the *column* whose value of @*colName* is 'id' as the value of the @*entityName*.
- If there are 'n' foreign keys in the table, there is a row in the table, and 'n' relations need to be established. For the *head* of the *relation*, it is specified as the *entity* corresponding to the *row*. For *tail* of the *relation*, we need to determine the field according to the *fkField* of the *foreignKey* at first, then find the association table according to the *fkCitedTable*, and then find the association *row* according to the value of the *fkCitedField*, which is equal to the value of the *fkField*. The *entity* corresponding to the *row* is the *tail* of the *relation*.

We use the function β to formalize these correspondences, as shown in table 1.

Through the above corresponding rules, the transformation from data source to knowledge graph can be realized.

6. A case of logistics domain

In this paper, we used MySQL database and Neo4J graph database to verify the correctness of the approach in specific scenarios. We designed a logistics scenario for verification, in which the MySQL model and data are shown in figure 4. There are 6 tables in total, and the relations are as follows.

- The foreign key ‘country_id’ of table ‘province’ is associated with the ‘id’ of table ‘country’.
- The foreign key ‘province_id’ of table ‘city’ is associated with the ‘id’ of table ‘province’.
- The foreign key ‘city_id’ of table ‘district’ is associated with the ‘id’ of table ‘city’.
- The foreign keys ‘start_id’ and ‘end_id’ of table ‘price’ are associated with the ‘id’ of table ‘city’ and the foreign key ‘company_id’ is associated with the ‘id’ of table ‘company’.

id	name
1	China

(a) Table country

id	name	country_id
1	ShanDong	1
2	HeBei	1

(b) Table province

id	name	province_id
1	JiNan	1
2	WeiHai	1
3	ShiJiaZhuang	2

(c) Table city

id	name
1	ShunFeng

(d) Table compnay

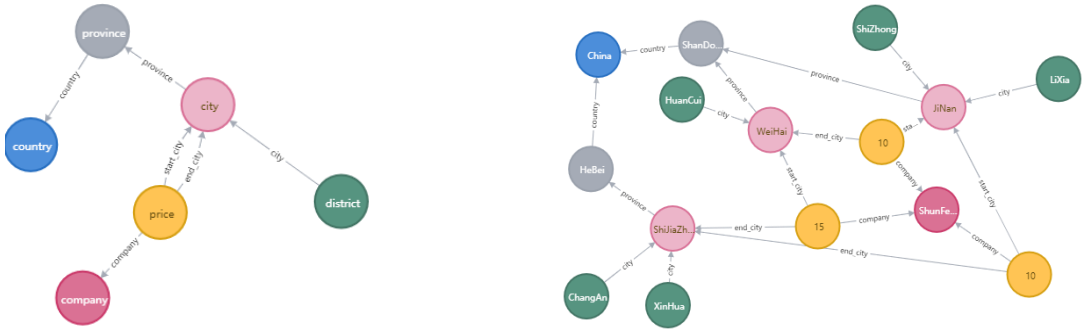
id	name	city_id
1	LiXia	1
2	ShiZhong	1
3	HuanCui	2
4	ChangAn	3
5	XinHua	3

(e) Table district

id	start_id	end_id	company_id	price
1	1	2	1	10
2	1	3	1	10
3	2	3	1	15

(f) Table price

Figure 4.MySQL’s model and data



(a) Schema layer **(b)**Data layer

Figure 5. Neo4J’s schema and data

After the transformation, the model and data are obtained in Neo4J and displayed in the form of graph. As shown in figure 5, the circles represent different types of entities, and the lines represent the relationships. The entity’s attributes are included in the entity, and aren’t reflected in the figure.

In figure 5, the left side is the schema layer, and the right side is the data layer. The table ‘country’ corresponds to the entity ‘country’ in the Neo4J’s schema layer, and its data ‘China’ corresponds to the entity ‘China’ in the Neo4J’s data layer. The correspondence between other tables and entities is similar. The foreign key ‘country_id’ of table ‘province’ corresponds to the relationship ‘country’ between the entity ‘province’ and the entity ‘country’ in the Neo4J’s schema layer and between the entity ‘ShanDong’ and the entity ‘China’ in the Neo4J’s data layer. Other relationships are similar.

7. Conclusion

In this paper, we use the concept of model driven to transform the relational database into knowledge graph, and make scattered knowledge graphical, which can help users better understand the domain knowledge; XML and XML Schema are used to describe the data source and knowledge graph, and the transformation from data source to knowledge graph is realized. The experimental results show that the approach can well realize the construction of vertical domain knowledge graph.

Acknowledgments
This work was supported by the National Key R&D Program of China under grant 2018YFB1403104.

References
[1] Sinahal, Amit. Introducing the Knowledge Graph: Things, Not Strings.Official Blog (of Google),2012.
[2] Peng Z. Research on Ontology Automatic Construction Based on Relational Database[D]. University of Science and Technology of China, 2019.
[3] Arenas, M., Bertails, A., Prud, E., Sequeda, J.: A Direct Mapping of Relational Data to RDF.W3C Recommendation 27 September 2012, http://www.w3.org/TR/rdb-direct-mapping/.
[4] Souripriya Das, O., Seema Sundara, O., Cyganiak, R.: R2RML: RDB to RDF mapping language.W3C, http://www.w3.org/TR/r2rml/.
[5] Hazber M A G, Li R, Xu G, et al. An Approach for Automatically Generating R2RML-Based Direct Mapping from Relational Databases[M]// Social Computing. Springer Singapore, 2016.
[6] Ying W, Sizhu W. Converting STKOS Metathesaurus to RDF Triples with R2RM[J]. Data Analysis and Knowledge Discovery, 2018, 2(12): 89-97.
[7] Octaviani D, Pranolo A, Othman S. RDB2Onto: An approach for creating semantic metadata from relational educational data[C]// 2015 International Conference on Science in Information Technology (ICSITech). IEEE, 2015.
[8] Jiaqing W,Weidong Y,Yizheng H. Entity Relationship Extraction in Relational Database[J]. Computer Applications and Software, 2019, 36(10): 10-16, 38.
[9] Naijing Z, Ping J. Constructing forestry domain ontology from relational database[J]. Computer Engineering And Design, 2017(07):274-282.
[10] Upadhyaya S R , Kumar P S . ERONTO: a tool for extracting ontologies from extended E/R diagrams[C]// Proceedings of the 2005 ACM Symposium on Applied Computing (SAC), Santa Fe, New Mexico, USA, March 13-17, 2005. ACM, 2005.
[11] Rodrigues T , Rosa P , Cardoso J , et al. Mapping XML to existing OWL ontologies[C]// 2006.
[12] Battle S . Gloze: XML to RDF and back again[C]// Jena User Conference. 2006.
[13] An Y , Borgida A , Mylopoulos J . Constructing Complex Semantic Mappings Between XML Data and Ontologies[M]// The Semantic Web – ISWC 2005. Springer Berlin Heidelberg, 2005.