Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

VQT: value cardinality and query pattern based R-schema to XML schema translation with implicit referential integrity*

Jinhyung KIM[1], Dongwon JEONG[2], Doo-Kwon BAIK‡1

([1]*Department of Computer Science and Engineering, Korea University, Seoul 136-713, Korea*)

([2]*Department of Informatics and Statistics, Kunsan National University, Kunsan 573-701, Korea*)

E-mail: jinhyung98.kim@gmail.com; djeong@kunsan.ac.kr; baikdk@korea.ac.kr

Received Jan. 15, 2008; revision accepted Apr. 30, 2008; CrossCheck deposited Nov. 10, 2008

Abstract: In this paper, we propose a new relational schema (R-schema) to XML schema translation algorithm, VQT, which analyzes the value cardinality and user query patterns and extracts the implicit referential integrities by using the cardinality property of foreign key constraints between columns and the equi-join characteristic in user queries. The VQT algorithm can apply the extracted implied referential integrity relation information to the R-schema and create an XML schema as the final result. Therefore, the VQT algorithm prevents the R-schema from being incorrectly converted into the XML schema, and it richly and powerfully represents all the information in the R-schema by creating an XML schema as the translation result on behalf of the XML DTD.

Key words: Value cardinality, Query pattern, Relational schema, XML schema, Implicit referential integrity relations, Explicit referential integrity

doi:10.1631/jzus.A0820036 **Document code:** A **CLC number:** TP31

INTRODUCTION

Since XML has emerged as a formidable data format in the Internet era, there is a considerable increase in the amount of data encoded in XML (Bray *et al.*, 2000; ISO/IEC, 2001). However, a majority of data is still stored and maintained in relational databases (Elmasri and Navathe, 2003). Therefore, we need to translate such relational data into XML documents (Andersson, 1995; Widom, 1999; Seligman and Rosenthal, 2001; Fan and Simeon, 2003; Witkowski *et al.*, 2005; Buche *et al.*, 2006; Prakash *et al.*, 2006; Zheng *et al.*, 2006). In the Relational schema (R-schema) to XML schema translation, a particularly complex problem appears when the applications are old or ill-designed and poorly documented (Chen *et al.*, 2006; Madiraju *et al.*, 2006). The implicit referential integrity (RI) relation is the relation necessary for exacting R-schema to XML schema translation, but it is not defined in the R-schema because of the designer's fault or old and poor documentation (Hainaut *et al*., 1993; 1996; Kim *et al*., 2005; 2007). Many translation methods have been developed taking into account structural and/or semantic aspects. However, earlier methods have considered only the explicit RI specified by the R-schema during translation or partially reflect the implicit RI. It causes several problems such as incorrect translations, abnormal relational model transitions. In addition, many conventional translation algorithms support XML DTD as the final translation result. However, it is insufficient to exactly represent the information of the R-schema. In this paper, we propose a new R-schema to XML schema translation algorithm, VQT, considering the implicit RI relation information by analyzing the cardinality between the data values and patterns of user queries. The VQT algorithm consists of

‡ Corresponding author

* Project supported by the 2nd Brain Korea Project

three parts: value cardinality analysis (VC module), query pattern analysis (QP module), and XML translation module. The RIs are realized by the cardinality between a PK field (defined as the primary key constraint) and an FK field (defined as the foreign key constraint), which exhibits a 1:*N* relation with respect to the VC module. The RIs can also be achieved by the equi-join characteristic of the QP module. According to this characteristic, columns related via equi-join in the user queries can exhibit semantic interrelationships. By using the VQT algorithm, we can achieve better accuracy and lower loss ratio as compared to those obtained from conventional algorithms. Moreover, the VQT algorithm can support more powerful and detailed translation results with the XML schema on behalf of the XML DTD.

RELATED WORKS

Various translation algorithms have been developed taking into account structural and/or semantic aspects. Generally, we can classify conventional translation methods into three categories: user-specific translation methods, structural translation methods, and semantic translation methods.

User-specific translation methods need specification inputs about mapping rules for R-schema to XML schema translation: XML Extender from IBM, XML Database Management System (XML-DBMS) (Naughton *et al.*, 2001), Trading between Relations and XML (SilkRoute) (Fernandez *et al.*, 2000), Publishing Object-Relational Data as XML (XPERANTO) (Carey *et al.*, 2000), and a tool for transforming relational databases into XML document (DB2XML) (Turau, 2002) are included in this group. In XML Extender, the users must define mapping rules by using data access definition (DAD) files or XML extender transform languages. A template-based mapping language is provided for the specification of the mapping rule in XML-DBMS. SilkRoute provides a declarative query language for the description of relational data. XPERANTO uses an XML query language for data searching in XML. DB2XML is similar to Flat Translation (FT), but it needs user specification for mapping. Evidently, user-specific translation methods have the drawback that users have to additionally input the rules for mapping.

FT (Lee *et al.*, 2001; 2003) and NeT (Nesting-based Translation) (Lee *et al.*, 2001; 2002; 2003) are typical algorithms used in structural translation methods. FT is the simplest method for R-schema to XML document translation by 1:1 mapping. By using the FT algorithm, tables and columns in the R-schema are changed to elements and attributes in the XML schema, respectively. The core idea of the NeT algorithm is to find an appropriate model by using nesting operators such as "*" and "+" (Hunter *et al.*, 2004). Therefore, we observe that NeT is useful for decreasing the data redundancy and obtaining a "more intuitive" schema by removing redundancies induced by multi-valued dependencies and performing grouping on the attributes.

CoT (Constraints-based Translation) (Lee *et al.*, 2002; 2003) and Relation Conversion to XML-nested Structures (ConvRel) (Duta *et al.*, 2004) are included in the semantic translation methods. The CoT algorithm is mostly associated with the usage of sub-elements and IDREF attributes for translation purposes. For two distinct tables s and t with lists of columns X and Y, respectively, suppose we have a foreign key constraint $s[\alpha]\subseteq t[\beta]$, where $\alpha\in X$ and $\beta\in Y$. Further, suppose that $K_S\in X$ is the key for s. Then, rewriting the above notations as follows: $T=\{s, t\}$, $C(s)=\{X\}$, $C(t)=\{Y\}$, $\Delta=\{s[\alpha]\subseteq t[\beta]$, $\beta \xrightarrow{\text{key}} t$, $K_S \xrightarrow{\text{key}} s\}$. That is, the CoT algorithm considers not only the structural part such as tables and columns but also the semantic part such as constraints and RI. However, the CoT algorithm can only reflect the explicit RI. If the implicit RI exists, the CoT algorithm cannot create an exact XML document. The ConvRel algorithm analyzes the relation between the tables and extracts the RI information by using the parent-child relationship properties. In the ConvRel algorithm, possible XML structures are classified and encoded as follows:

Class 1 designs the Parent→Child nested structure;

Class 2 designs the Child→Parent nested structure;

Class 3 designs the XML flat structure using *Keyref* references;

Class 4 designs additional Parent→Child nested structures for the *M*:*N* relationships modeled as a combination between a nested structure and a *Keyref* reference.

PRINCIPAL ISSUE

In this section, we describe issues related to implicit RI information with simple datasets. A simple dataset can be illustrated as the initial R-schema model shown in Definition 1. Definition 1 describes the detailed elements of the initial R-schema model, and Example 1 describes the initial R-schema model concerning the simple dataset.

Definition 1 (Initial R-schema model) The R-schema and data values can be defined as the initial R-schema model. An initial R-schema model is denoted by a 7-tuple R_i=(T, C, P, RI_{exp}, K, DV, UQ), where T is a finite set of table names; C is a function that represents a set of column names in each table; K is a function that represents the primary key information; P is a function that represents the properties of each column and the result of P consists of a 3-tuple (t, u, n), with t representing the data type of a column such as an integer, a string, u representing the value of a column to be unique (u) or not unique (~u), and n representing the nullability of the value of a column as nullable (n) or non-nullable (!n); RI_{exp} represents explicit information about the referential integrities; DV is a function that represents a set of data values for each column; UQ represents user query lists stored in the DBMS.

Example 1 The initial R-schema model of simple datasets is defined as follows:

T={*Student*, *Professor*, *Class*, *TakingLecture*, *Project*, *Room*}, C(*Student*)={*SID*, *Sname*, *PID*}, C(*Professor*)={*PID*, *Pname*, *Office*}, C(*Class*)={*Cname*, *Room*, *Time*}, C(*Project*)={*Projname*, *PID*, *SID*}, C(*TakingLecture*)={*Seq_ID*, *SID*, *Cname*}, C(*Room*)={*Place*, *Usage*}, K(*Student*)={*SID*}, K(*Professor*)={*PID*}, K(*Class*)={*Cname*}, K(*Project*)={*Projname*}, K(*TakingLecture*)={*Seq_ID*}, K(*Room*)={*Place*}, RI_{exp}={(*TakingLecture.Cname*, *Class.Cname*), (*Student.PID*, *Professor.PID*), (*TakingLecture.SID*, *Student.SID*)}, P(*SID*)={*string*, u, !n}, P(*Sname*)={*string*, ~u, !n}, P(*PID*)={*string*, ~u, !n}, P(*Pname*)={*string*, ~u, !n}, P(*Office*)={*integer*, u, n}, P(*Seq_ID*)={*integer*, u, !n}, P(*Cname*)={*string*, u, !n}, P(*Room*)={*integer*, u, !n}, P(*Time*)={*integer*, ~u, n}, P(*Projname*)={*string*, u, !n}, P(*PID*)={*string*, ~u, !n}, P(*SID*)={*string*, ~u, !n}, P(*Place*)={*number*, u, !n}, P(*Usage*)={*string*, ~u, n}.

This relational database consists of 6 tables and 17 columns. Each student can take one or more classes, and must have an academic adviser. Each professor can teach one or more students. The office column can be null. Each project is related to one or more students and professors. In addition, students can participate in a project with not only an academic adviser but also other professors. Three explicit RIs are evident in Example 1: {(PID_{pro}, PID_{stu}), ($Cname_{tak}$, $Cname_{cla}$), and (SID_{stu}, SID_{tak})}. We can translate the sample R-schema into the XML DTD by using NeT and CoT.

The translation results by the NeT algorithm and the CoT algorithm are shown in Table 1. NeT cannot reflect the RI during the R-schema to XML DTD translation. As shown in Table 1, NeT can simply execute structural translation. In other words, NeT cannot create exact XML documents: it loses all RI information during the translation. CoT can extract and partially reflect the RI information. However, CoT simply extracts the explicit RI information during translation. The underlined part in Table 1 shows the explicit RI information (*Student.PID*⊂*Professor.PID*, *TakingLecture.Cname*⊂*Class.Cname*, *TakingLecture.SID*⊂*Student.SID*). However, we need to extract and reflect not only the explicit RI information but also the implicit RI information (*Project.PID*⊂*Professor.PID*, *Project.SID*⊂*Student.SID*, *Class.Room*⊂*Classroom.Place*) for a more accurate translation. In addition, NeT and CoT only support

Table 1 Translation results of NeT and CoT

Algorithm	Translation result
NeT	<!ELEMENT Student (SID, Sname, PID+, Cname*)> <!ELEMENT Professor (PID, Pname, Office?)> <!ELEMENT Class (Cname, Room, Time)> <!ELEMENT TakingLecture (Seq_ID, SID, Cname)> <!ELEMENT Project (Projname, PID, SID)> <!ELEMENT Classroom (Place, Usage)>
CoT	<!ELEMENT Student (SID, Sname, Taking-Lecture*)> <!ELEMENT Professor (PID, Pname, Office, Student*)> <!ELEMENT Class (Cname, Room, Time)> <!ATTLIST Class ID Class ID> <!ELEMENT TakingLecture (Seq_ID, Cname)> <!ATTLIST TakingLecture Ref Class IDREF> <!ELEMENT Project (Projname, PID, SID)> <!ELEMENT Classroom (Place, Usage)>

XML DTD as a translation result. There is a limitation because of which we cannot represent R-schema information such as the data type, and cardinality. Therefore, we need to develop a new translation algorithm that supports XML schema as a translation result.

VQT ALGORITHM

The VQT algorithm consists of three core modules (and eight steps in total): the VC module, QP module, and XML translation module. The VC and QP modules have several detailed processes for extracting the implicit RI information. The VC and QP modules are executed in parallel, and they complement each other. The extraction results obtained using the VC and QP modules are integrated and translated into the XML schema by the XML translation module using translation rules. In this section, each step in the VC or QP module is described with an example. Due to space limitation, for each example, a part of the entire translation process is given.

VC module

The VC module obtains the metadata of the R-schema and analyzes the cardinality between the data values. A VC module process consists of four steps: preprocessing, WordNet-based column extraction, candidate extraction, and refinement.

1. Preprocessing

In this step, we select and determine the targets for an efficient comparison. First of all, we obtain the metadata information of the relational database from a data buffer cache and a shared SQL area in the DBMS. The metadata information includes the names of the tables and columns, properties of the columns (data type and nullable information), foreign key constraints, and so on (Table 2). Second, we select one of the primary key columns for comparison. Based on the selected primary key column, we select the target columns for comparison. The primary key columns of the other tables are eliminated from the comparison target because the values of the primary key column in one table and those in the other table cannot have the same values. The columns that have value types different from that of the selected primary key column are also removed. The columns that have explicit RI with the selected primary key column are removed, because we do not need to extract them again. The preprocessing algorithm used in the VC module is given as follows:

Table 2 Summary of notations and symbols

Notation	Description
T_i	Name of the ith table
$T_i.C_j$	Name of the jth column in the ith table
$T_i.C_j.V_k$	Name of the kth data instance of the jth column in the ith table
St_i	Name of the primary key column in the ith table
$Synset_{ij}$[]	Lists of similar words and words with the same meaning of the jth column in the ith table extracted by WordNet
$Synset_{PKi}$[]	Lists of similar words and words with the same meaning of the primary key column in the ith table extracted by WordNet
ECL[]	Excepted column lists in the VC module
RI_{exp}[]	Explicit RI information
RI_{can}[]	Candidate lists of implicit RI information
$T_{i\text{INFO}}$	All the table lists in the initial relational database
$C_{\text{Info}}T_i$[]	All the column lists in the ith table
$T_i.C_j.ty$	Data type of the jth column in the ith table
$T_i.C_j.nu$	Nullable information of the jth column in the ith table
Q_L[]	User query lists in the DBMS
Q_T[]	Tokenized user query lists
Q_{WH}[]	"Where" clause set lists of user query lists
Q_{dis}[]	Unnecessary user query lists
$RI_{\text{IMP-VC}}$[]	Implicit RI information extracted by the VC module in the VQT algorithm
$RI_{\text{IMP-QP}}$[]	Implicit RI information extracted by the QP module in the VQT algorithm
GetMetadata()	Function for getting the metadata information of R-schema from the DBMS
GetQueryLists()	Function for getting the user query lists of R-schema from the DBMS
SearchWordNet()	Function for searching lists with similar words and words, with the same meaning from WordNet
CompareSynset()	Function for comparison between word lists (synset) extracted by WordNet
Translate()	Function for translation from R-schema to XML schema
Setselectorpath()	Function for setting a selector path in XML schema
Setfieldpath()	Function for setting a field path in XML schema
CreateAttribute()	Function for creating an attribute in an element
Levelup(), *Leveldown*()	Function for moving to an upper level (lower level) in the hierarchical structure of the element

Algorithm 1 Preprocessing of the VC module

Initialize i=1, j=1, k=1, n=1, m=1, a=1, b=1;
Do (while $i<i_{(max)}$)
  select T_i;
  set $T_i.PK$ to St_i;
  For j=1 to $j_{(max)}$
    GetMetadata();
    $ECL[a]=(T-T_i).PK$;
    increase a;
    If ($St_i.ty\neq(T-T_i).C_j.ty$)
      $ECL[a]=(T-T_i).C_j$;
      increase a;
    End If
  Next j
  $ECL[a]=RI_{exp}[b]$;
  increase a, b;
Loop

Example 2 Preprocessing of the VC module about Example 1 is described as follows:

➢ Primary key column: K(*Student*)={SID_{stu}}

(1) Excepted column lists

Primary key columns in other tables: {PID_{pro}, $Cname_{cla}$, *Projname*, *Seq_ID*, *Place*};

Different data type: {*Office*, *Room*, *Time*, *Place*};

Explicit RI: {SID_{tak}}.

(2) Comparison target column list: {*Pname*, PID_{prj}, SID_{prj}, $Cname_{tak}$, *Usage*}

2. WordNet-based column extraction

In the WordNet-based column extraction step, we extract the semantics between the columns using WordNet. In this step, we use two important functions: *SearchWordNet*() and *CompareSynset*(). The input to the *SearchWordNet* function is all the column names, and its output is the synset of each column that lists words with similar or same meaning extracted from WordNet. *CompareSynset*() is a function for performing a comparison between the word lists (synset) extracted by WordNet. Synsets extracted by the *SearchWordNet* function are stored in $Synset_{ij}$[], and each synset is compared by using the *CompareSynset* function. If one of the elements in the synset of some column is the same as one of the elements in the synset of another column, these two columns are defined as an implicit RI candidate and stored into RI_{can}[]. The algorithm for WordNet-based column extraction in the VC module is given as follows:

Algorithm 2 WordNet-based column extraction in the VC module

Initialize i=1, j=0, k=1, o=1, p=0, q=0;
For i=1 to $i_{(max)}$
  For j=1 to $j_{(max)}$
    For k=1 to $k_{(max)}$
      $Synset_{ij}[]=SearchWordNet(T_i.C_j)$;
    Next k
  Next j
Next i
For i=1 to $i_{(max)}$
  For j=1 to $j_{(max)}$
    For k=1 to $k_{(max)}$
      For o=1 to $o_{(max)}$
        For p=1 to $p_{(max)}$
          Do (while $i\neq k$)
            CompareSynset($Synset_{PKi}[j]$, $Synset_{ko}[p]$);
            If ($Synset_{PKi}[j]==Synset_{ko}[p]$)
              $RI_{can}[q]=(St_i, T_k.C_o)$;
              increase q;
            End If
          Loop
Next p, o, k, j, i

Example 3 WordNet-based column extraction in the VC module about Example 1 is represented as follows:

➢ Extraction of synsets of columns

Synset(*SID*)=*Synset*(*Sname*)=*Synset*(*Pname*)=*Synset*(*Cname*)=*Synset*(*Seq_ID*)=∅;

Synset(*Place*)={*spot*, *property*, *stead*, *position*, *lieu*, *shoes*, *home*, *station*, *seat*, *space*};

Synset(*Office*)={*office*, *business office*, *agency*, *place*, *situation*, …};

Synset(*Room*)={*way*, *elbow room*, *place*}.

➢ Comparison between the primary key synset and other synsets

K(*Classroom*)={*Place*}, *Synset*(*Place*)=*Synset*(*Office*)=*Synset*(*Room*)={*place*};

(*Office*, *Place*) and (*Room*, *Place*) are extracted as RI candidates.

3. Implicit RI candidate extraction

In the implicit RI candidate extraction, we repeatedly compare the selected primary key column with the target columns until the comparison is completed for every column in the initial R-schema. We extract an implicit RI candidate based on the comparison result. If the selected primary key column and a certain column have the same value, these two columns are defined as the implicit RI candidate. The

final implicit RI will be decided at the refinement step. The algorithm for candidate extraction of the VC module is given as follows:

Algorithm 3 Candidate extraction of the VC module

```
Initialize i=1, j=1, k=1, n=1, m=1, a=1, b=1;
For j=1 to j(max)
  For k=1 to k(max)
    For n=1 to n(max)
      If (St.vk=((T−Ti).Cj−ECL[a]).Vn)
        RIcan[m]=(Sti, (T−Ti).Cj);
          //T is a set of tables
          //T−Ti is a set of tables except for Ti
        increase m;
      End If
    Next n
  Next k
Next j
increase i;
```

Example 4 Implicit RI candidate extraction of the VC module about Example 1 is represented as follows:

➢ Primary key column: K(*Student*)={SID_{stu}}

(1) Comparison target column list: {*Pname*, PID_{prj}, SID_{prj}, $Cname_{tak}$, *Usage*}

(2) Comparison: {SID_{stu}, *Pname*}, {SID_{stu}, PID_{prj}}, {SID_{stu}, $Cname_{tak}$}, {SID_{stu}, *Usage*}

No RI.

(3) Comparison: {SID_{stu}, SID_{prj}}, DV(SID_{prj})={s01, s02, s03, s04, s03}

RI candidate.

4. Implicit RI refinement

In the implicit RI refinement, we check the 1:N cardinality between the columns in the implicit RI candidate extracted in the second and third steps. Generally, if the values of a primary key column appear in the other columns many times, the implicit RI candidate is finally defined as the implicit RI. This result is based on the general property of the foreign key constraint that the values in the PK field and those in the FK field have 1:N cardinality. The algorithm for the refinement of the VC module is given as follows:

Algorithm 4 Refinement of the VC module

```
Initialize m=2, p=1;
RI[1]=RIcan[1];
Do (while m<m(max))
  For p=1 to p(max)
    If (RIimp−VC[m]=RIcan−VC[p])
      increase p;
    Else RIdis[m]=RIcan−VC[p];
    End If
    increase m;
Loop
```

Example 5 Implicit RI refinement of the VC module about Example 1 is illustrated as follows:

➢ Implicit RI candidate column lists

{SID_{stu}, SID_{prj}}, {PID_{pro}, PID_{prj}}, {*Place*, *Office*}, {*Place*, *Room*}.

➢ Refinement of candidate

{SID_{stu}, SID_{prj}}, SID_{stu} and SID_{prj} have 1:N relation regarding the "s03" value;

Implicit RI is concluded.

QP module

The QP module receives user query lists using the shared SQL area in the DBMS. The QP module analyzes the user query lists and extracts the implicit RI information using the equi-join property. A process in the QP module consists of three steps: resource generation, candidate extraction, and refinement. In addition, the QP module can complement the limitation of the VC module.

1. Resource generation

In the resource generation, we create new resources for the implicit RI candidate by analyzing the query pattern. First, we get a user query list from the shared SQL area in the DBMS and store queries in a "Query Stack". We tokenize the query lists and extract "where" the clause came from in the query lists and store them in the "Where Clause Stack (Array)". The algorithm for resource generation in the QP module is given as follows:

Algorithm 5 Resource generation in the QP module

```
Initialize a=1, b=1, c=1, d=1, g=1, n=1, m=1, j=1;
Do (while b<b(max))
  QL[a]=GetQueryList(b);
  increase a, b;
  For c=1 to c(max)
    For d=1 to d(max)
      QT[c][d]=GetToken(QL[c], d);
    Next d
  Next c
  For c=1 to c(max)
    For d=1 to d(max)
      If (QT[c][d]='where')
        For n=1 to n(max)
          QWH[c][n]=QT[c][d+n];
        Next n
      End If
    Next d
  Next c
```

Example 6 Resource generation in the QP module about Example 1 is illustrated as follows:

➢ Where clause stack (array)

Where *Professor.PID=Project.PID*;

Where *PID*=p04.

2. Implicit RI candidate extraction

In candidate extraction, we analyze the "Where Clause Stack" and extract the columns connected by the "=" operator, where the "=" operator denotes the equi-join operation. In addition, we eliminate the columns that are connected by the "=" operator but not related by the equi-join operation such as "*PID*=03". Except for equi-join, the column and data values can also be represented by the "=" operator. We extract columns related to equi-join are stored in the "Equi-join Stack". The implicit RI candidate obtained from the QP module can be represented as a pair of columns such as (*Professor.PID, Project.PID*). The algorithm for candidate extraction in the QP module is given as follows:

Algorithm 6 Candidate extraction in the QP module

For g=1 to $g_{(max)}$
  For j=1 to $j_{(max)}$
  If ($Q_{WH}[g][j]$.*equalTo*("="))
    For m=1 to $m_{(max)}$
      If ($Q_{WH}[g][j-3]|Q_{WH}[g][j-1] | Q_{WH}[g][j+1] | Q_{WH}[g][j+3]$==0)
      Then Next g
      Else RI_{can}−$QP[m]$=($Q_{WH}[g][j-3]$.$Q_{WH}[g][j-1]$, $Q_{WH}[g][j+1]$.$Q_{WH}[g][j+3]$);
      Next m
  Else Next j
Next g
Initialize m=2, p=1;
$RI[1]=RI_{can}-QP[1]$;

Example 7 Implicit RI candidate extraction in the QP module about Example 1 is shown as follows:

➢ Candidate extraction

(1) Where *Professor.PID=Project.PID*, {*Professor*.PID, *Project.PID*}

Implicit RI candidate.

(2) Where *PID*=p04

Removed because of not related by equi-join.

3. Implicit RI refinement

In the implicit RI refinement, we refine the column list extracted in the previous steps and select the implicit RI using the equi-join properties. These properties imply that the two columns related by equi-join are semantically interconnected, and an RI can exist between these two columns. An algorithm for the refinement of the QP module is given as follows:

Algorithm 7 Refinement of the QP module

Do (while $m<m_{(max)}$)
  For p=1 to $t_{(max)}$
    If ($RI_{imp}-QP[m]=RI_{can}-QP[p]$)
      increase p;
    Else $RI_{dis}[m]=RI_{can}-QP[p]$;
    End If
    increase k;
  Next p
Loop

Example 8 Implicit RI refinement of the QP module about Example 1 is shown as follows:

➢ Implicit RI candidate column lists

{*Professor.PID, Project.PID*} is extracted three times.

➢ Refinement of candidate

{*Professor.PID, Project.PID*} is concluded as the implicit RI relation.

XML translation module

In the XML translation module, the RI, table, column, column type, column cardinality, and column nullable information in the R-schema are translated into the XML schema via translation rules. The translation rules are described in Table 3.

The PK column is translated into the "Key" field and the FK column is translated into the "Keyref" field. In the XML schema, the locations or paths of the columns and attributes related to the RI are defined as the XPath form. The tables and columns are hierarchically translated into elements, and the type, cardinality, and nullable information are also represented in the XML schema in detail. The algorithm for XML translation of the XML translation module is given in Algorithm 8.

Table 3 XML schema translation rules

Category	Classification	Item	XML schema
Referential integrity (explicit and implicit)	Primary key column	Name	<xsd:Key name>
		Column path	<xsd:selector xpath>
		Attribute path	<xsd:field xpath>
	Foreign key column	Name	<xsd:Keyref name>
		Column path	<xsd:selector xpath>
		Attribute path	<xsd:field xpath>
Relational DB table	Table	Name	<xsd:element name>
		Type	<xsd:element type>
Relational DB column	Not PK or FK column	Name	<xsd:element name>
		Type	<xsd:element type>
		Cardinality	<xsd:element (min|max) Occurs>
	PK or FK column	Name	<xsd:element name>
		Type	<xsd:element type>
		Cardinality	<xsd:element (min|max) Occurs>
		Attribute name	<xsd:attribute name>
		Attribute type	<xsd:attribute type>
		Attribute nullable info	<xsd:attribute use>

Algorithm 8 Translation of the XML translation module

Initialize c=1, d=1;
Load b, m;
For b=1 to $b_{(max)}$
  For c=1 to 3
    setDelimiter(",");
    RIE[b][c]=*Tokenize*(RI_{exp}[b], *delimiter*, c−1);
    If (d=1)
      set *RIE*[b][c] to PKE_b;
      Translate(PKE_b, Key_{Eb});
      Setselectorpath(Key_{Eb});
      Setfieldpath(Key_{Eb});
    Else if (d=3)
      Set *RIE*[b][c] to FKE_b;
      Translate(FKE_b, $Keyref_{Eb}$);
      Setselectorpath($Keyref_{Eb}$);
      Setfieldpath($Keyref_{Eb}$);
  increase c;
increase b;
For m=1 to $m_{(max)}$
  For d=1 to 3
    RII[m][d]=*Tokenize*(RI_{imp}[m], *delimiter*, d−1);
    If (d=1)
      set *RII*[m][d] to PK_{lm1};
      Translate(PK_{lm}, Key_{lm});
      Setselectorpath(Key_{lm});
      Setfieldpath(Key_{lm});
    Else if (d=3)
      set *RII*[m][d] to PK_{lm3};
      Translate(PK_{lm}, $Keyref_{lm}$);
      Setselectorpath($Keyref_{lm}$);
      Setfieldpath($Keyref_{lm}$);
  increase d;
increase m;
For i=1 to $e_{(max)}$
  Translate(T_{iInfo}, *element*);
  LevelDown();
  Translate($CInfoT_i$, *element*);
  For j=1 to $j_{(max)}$
    Translate($T_i.C_j.ty$, *type*);
    Translate($T_i.C_j.nu$, *use*);
    If ($CInfoT_i$=(*Key* | *Keyref*))
      CreateAttribute($CInfoT_i$);
      increase j;
    Else increase j;
  Levelup();
increase i;

VQT SYSTEM ARCHITECTURE

Fig.1 shows a system architecture supporting the VQT algorithm. First of all, the VQT translator receives the metadata of the database instances from the data buffer cache and data dictionary area in the DBMS. The data dictionary area includes information about the tables and views, and the data buffer cache comprises table information related to the SQL statements. The VQT translator consists of three modules: VC module, QP module, and XML translation module.

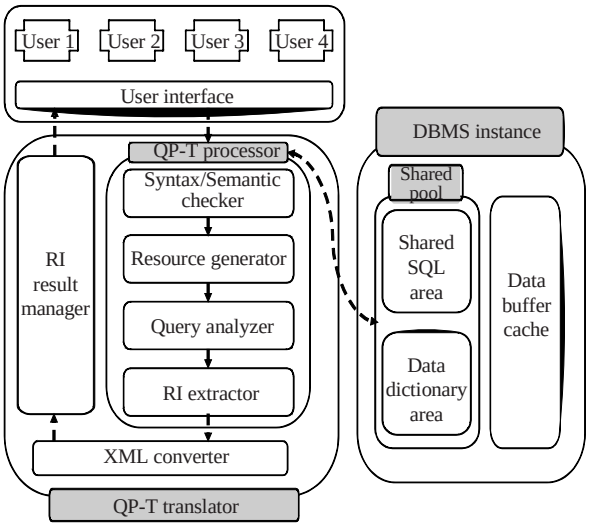

Fig.1 VQT system architecture

The VC module consists of four components: preprocessor (PP), comparison resource generator (CRG), value analyzer (VA), and RI extractor. The PP receives the metadata information of the database instance from the DBMS instance and stores this information into a memory space. The PP also checks whether there is a syntax problem. The CRG creates and filters the resources for comparison, removes columns not required for the comparison, and removes the primary columns in the other tables and columns that have other type instances. In addition, the CRG eliminates the columns related to the explicit RI information that is already defined in the initial relational schema model. The VA analyzes the value pattern, compares every column instances among the comparison target columns, and extracts the implicit RI information candidates based on the value analysis results. The RI extractor (RIE_{VC}) examines the implicit RI information candidates and extracts the implicit RI by 1:*N* cardinality of the foreign key constraints. The execution of the RIE is performed semi-automatically. The efforts of the designer or expert can be added to the RIE for refinement with better accuracy.

The QP module also consists of four components: syntactic and semantic checker (SSC), resource generator (RG), query analyzer (QA), and RI extractor (RIE_{QP}). The SSC parses the queries received from the DBMS and checks whether there is any syntactic or semantic error in the queries. The RG separates the "Where" clause from the original queries and creates a new resource for the analysis of the query pattern. The QA analyzes the query pattern of the user queries that exhibit the equi-join property and extracts the column pairs related to equi-join as the implicit integrity relation candidates. The QA also eliminates the columns not related via equi-join. The RIE_{QP} refines the implicit integrity relation candidates and decides the final implicit integrity relation.

The results of the VC and QP modules are integrated and sent to the XML translation module (XTM). The XTM translates the R-schema into the XML schema using the translation rules with explicit and implicit RI information. Finally, the RI result manager (RIRM) transmits the translated XML schema to the user interface.

PERFORMANCE EVALUATION

Formal verification

In this section, we use set theory to formally verify the implicit RI information extracted by the VQT algorithm. Set theory is the mathematical theory of sets, which represents a collection of abstract objects. In most modern mathematical formalisms, set theory provides the language in which mathematical objects are described. It is (along with logic and predicate calculus) one of the axiomatic foundations for mathematics, allowing mathematical objects to be constructed formally from the undefined terms of "set". It is a branch of mathematics in its own right and an active field of mathematical research (Deschrijver, 2007).

First, for the formal verification of the extraction process of the VQT algorithm, we need to formulate the database model. The database model consists of four elements: database schema, initial RI, extracted RI, and database language.

Definition 2 A database model MD is a set of database schemas $S=\{R_1, R_2, \ldots, R_n\}$. The database schema is a set of rules describing the structural format of the database. The database model MD is also a set of RI information defined initially as $RI_{\text{ini}}=\{A_{ij} \xrightarrow{RI_{\text{ini}}} A_{mn}\}$. The RI information extracted by the VC and QP modules can be added to the data model as $RI_{\text{VC}}=\{A_{ij} \xrightarrow{RI_{\text{VC}}} A_{mn}\}$ and $RI_{\text{QP}}=\{A_{ij} \xrightarrow{RI_{\text{QP}}} A_{mn}\}$.

Definition 3 A database schema R_i consists of the relation name and a list of attributes $R_1=\{A_{11}, A_{12}, \ldots, A_{1n}\}$ and $R_2=\{A_{21}, A_{22}, \ldots, A_{2m}\}$. Each attribute A_i includes many instances as $A_i=\cup INS(A_{in})$. Each attribute A_i is also associated with its domain $Dom(A_i)$. $Dom(MD)$ can be defined as $Dom(MD)=Dom(A_{11})\times Dom(A_{12})\times\ldots\times Dom(A_{nm})$.

Definition 4 The domain includes the data type and semantic description information, and has the following property: iff $A_{11}\neq A_{12}$, then $Dom(A_{11})\neq Dom(A_{12})$ and thus iff $Dom(A_{11})\cap Dom(A_{12})=\varnothing$, then $A_{11}\neq A_{12}$.

Example 9 The data model definition about the simple dataset shown in Algorithm 1 using Definitions 1~3 is as follows:

$MD=\{S, RI_{\text{ini}}, RI_{\text{VC}}, RI_{\text{QP}}, LS\}$;

S={*Student*, *Professor*, *Class*, *TakingLecture*, *Project*};

Student={*SID*, *Sname*, *PID*}, *Professor*={*PID*, *Pname*, *Office*}, *Class*={*Cname*, *Room*, *Time*}, *TakingLecture*={*Seq_ID*, *SID*, *Cname*}, *Project*={*Projname*, *PID*, *SID*};

INS(*SID*)={s01, s02, s03, s04, s05, s06}, *INS*(*Sname*)={Tom, John, Cathy, Brown, Cabin, Tom}, *INS*(*PID*)={p01, p02, p02, p03, p03, p04};

INS(*PID*)={p01, p02, p03, p04}, *INS*(*Pname*)={*Kim*, *Lee*, *Park*, *Jean*}, *INS*(*Office*)={217, 633, 121, 222};

INS(*Cname*)={*Database*, *Automata*, *Simulation*, *Algorithm*}, *INS*(*Room*)={701, 702, 703, 702}, *INS*(*Time*)={2, 1, 2, 3};

INS(*Seq_ID*)={1, 2, 3, 4, 5, 6, 7, 8}, *INS*(*SID*)={s01, s01, s02, s02, s02, s03, s04, s04}, *INS*(*Cname*)={*Database*, *Automata*, *Simulation*, *Algorithm*, *Automata*, *Database*, *Simulation*, *Algorithm*};

INS(*Projname*)={Data Integration in Sensor Network, Wireless SN Design, Ontology System for DI, Integration System based on XML, e-Government Roadmap Design}, *INS*(*PID*)={p01, p02, p03, p04, p03}, *INS*(*SID*)={s01, s02, s03, s04, s03};

$RI_{\text{ini}}=\{A_{13}\xrightarrow{RI_{\text{ini}}}A_{21}, A_{42}\xrightarrow{RI_{\text{ini}}}A_{11}, A_{43}\xrightarrow{RI_{\text{ini}}}A_{31}\}$;

$RI_{\text{VC}}=\{A_{53}\xrightarrow{RI_{\text{VC}}}A_{11}\}$, $RI_{\text{QP}}=\{A_{52}\xrightarrow{RI_{\text{QP}}}A_{21}\}$.

Example 10 The domain information of each attribute includes the data type information of the instances (Dom_{DT}) and the semantic description (Dom_{SD}). The semantic description is related to the column list by the analysis of the user queries.

Dom_{DT}(*SID*)={*String*}, Dom_{SD}(*SID*)={*TakingLecture.SID*, *Project.SID*}, Dom_{DT}(*Sname*)={*String*}, Dom_{SD}(*Sname*)={Null}, Dom_{DT}(*PID*)={*String*}, Dom_{SD}(*PID*)={*Professor.PID*, *Project.PID*};

Dom_{DT}(*PID*)={*String*}, Dom_{SD}(*PID*)={*Student.PID*, *Project.PID*}, Dom_{DT}(*Pname*)={*String*}, Dom_{SD}(*Pname*)={Null}, Dom_{DT}(*Office*)={*Integer*}, Dom_{SD}(*Office*)={Null};

Dom_{DT}(*Cname*)={*String*}, Dom_{SD}(*Cname*)={*TakingLecture.Cname*}, Dom_{DT}(*Room*)={*Integer*}, Dom_{SD}(*Room*)={Null}, Dom_{DT}(*Time*)={*Integer*}, Dom_{SD}(*Time*)={Null};

Dom_{DT}(*Seq_ID*)={*Integer*}, Dom_{SD}(*Seq_ID*)={Null}, Dom_{DT}(*SID*)={*String*}, Dom_{SD}(*SID*)={*Student.SID*, *Project.SID*}, Dom_{DT}(*Cname*)={*String*}, Dom_{SD}(*Cname*)={*Class.Cname*};

Dom_{DT}(*Projname*)={*String*}, Dom_{SD}(*Projname*)={Null}, Dom_{DT}(*PID*)={*String*}, Dom_{SD}(*PID*)={*Student.PID*, *Professor.PID*}, Dom_{DT}(*SID*)={*String*}, Dom_{SD}(*SID*)={*TakingLecture.SID*, *Student.SID*}.

Definition 5 The primary key is a subset of $\{A_{i1}, A_{i2}, \ldots, A_{in}\}$ and can be represented as $R_i[PK]=\{A_{i1}\}$. The foreign key is also a subset of $\{A_{j1}, A_{j2}, \ldots, A_{jn}\}$ and can be represented as $R_i[FK]=\{A_{j2}\}$.

Example 11 The representation of PK and FK information about the simple dataset shown in Algorithm 1 is as follows:

Student[*PK*]={*SID*}, *Professor*[*PK*]={*PID*}, *Class*[*PK*]={*Cname*}, *TakingLecture*[*PK*]={*Seq_ID*}, *Project*[*PK*]={*Projname*};

Student[*FK*]={*PID*}, *TakingLecture*[*FK*]={*SID*, *Cname*}, *Project*[*FK*]={*PID*, *SID*}.

We defined the data model and detailed the information such as the domain information, primary key information, and foreign key information. With these information defined by Definitions 2~5, we performed a formal verification about the implicit RI information extraction of the VQT algorithm. Through this formal verification, we can validate that the extracted implicit RI information by the VQT algorithm is exact. Before this validation, we define the RI.

Definition 6 If column A_{22} is the foreign key, A_{11} is the primary key, and there is referential integrity between columns A_{22} and A_{11}, the RI can be

represented as $A_{22} \xrightarrow{RI} A_{11} \Leftrightarrow \forall x, x \subseteq R_2 \Rightarrow \exists y, y \subseteq R_1 \wedge x.A_{22}=y.A_{11})$.

To validate the extracted implicit RI information by the VQT algorithm, we verify the following four points:

(1) If there is the RI between columns A_{22} and A_{11}, the data types in the domain information of columns A_{22} and A_{11} are the same.

(2) If $A_{22} \xrightarrow{RI} A_{11}$, the semantic description in the domain information of column A_{22} includes the column name of A_{11}.

(3) If $A_{22} \xrightarrow{RI} A_{11}$, some instances in the foreign key column of relation (2) are the same as the instance in the primary key column of relation (1).

(4) If column A_{22} refers to A_{11} as the RI, the union of the instances in column A_{22} is a subset of the union of the instances in column A_{11}.

Definition 7 If there is an RI between columns A_{11} (primary key) and A_{22} (foreign key), then the following conditions also exist:

Iff $A_{22} \xrightarrow{RI} A_{11}$, then $Dom_{DT}(A_{22})=Dom_{DT}(A_{11})$;

Iff $A_{22} \xrightarrow{RI} A_{11}$, then $(\exists i)(Dom_{SDi}(A_{22}))=A_{11}$;

Iff $A_{22} \xrightarrow{RI} A_{11}$, then $(\exists i)(INS_i(R_2[FK]))=(\exists j)(INS_j(R_1[PK]))$;

Iff $A_{22} \xrightarrow{RI} A_{11}$, then $\cup INS_i(R_2[FK]) \subset \cup INS_i(R_1[PK])$.

Example 12 Through the VQT algorithm, two implicit RIs are extracted: $RI_{VC}=\{A_{53} \xrightarrow{RI_{VC}} A_{11}\}$ and $RI_{QP}=\{A_{52} \xrightarrow{RI_{QP}} A_{21}\}$, where $A_{52}=\{Project.PID\}$, $A_{21}=\{Professor.PID\}$, $A_{53}=\{Project.SID\}$, $A_{11}=\{Student.SID\}$.

(1) $Dom_{DT}(A_{52})=Dom_{DT}(A_{21})=\{String\}$, $Dom_{DT}(A_{53})=Dom_{DT}(A_{11})=\{String\}$

If $A_{53} \xrightarrow{RI_{VC}} A_{11}$, then $Dom_{DT}(A_{53})=Dom_{DT}(A_{11})$; [True]

If $A_{52} \xrightarrow{RI_{QP}} A_{21}$, then $Dom_{DT}(A_{52})=Dom_{DT}(A_{21})$. [True]

(2) $Dom_{SD}(A_{52})=\{Student.PID, Professor.PID\}$, $(A_{21})=\{Professor.PID\}$, $Dom_{SD}(A_{53})=\{TakingLecture.SID, Student.SID\}$, $(A_{11})=\{Student.SID\}$, $Dom_{SD}(A_{52})=A_{21}$, $Dom_{SD}(A_{53})=A_{11}$

If $A_{53} \xrightarrow{RI_{VC}} A_{11}$, then $Dom_{SD}(A_{53})=A_{11}$; [True]

If $A_{52} \xrightarrow{RI_{QP}} A_{21}$, then $Dom_{SD}(A_{52})=A_{21}$. [True]

(3) $INS_1(A_{53})=INS_1(A_{11})=\{s01\}$, $INS_2(A_{53})=INS_2(A_{11})=\{s02\}$, $INS_3(A_{53})=INS_5(A_{53})=INS_3(A_{11})=\{s03\}$, $INS_4(A_{53})=INS_4(A_{11})=\{s04\}$

If $A_{53} \xrightarrow{RI} A_{11}$, then $(\exists i)(INS_i(R_5[FK]))=(\exists j)(INS_j(R_1[PK]))$. [True]

(4) $INS_1(A_{52})=\{p01\}$, $INS_2(A_{52})=\{p02\}$, $INS_3(A_{52})=\{p03\}$, $INS_4(A_{52})=\{p04\}$, $INS_5(A_{52})=\{p03\}$, $INS_1(A_{21})=\{p01\}$, $INS_2(A_{21})=\{p02\}$, $INS_3(A_{21})=\{p03\}$, $INS_4(A_{21})=\{p04\}$, $INS_1(A_{53})=\{s01\}$, $INS_2(A_{53})=\{s02\}$, $INS_3(A_{53})=\{s03\}$, $INS_4(A_{21})=\{s04\}$, $INS_5(A_{53})=\{s03\}$, $INS_2(A_{11})=\{s01\}$, $INS_2(A_{11})=\{s02\}$, $INS_3(A_{11})=\{s03\}$, $INS_4(A_{11})=\{s04\}$, $INS_5(A_{11})=\{s05\}$, $INS_6(A_{11})=\{s06\}$

If $A_{52} \xrightarrow{RI} A_{21}$, then $(\exists i)(INS_i(R_5[FK]))=(\exists j)(INS_j(R_2[PK]))$. [True]

(5) $\cup INS_i(R_5[FK])=\{s01, s02, s03, s04\}$, $\cup INS_i(R_1[PK])=\{s01, s02, s03, s04, s05, s06\}$

If $A_{53} \xrightarrow{RI} A_{11}$, then $\cup INS_i(R_5[FK]) \subset \cup INS_i(R_1[PK])$. [True]

(6) $\cup INS_i(R_5[FK])=\{p01, p02, p03, p04\}$, $\cup INS_i(R_2[PK])=\{p01, p02, p03, p04\}$

If $A_{52} \xrightarrow{RI} A_{21}$, then $\cup INS_i(R_5[FK]) \subset \cup INS_i(R_2[PK])$. [True]

Because RI_{VC} and RI_{QP} meet all the four conditions described in Definition 7, they represent the exact implicit RI information. The implicit RI information extracted by the VQT algorithm is correct.

Experimental datasets

For the performance evaluation of the VQT algorithm, we use an MS Access Northwind sample database, a TPC-H sample schema v1.2.0 (Bray *et al.*, 2000; Transaction Processing Performance Council, 2006), and an Oracle Business Database sample schema. The MS Access Northwind sample database consists of 9 tables, 87 columns, and 7 foreign key constraints. The TPC-H sample schema v.1.2.0 consists of 8 tables, 59 columns, and 8 foreign key constraints. These two sample databases are simple and small databases for scholarly experiments, but the Oracle Business Database sample schema is a practical database. It consists of 24 tables, 122 columns, and 37 RIs about the goods selling, order, and human resource management. The basic information and limitation of the experimental data are listed in Table 4.

Table 4 Basic information of datasets

Sample data sets	T	C	RI_{EXP}	RI_{IMP}	$RI_{IMP\text{-}NV}$	$RI_{IMP\text{-}QP}$
MS Access Northwind	9	87	7	2	1	1
TPC-H sample schema	8	59	8	2	1	0
Oracle Business sample schema	24	122	37	34	4	1

T: number of tables; C: number of columns; RI_{EXP}: number of explicit RIs; RI_{IMP}: number of implicit RIs; $RI_{IMP\text{-}NV}$: number of implicit RIs that cannot be extracted using the VC module; $RI_{IMP\text{-}QP}$: number of $RI_{IMP\text{-}NV}$ that can be extracted using the QP module

At present, all the RIs in the MS Access Northwind Database and the TPC-H sample schema are explicitly defined. Therefore, we modify some RIs in these experimental data by removing a part of the physical relation for clear and precise experiments.

Translation accuracy

The accuracy of the R-schema to XML schema translation represents how much the RIs are exactly reflected during translation. The translation accuracy formula is given as follows:

$$RI_{TA}=\frac{RI_{REXP}+RI_{IMP\text{-}SYN}+RI_{IMP\text{-}SEM}-RI_{MIS\text{-}IMP}}{RI_{ALL}}\times 100\%,$$

where RI_{TA} is the RI translation accuracy, RI_{ALL} the number of all the RIs, RI_{REXP} the number of extracted explicit RIs, $RI_{IMP\text{-}SYN}$ the number of extracted syntactic implicit RIs, $RI_{IMP\text{-}SEM}$ the number of extracted semantic implicit RIs, and $RI_{MIS\text{-}IMP}$ the number of mis-extracted implicit RIs.

Fig.2 shows the translation accuracy of the three sample datasets. The NeT algorithm does not consider the RI and only considers the structural translation. Therefore, the NeT algorithm cannot extract and reflect the RI information during R-schema to XML schema translation. The CoT algorithm can extract the explicit RI of the sample datasets, but cannot extract the implicit RI information and also cannot guarantee the translation accuracy during R-schema to XML schema translation, because it only considers the explicitly defined RI. The ConvRel algorithm is executed in a similar manner to that of CoT, but it considers only the parent-child relationship. That is, the ConvRel algorithm might not exactly extract the RI. However, the VQT algorithm considers not only the structural part but also the semantic aspect. That is, the VQT algorithm can extract the explicit and implicit RI information both syntactically and semantically. In addition, because the VQT algorithm involves several checking and refinement steps, the VQT algorithm can extract and reflect exact and accurate RI. Therefore, the VQT algorithm yields 10%~40% better translation accuracy than the NeT, CoT, and ConvRel algorithms, and it can yield more exact translation results.

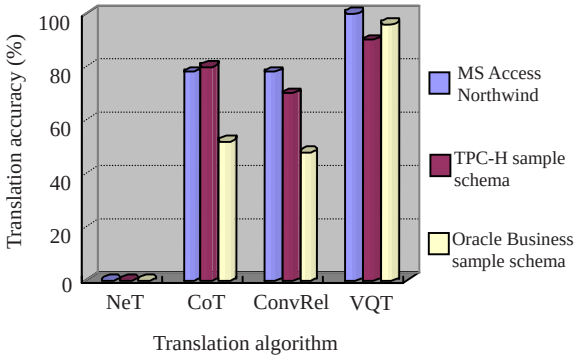

Fig.2 Translation accuracy

RI loss ratio

The RI information loss ratio during R-schema to XML schema translation is shown in Fig.3. The loss ratio represents the number of RIs that are not extracted and lost during the translation process. The formula for the RI loss ratio is shown as follows:

$$RI_{LR}=\frac{RI_{ALL}-(RI_{EXP}+RI_{IMP\text{-}VC}+RI_{IMP\text{-}NV}+RI_{IMP\text{-}QP})}{RI_{ALL}}\times 100\%,$$

where RI_{LR} is the RI loss ratio; RI_{ALL} the number of all the RIs, RI_{EXP} the number of explicit RIs, $RI_{IMP\text{-}VC}$ the number of implicit RIs extracted by the VC

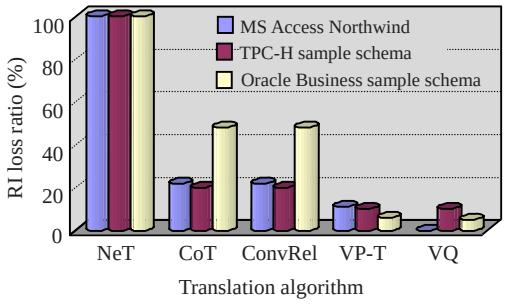

Fig.3 Referential integrity (RI) loss ratio

module, $RI_{\text{IMP-NV}}$ the number of implicit RIs that cannot be extracted by the VC module, and $RI_{\text{IMP-QP}}$ the number of implicit RIs that can be extracted by the QP module.

The RI loss ratio of the NeT algorithm is always 100% because the NeT algorithm does not consider the RI during translation. That is, the NeT algorithm only exactly translates the structural part (such as tables and columns), and the semantic information (such as RI and constraints) is lost during R-schema to XML schema translation. The CoT algorithm can extract and reflect the explicit RI information without any loss, but it loses all of the implicit RI information. The RI loss ratio of the VP-T algorithm is lower than those of the other algorithms because the VP-T algorithm can extract and reflect not only the explicit RI information but also the implicit RI information by analyzing the value pattern. However, because the VP-T algorithm loses the implicit RI (which includes the 1:1 or 1:0 relation), the VP-T algorithm can lose a part of the implicit RI. The VQT algorithm can overcome this limitation of the VP-T algorithm by including the analysis of the user query lists. The VQT algorithm can reflect the implicit RIs that do not include the 1:*N* relation as well as the implicit RIs with a 1:*N* relation. The VQT algorithm presents the lowest loss ratio with regard to the RI among the R-schema to XML schema translation algorithms.

Except for the translation accuracy and the loss ratio, the time complexity of translation is also an important factor in evaluating the translation system. Obviously, the VQT algorithm consumes more time for translation than existing systems because the VQT algorithm includes many steps for considering syntactic/semantic elements in R-schema and refining extracted results for more precise results. However, the major contribution of this paper is to present a method that can convert relational schemas to XML schemas more precisely than the methods proposed before. In addition, the strengths of the method are a more precise translation and support of XML schema that is more structured and capable than XML DTD. Relational schema to XML schema conversion is usually not a very time-consuming task. Precision is more important than speed in most translation cases. Therefore, we do not focus on the time complexity of the VQT algorithm in this study.

CONCLUSION

In this paper, we have defined and proposed the VQT algorithm for the automatic extraction of implicit RI information by analyzing value cardinality and user query patterns. In addition, we execute a formal verification for evaluating the result of the implicit RI extracted by the VC and QP modules. We further describe quantitative comparison experiments among the VQT algorithm and conventional translation algorithms. By using the VQT algorithm, we can obtain more exact XML documents and execute more effective translation as compared to conventional R-schema to XML schema translation algorithms such as the NeT, CoT, ConvRel, and VP-T. Further, we can avoid insertion and deletion errors by using the VQT algorithm.

References

Andersson, M., 1995. Extracting an entity relationship schema from a relational database through reverse engineering. *Int. J. Cooper. Inf. Syst.*, **4**(2-3):259-286. [doi:10.1142/S0218843095000111]

Bray, T., Paoli, J., Cavary, M., 2000. Extensible Markup Language (XML) 1.0 (2nd Ed.). W3C Recommendation.

Buche, P., Barthelemy, J., Haemmerle, O., Hignette, G., 2006. Fuzzy semantic tagging and flexible querying of XML documents extracted from the Web. *J. Intell. Inf. Syst.*, **26**(1):25-40. [doi:10.1007/s10844-006-5449-8]

Carey, M., Jerry, K., Jayavel, S., Eugene, S., Subramanian, S., 2000. XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents. Proc. Very Large Data Bases, Cairo, Egypt, p.646-648.

Chen, S.K., Lo, M.L., Wu, K.L., Yih, J.S., Viehrig, C., 2006. A practical approach to extracting DTD-conforming XML documents from heterogeneous data sources. *Inf. Sci.*, **176**(7):820-844. [doi:10.1016/j.ins.2004.12.009]

Deschrijver, G., 2007. Arithmetic operators in interval-valued fuzzy set theory. *Inf. Sci.*, **177**(14):2906-2924. [doi:10.1016/j.ins.2007.02.003]

Duta, A., Barker, K., Alhajj, R., 2004. ConvRel: Relationship Conversion to XML Nested Structures. Proc. ACM SIG Symp. Applied Computing, Nicosia, Cyprus, p.698-702.

Elmasri, R., Navathe, S., 2003. Fundamental of Database Systems (4th Ed.). Addison-Wesley, United Kingdom.

Fan, W., Simeon, J., 2003. Integrity constraints for XML. *J. Comput. Syst. Sci.*, **66**(1):254-291. [doi:10.1016/S0022-0000(02)00032-6]

Fernandez, M., Tan, W., Suciu, D., 2000. SilkRoute: trading between relations and XML. *Computer Networks*, **33**(1-6):723-745. [doi:10.1016/S1389-1286(00)00061-X]

Hainaut, J., Chandelon, M., Tonneau, C., Joris, M., 1993. Schema transformational techniques for database reverse engineering. *LNCS*, **823**:364-375.

Hainaut, J., Hanrard, J., Roland, D., Engelbert, V., Hick, J., 1996. Structure Elicitation in Database Reverse Engineering. Proc. 3rd Working Conf. on Reverse Engineering, CA, USA, p.131-140.

Hunter, D., Watt, A., Rafter, J., *et al.*, 2004. Beginning XML (3rd Ed.). Wrox Press, USA.

ISO/IEC, 2001. Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML). JTC 1 SC 34 and ISO/IEC 8839:1986.

Kim, J., Jeong, D., Baik, D., 2005. An algorithm for automatic inference of referential integrities during translation from relational database to XML schema. *LNAI*, **3802**:161-170.

Kim, J., Jeong, D., Baik, D., 2007. Query pattern-based relational data to XML data translation algorithm. *J. Comput. Sci.*, **3**(4):212-217.

Lee, D., Mani, M., Chiu, F., Chu, W., 2001. Nesting-based Relational-to-XML Schema Translation. Proc. Int. Workshop on the Web and Databases, Santa Barbara, CA, USA, p.61-66.

Lee, D., Mani, M., Chit, F., Chu, W., 2002. NeT & CoT: Translating Relational Schemas to XML Schemas Using Semantic Constraints. Proc. 11th ACM Int. Conf. on Information and Knowledge Management, McLean, VA, USA, p.282-291.

Lee, D., Mani, M., Chu, W., 2003. Schema Conversion Methods Between XML and Relational Models. Knowledge Transformation for the Semantic Web. IOS Press, the Netherlands.

Madiraju, P., Sunderraman, R., Navathe, S., Wang, H., 2006. Semantic integrity constraint checking for multiple XML databases. *J. Database Manag.*, **17**(4):1-19.

Naughton, J., DeWitt, D., Maier, D., 2001. The Niagara Internet query system. *IEEE Data Eng. Bull.*, **24**(2): 27-33.

Prakash, S., Bhowmick, S., Madria, S., 2006. Efficient recursive XML query processing using relational database systems. *Data Knowl. Eng.*, **58**(3):207-242. [doi:10.1016/j.datak.2005.07.001]

Seligman, L., Rosenthal, A., 2001. XML's impact on databases and data sharing. *IEEE Computer*, **34**(6):59-67.

Transaction Processing Performance Council, 2006. TPC-H Standard Specification, Version 1.1.0.

Turau, V., 2002. A Framework for Automatic Generation of Web-based Data Entry Applications Based on XML. Proc. ACM Symp. on Applied Computing, Madrid, Spain, p.1121-1126.

Widom, J., 1999. Data management for XML: research directions. *IEEE Data Eng. Bull.*, **22**(3):44-52.

Witkowski, A., Bellamkonda, S., Bozkaya, T., Folkert, N., Gupta, A., Haydu, J., Sheng, L., Subramanian, S., 2005. Advanced SQL modeling in RDBMS. *ACM Trans. on Database Syst.*, **30**(1):83-121. [doi:10.1145/1061318.1061321]

Zheng, X.Q., Chen, H.J., Wu, Z.H., Mao, Y.X., 2006. Dynamic query optimization approach for semantic database grid. *J. Comput. Sci. Technol.*, **21**(4):597-608. [doi:10.1007/s11390-006-0597-4]