Transforming Queries from a Relational Schema to an Equivalent Object Schema: A Prototype Based on F-logic[1]

Yahui Chang and Louiqa Raschid and Bonnie Dorr
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742
{yahui, louiqa, bonnie}@umiacs.umd.edu
TEL: (301)405-2715
FAX: (301)405-6707

Keywords: heterogeneous intelligent information systems, knowledge representations using F-logic, query transformation techniques, architecture for interoperable query processing.

Abstract: This paper describes a technique to support interoperable query processing when multiple heterogeneous knowledge servers are accessed. The problem is to support query transformation transparently, so a user can pose queries locally, without any need of global knowledge about different data models and schema. In a companion paper, an architecture for supporting interoperability was described and we provided some details of transforming queries from an object schema to an equivalent relational schema. In this paper, we focus on transforming SQL source queries, posed against a relational schema, to XSQL queries to be evaluated against equivalent object schema.

We describe some functional modules in detail, namely an extractor module (EM) which extracts semantics from a source query, and a heterogeneous mapping module (HTM) which maps among entities in different schema, based on some mapping rules which reflect global knowledge of the models, schema and query languages. The local and global dictionary knowledge is represented in a canonical form, using a high-level logic representation, namely F-logic.

[1]This research has been partially supported by the Defense Advanced Research Project Agency under grant DARPA/ONR grant 92-J1929.

1 Introduction and comparison with related research

One of the most important requirements of a heterogeneous information system is to support interoperable query processing when multiple heterogeneous servers are being accessed. The problem is to support query transformation transparently, so a user can pose queries locally, without needing global knowledge about different data models and schema. After such a transformation, a query may be answered using information from multiple sources. In our research described in this paper and related paper (Raschid *et al* (1993), Raschid *et al* (1994)), we present a technique to achieve interoperability, through capturing knowledge about source and target query languages, data models, schema and mapping information, in local and global dictionaries, and applying aset of mapping rules to obtain a transformation. We represent the dictionary knowledge and mapping rules using a high level logiclanguage, F-logic (see Kifer and Lausen (1989), Kifer *et al* (1990), Lawley (1993)).

This research can be placed in the context of other research in the area of *representational heterogeneity*, where equivalent information is stored in multiple schema using different models, leading to possible mismatch. However, since the knowledge is equivalent, it is assumed that it is always possible to resolve any mismatch in the schema.

Many approaches advocate the use of a global schema (see a summary in Batini *et al* (1986). However, past research in schema conversion was often ad hoc and mapping information to resolve discrepancies was not represented declaratively. Global schema were handcraftedas were the query transformation algorithms. A disadvantage of the global schema approach is that a single global schema needed to be accepted by all users; this may not be feasible in many environments. Furthermore, most research only addresses the issue of accessing local schema via queries posed against the global schema. Little work has been done on interoperable query processing with local queries (legacy applications) that may wish to access information in other servers. Few researchers deal with object and deductive data models; most focus on the relational model.

In contrast, our approach does not require schema integration among the multiple servers. Moreover, our approach addresses the problem of transforming queries transparently, so the local user (or legacy application) does not need explicit knowledge (about the global schema or mapping knowledge among the schema) to pose a query. In companion papers (e.g., Raschid *et al* (1993)), we have described other research approaches (Kent (1991), Krishnamurthy *et al* (1991), Ahmed *et al* (1993), etc,) and compared them with our research.

The research that is closest to our work is described by Lefebvre *et al* (1992) and Qian (1993). Lefebvre *et al* (1992) consider the problem of interoperable query processing, but their approach is limited to relational schema. F-logic is used to express the mapping information among multiple relational schema and to express the algorithm for query transformation. Our work, which includes heterogeneous models, can be seen as an extension of their approach. Qian (1993) suggests that a language which has minimal *representation bias* may express mismatch in representation among heterogeneous schema. The paper describes an approach to interoperable query processing for different schema based on the same data model (entity relationship model), but should be applicable to heterogeneous data models as well.

Our approach encapsulates the information on resolving representational heterogeneity in a knowledge dictionary and uses this to support query mapping and query transformation. Users need not write their queries against a global schema; thus, we provide interoperability in a transparent manner. In addition, the knowledge dictionary represents the mapping and transformation information in a declarative manner; this can also be treated as a knowledge source, to be used for providing explanations, etc. Finally, our approach also extracts relevant semantics from a source query; this allows the transformed queries to reflect the user's requirements more closely and may be more efficient to process. In Raschid *et al* (1993), we studied the problem of transforming queries

posed against an object schema into queries against a relational schema. We considered XSQL queries, an extension to an SQL query allowing *path expressions* that represent a path in an object schema, along a class hierarchy or along reference pointers.[2] These path expressions are translated to equivalent constructs in an SQL query, to reflect the mapping between the object schema and the relational schema. The mapping information is represented in a parameterized structure and is used during the transformation. Applying this mapping information is crucial to obtaining a correct transformation. A direct translation from XSQL to SQL queries that does not use the mapping information will be incorrect since it will not be able to handle the heterogeneity of the object and relational schema.

In this paper, we address the issue of transforming queries posed against a relational schema to produce queries against the object schema. A direct translation of the queries will not handle schema heterogeneity. The relational model does not express (explicitly) the class hierarchy and object reference captured in the object model. During the transformation, wemust determine if a comparison in the WHERE clause of an SQL query corresponds to identifying a specific class in the class hierarchy or an object reference. The mapping knowledge between the relational and object schema is used to make this determination.

We illustrate the tasks involved through an example that demonstrates the transformation of a source SQL query (posed against a relational schema) to a target XSQL query for an equivalent object schema. This paper is organized as follows: section 2 gives examples of transforming from SQL queries to equivalent XSQL queries. Section 3 provides an architecture for interoperable query processing. Section 4 describes the function of the Extractor Module (EM) which extract semantics from a source SQL query and then represents the semantics in a canonical form. Section 5 describes the F-logic implementation which transforms a canonical form for a SQL query to an equivalent XSQL query.

2 Examples of Transforming Queries from Relational to Object Schema

We use a sample relational schema (Figure 1) and two object schemas (Figures 2 and 3). Each of the object schema is equivalent to the relational schema, but they use different class hierarchies to organize the information in the relations.

In the relational schema, relation Register has an attribute Ssn which refers to the primary identifier, social security number, for a student who is currently registered. There are several relations containing information about students. Student, GradStudent and UGStudent have Ssn as the primary key. There are four other relations corresponding to graduate and undergraduate students in two departments; these use Name as the primary key and have an attribute WorksIn which refers to a Project in which the student participates. The relation Project describes these projects and is identified based on Pno.

In the object schema,[3] each node is an object. The dotted line represents a class hierarchy, and a solid arc represents a pointer to another object, e.g., the attribute Sid of an object Register-class points to an object Student-class. In object schema #1, Student-class has two immediate sub-classes Grad-class andUG-class. Each of them has also two sub-classes, identifying the Cs and Math graduate and undergraduate students (4 classes). In contrast, in object schema #2, CsStudent-class and MathStudent-class are the sub-classes of Student-class. Then, each of these classes has two sub-classes, for graduate and undergraduate students, respectively. To compare

[2]The detailed syntax of XSQL queries is described in Kifer *et al* (1992).

[3]Each object in the schema is a *class* object which may be distinguished from the *individual* objects or instances.

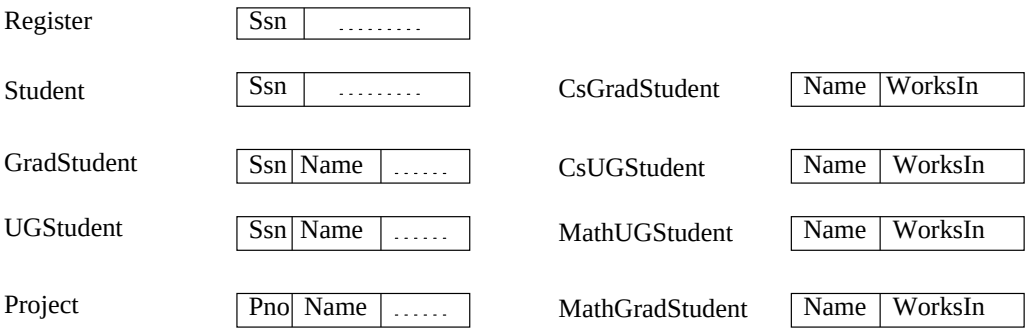

Figure 1: Example relational schema

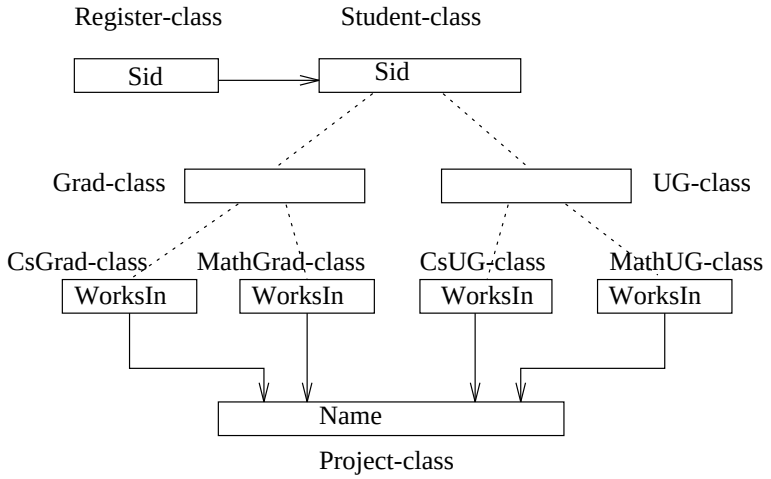

Figure 2: Example object oriented schema #1

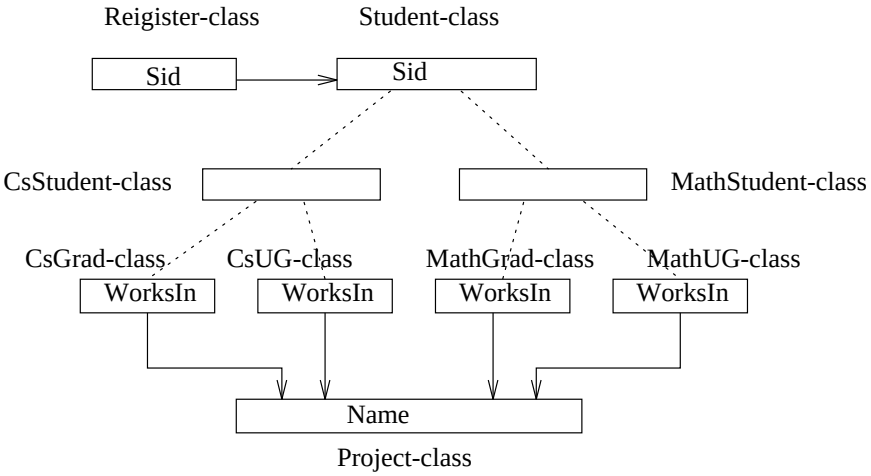

Figure 3: Example object oriented schema #2

the two object schemas, in schema #1, all grad students are in one class Grad-class but in schema #2, they are in two classes, CsGrad-class and MathGrad-class. Similarly, in schema #1, computer science students are in two classes, CsGrad-class and CsUG-class, whereas in schema #2 they are in one class, CsStudent-class.

Queries against the relational schema will be expressed using SQL. To express a query against the object schema, we use a XSQL-like query.[4] We use simple XSQL queries that only differ from SQL queries in the path expression of the where clause. The *path expression* we consider is of the form

$$\text{sel}_0.\text{Att}_1\{[\text{sel}_1]\}.\cdots.\text{Att}_m\{[\text{sel}_m]\}$$

A selector sel_i is a class label and Att_i is an attribute. The attribute Att_{i+1} following each selector sel_i is either directly defined in the class sel_i or it is an inherited attribute for sel_i and must be defined for one of the super-classes of sel_i. Each $sel_i.Att_{i+1}$ can either return a value or a pointer to an object. Braces {} denote optional terms of the path expression. The selector sel_{i+1} following any expression such as $sel_i.Att_{i+1}$ indicates that sel_{i+1} could correspond to the class object that is pointed to, or it could be any sub-class of this class.

Suppose a user wants to identify all the projects in which currently registered computer science graduate students conduct research. This would correspond to the following SQL query against the relational schema:

Query 1 in relational schema

Select Pname
from Register, GradStudent, CsGradStudent, Project
where Register.Ssn = GradStudent.Ssn and
GradStudent.Name = CsGradStudent.Name and
CsGradStudent.WorksIn = Project.Pno

Note that tuples from the relation Register cannot be joined with tuples from the relation CsGradStudent directly, because they use different attributes as keys. In the object schema, these joins must be interpreted as selecting those students in CsGrad-class, and the projects in which they work. CsGrad-class occurs as a sub-class of different classes in object schema #1 and #2. However, the equivalent XSQL query for these object schemas will be the same, since the XSQL query need not specify the different paths from Student-class to CsGrad-class in the two schema due to the fact that CsGrad-Class is a sub-class of Student-class in both cases. The query is as follows:

Query 2 in object schema #1 and #2:

Select Z.Name
from Register-class X, CsGrad-class Y, Project-class Z
where X.Sid[Y].WorksIn[Z]

In the next example, an SQL query corresponds to two different XSQL queries in these two object schema. Suppose we consider a SQL query in the relational schema which determines all currently registered graduate students. The SQL query is as follows:

Query 3 in relational schema

Select Name
from Register, GradStudent
where Register.Ssn = GradStudent.Ssn

In object schema #1, there is a single Grad-class, which is an immediate sub-class of Student-class, representing all graduate students. However, in object schema #2, there is no single object representing graduate students; rather, these are represented by two objects, CsGrad-class and

[4] XSQL is an extension of SQL that considers object schema; the language is described in more detail in Kifer *et al* (1992).

MathGrad-class. As a result, we will get two different XSQL queries. The corresponding XSQL query in object schema #1 is as follows:

Query 4 in object schema #1

Select Y.Name
from Register X, Grad-class Y
where X.Sid[Y]

The corresponding XSQL query in object schema #2 will be as follows: [5]

Query 5 in object schema #2

Select Y.Name | Y′.Name
from Register X, CsGrad-class Y, MathGrad-class Y′
where X.Sid[Y] | X.Sid[Y′]

3 Architecture for Query Transformation

This section describes an architecture for a system that supports interoperable query processing through transparent query transformation. Figure 4 illustrates the overall design of the system. It is assumed that local and global knowledge dictionaries that support the mapping among the heterogeneous schema are represented in a declarative language in some canonical form represented by F-logic. Details concerning the F-logic language (as well as the canonical representation) are discussed in a later section. We describe some functional modules of the **Interoperability Module (IM)**, in detail, and refer to various knowledge dictionaries, as needed.

Corresponding to each DBMS in the heterogeneous environment, there is a local knowledge dictionary that encodes relevant knowledge on (1) the query language constructs,and (2) the data model and the local schema. This local dictionary is accessed by the **Extraction Module (EM)** of the IM. This module accesses both the local query language and schema dictionary for the task of extracting relevant semantic information from the source query. The EM will represent the semantic information, extracted from the query constructs, in a corresponding canonical representation, CanonicalRepresentation (CRobj or CRrel, resp.).

The next module performs the transformation among the heterogeneous schema and is known as the **HeTerogeneous Mapping (HTM).** It has several functions, each of which accesses the global knowledge dictionary. The first function is identifying the mapping knowledge, HTMapping, representing the correspondence among entities in the different schema. In the case of a mapping from an object to a relational schema, this knowledge is classified in a structure HTobjrelMapping. For a mapping from a relational to an object schema, this knowledge is classified as HTrel-objMapping.

The second function is identifying and applying the correct mapping rule, which determines the correct transformation. To determine this, the HTM accesses information in the corresponding CR structure and the relevant HTMapping information in the global dictionary. After applying the rules, HTM produces a TransfQuery (TQrel or TQobj, resp.) to represent the target query in the canonical representation. The mapping rules are classified into different groups, each of which performs a class of transformations relevant to the target model/query language. When mapping from a relational schema to an object schema, the groups of rules which we use are as follows: **Ident-Class-Hier** (identifying class hierarchies), **Object-Ptr** (identifying pointers to objects), and **Exp-Join** (producing explicit join pair). Selected rules in each group will be applied, and each produces a part of the TQobj.

Currently, the resulting TQobj structure we obtain is simple. This is because we only consider a subset of SQL query and also because the HTrel-objMapping and the mapping rules resolve

[5]The symbol "|" functions as the operator "OR" in SQL.

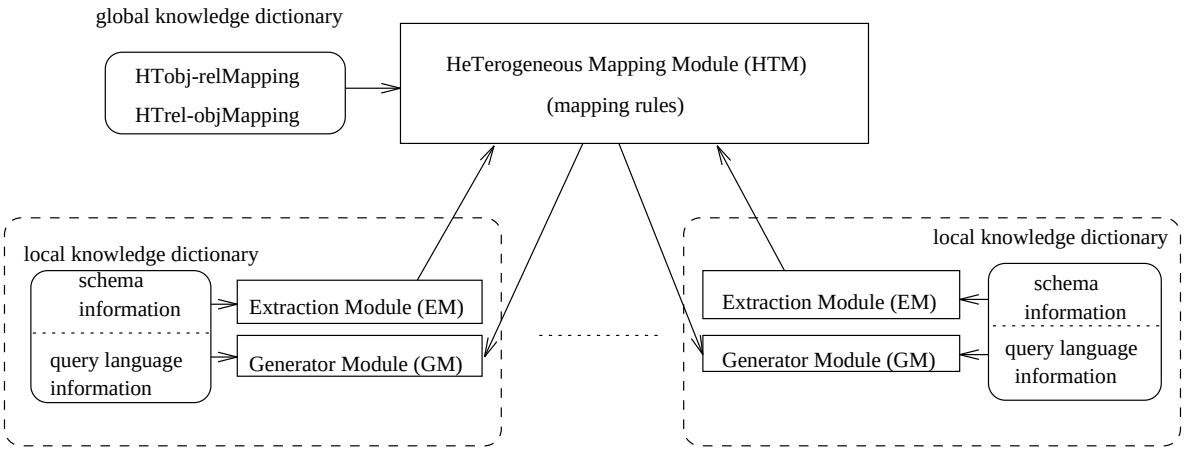

Figure 4: Architecture of the Interoperability Module (IM)

fairly simple conflicts when going from a relational schema to an object schema. As we consider more complex source queries and resolve more complex conflicts, we expect to extend the HTrel-objMapping structure, CRrel structure, TQobj structures, and the classes of mapping rules, to reflect this added complexity and knowledge. We note that if a unified schema were available, then HTM could actually provide interoperability with respect to this unified schema.

The final module of the IM is the **Generator Module (GM)** . This module accesses the local knowledge dictionary, corresponding to the target schema. It may use knowledge on the cost of processing a query, and knowledge of equivalences in the target query language and schema, to produce an optimal query in the target query language.

4 Algorithm for Extraction Module

We will present the functions and the algorithm for the **Extraction Module** (EM) in this section. The EM operates on a source relational query and produces a corresponding canonical structure CRrel. There are several reasons for adopting such a structure instead of direct translation of a source query to the target query language form. First, different query languages (object or relational) have different constructs, e.g., path expression in XSQL, functional composition in OSQL, etc. In addition, we have the heterogeneity of the different schema themselves. Using a canonical structure allows us to separate the mapping knowledge due to the heterogeneity of schema from the knowledge needed to translate between the constructs in the different languages. If we use a direct translation approach, the rules will be very complicated because they have to resolve the schema conflicts and also consider the syntax of each language. By extracting the semantics of the query and representing them in a canonical form, the semantics is considered independent of the query languages; thus, we simplify the mapping rules in the HTM. This is precisely analogous to arguments for the use of an interlingual representation over a direct-mapping approach in the field of natural language translation (see, e.g., Dorr (1993)).

A second reason for adopting a canonical struccture is that the user might provide a poor query (e.g., redundant joining pairs in the WHERE part of a SQL query) because of lack of sufficient knowledge about the local schema. The EM has access to the local schema dictionary and may

be able to modify the query to provide an optimal form for query transformation. This might be especially useful when the mapping information is incomplete. By using a canonical form, we can identify the exact mapping information we need.

We use a data structure, CRrel, to represent the semantics of an SQL query. The SELECT clause and the FROM clause of an SQL query, which are used to describe the desired relations and attributes, will be represented directly in the CRrel. Aggregate functions, like MIN, MAX, AVG, etc., will also be expressed in CRrel using similar constructs. We will omit the discussion about these. The WHERE clause of an SQL query is used to express constraints and probably involves complicated operations. We assume the WHERE clauses only consist of conjunctions, with the form: cond_1 AND cond_2 AND $\cdots$ cond_n. Each conjunct cond_i could be viewed as a that is represented by comparing the attribute of relations with some other values, set-operation involving nested queries, etc. The disjunction is removed by transforming the WHERE part into a disjunctive normal form, splitting the query by disjuncts, creating corresponding CRrel for each disjunct, and making a union of the result of each sub-query. Nesting queries will be simplified and represented by joining relation-attribute pairs if applicable; otherwise, they will be represented by other CRrel structures.

The CRrel structure is represented as a F-logic data structure. Details of F-logic syntax are given in the next section.

CRrel[sub-rel $\leftrightarrow$ (R_{sup}, R_{sub});
reference $\leftrightarrow$ REF;
comparison $\leftrightarrow$ (REL-OPR, OP1, OP2);
set-operation $\leftrightarrow$ (SET-OPR, OP1, OP2)]

The values of the attribute *sub-rel* is a set of pairs with the form (R_{sup},R_{sub}), which means the key of the relation R_{sub} is a subset of the key of the relation R_{sup}. The value of the attribute *reference*, REF, is a set of ordered lists of quadruples. Each quadruple has the form (R1,A1,R2,A2), with the attribute A1 of R1 having *referential* relationship with the attribute A2 of R2. This could correspond to the joining pair R1.A1 = R2.A2 in SQL when there is a foreign key relationship between R1.A1 and R2.A2. The value of the attribute *comparison* is a set of triples with the form (REL-OPR, OP1, OP2), where REL-OPR represents a relational operator, which could be in the set: $\{ \leq, \geq, <, >, \neq \}$. OP1 is the first operand and OP2 is the second operand. These could be relation-attribute pairs or each could be an arithmetic expression. This corresponds to a comparison of the values of OP1 and OP2, and does not have special structural meanings. The last attribute *set-operation* corresponds to the operation involving sets. SET-OPR could be IN, or NOT IN, etc., which are used to test if the value of OP1 is IN or NOT IN the set represented by OP2. OP2 is a nested query and will be represented by another CRrel structure.

We now describe the portion of the algorithm for the EM which operates on a set of joining relation-attribute pairs. The aim is to consider each joining pair in the WHERE clause, and use schema information such as keys, key inclusion dependency, foreign keys, etc., to simplify the query, eliminate redundant pairs, and obtain relevant structure information. Basically, the first step is to represent the set of joining pairs using graphs, if they convey structural information. The next step uses key inclusion dependency to identify the minimal subsets of key attributes in the query. (This corresponds to the most specific subclasses.) We will build an *inclusive class* named by each minimal subset with all its superset constituents as the elements of the class. The third step is using a foreign key relationship to construct an ordered list which corresponds to referencing a sequence of objects.[6]

[6]We use the construct "cons" in F-logic to represent an order list, which looks like:

Algorithm for Extraction Module:

- Input : a set of joining relation-attribute pairs
- Output : CRrel in F-logic form

1. Build a graph where each node corresponds to a relation, and each edge is either an *inc-link* or a *ref-link* when applicable. The *inc-links* represent a key inclusion dependency, and the *ref-links* represent a foreign key relationship.
   For each input pair (R1.A1 = R2.A2), create new nodes for relations R1 or R2 if they have not been created previously. Then check if the inclusion dependency exists and create corresponding *inc-links* and *ref-links*:
   (a) If A1 and A2 are both keys, check if there exists a key inclusion dependency between them. If so, create an inc-link pointing from the node corresponding to the subset relation to the superset relation. For example, if (R1.A1) $\subseteq$ (R2.A2), the link is pointing from node R1 to node R2.
   (b) If (R1.A1) is a foreign key of R2 which has the key A2, create a ref-link pointing from the node R1 to the node R2. Annotate the link with the quadruple (R1, A1, R2, A2) to denote the joining relation-attribute pairs. Also test if R2.A2 is a foreign key of relation A1.
   (c) If the pairs do not have specific structural meanings, output the F-logic form:
   $$\text{CRrel[comparison} \leftrightarrow (\text{“=”,R1.A1,R2.A2)]}$$
2. Check inc-links and build *inclusive classes*.
   Let each node which does not have any incoming inc-links in the graph be the root, say R1. It represents the most specific subset in that hierarchy with respect to inclusion dependency. Traverse through the inc-links and include each visited node, say R_j, in the inclusive class denoted by R_i. Note that R_j may be in several inclusive classes if it has more than one incoming inc-links. The corresponding output structure is:
   $$\text{CRrel[sub-rel} \leftrightarrow (R_j, R_i)].$$
   If there is a cycle through the inc-links, all those corresponding relations $(R1, \cdots Rn)$ will have the same set of values for their keys. It means they actually correspond to a vertically partitioned relation where the attributes of a tuple are distributed in several relations. We will randomly pick one relation R_i as the name of the inclusive class and create the following constructs:
   CRrel[sub-rel $\leftrightarrow$ (R1, R_i)], $\cdots$, CRrel[sub-rel $\leftrightarrow$ (R_i, R_i)], $\cdots$, CRrel[sub-rel $\leftrightarrow$ (Rn, R_i)].
   For those nodes R_i which do not have incoming and outgoing inc-links, we will produce the following structure: CRrel[sub-rel $\leftrightarrow$ (R_i, R_i)].
3. Check the ref-links and produce ordered lists.
   (a) This step is to modify the graph by removing the inc-links and “collapse” the nodes which are in the same inclusive classes.
   For each node, say R, whose inclusive class, say C, is not equal to R, do the following: for each outgoing ref-link n pointing from node R to a certain node R', add a new ref-link pointing from C to the inclusive class C' of R', and delete n;
   for each incoming ref-link n pointing to R from another node R', add a new ref-link pointing to C from the inclusive class of R', and delete n.

cons((R1,A1,R2,A2),cons((R3,A3,R4,A4),$\cdots$)).

(b) This step is to navigate through the ref-links and produce an ordered list which corresponds to a sequence of referential relationships.
Let each node which does not have incoming ref-links be a root. Traverse through the links from the root to each leaf to create an ordered list of joining pairs. Suppose there is a link pointing to the node N, with annotation (R1,A1,R2,A2), and a link pointing from the node N, with annotation (R3,A3,R4,A4), then output:
CRrel[reference $\leftrightarrow$ cons(cons(L,(R1,A1,R2,A2)),(R3,A3,R4,A4))], where L is the output from the previous traversal. Initially it is given the value nil.[7]

5 Representation for Mappings

This section discusses the mapping rules in the **Heterogeneous Mapping Module**, which utilizes the HTrel-objMapping information from the global knowledge dictionary. These rules are represented as a F-logic program. We will give a brief description of the syntax of F-logic in section 5.1, explain the data structures used by the mapping rules in section 5.2, and present the mapping rules in section 5.3. Examples of rule application will also be given.

5.1 F-logic

We borrow the brief description from Lefebvre *et al.* (1992) to introduce the syntax of F-logic. In F-logic, the *instance term* $o : c$ means that the object o is an instance of class c. A *data term* $o[m@a_1, \cdots, a_n \rightarrow v; m'@a_1, \cdots, a_p \leftrightarrow \{v', v''\}]$ means that the value of the functional method m with arguments a_1 to a_n for the object o is a set containing the values v' and v''. If a method m has no arguments, "@" will be omitted. The symbol $\rightarrow$ indicates a single-valued method, and the symbol $\leftrightarrow$ indicates a set-valued method.[8] Note that other values could also be members of this set, and that the expression above does *not* restrict the value of m' for o to be $\{v', v''\}$; it only indicates *some of* the values of the corresponding set. An object can be denoted by a constant, or a term. For example, $dept(cs)$ is a valid object identifier. Notational conventions allow us to write $o[m' \leftrightarrow v]$ instead of $o[m' \leftrightarrow \{v\}]$ for a single element of a set-valued method; the expression $o[m \rightarrow v; n \rightarrow v']$ is equivalent to $o[m \rightarrow v] \wedge o[n \rightarrow v']$.

A F-logic program consists of a set of data declarations(data or instance terms), and a set of *deduction rules*. A deduction rule has a *head*, which is a data term, and a *body*, which is a conjunction of data and instance terms. Disjunction and negation are allowed in the body of rules.

5.2 Data Structures

There are three kinds of data structures used by the mapping rules described in the section 5.3. CRrel produced by the Extractor Module represents the semantics extracted from an SQL query. HTrel-objMapping captures the mapping information from a relational schema to an equivalent object schema. Applying the mapping rules and HTrel-objMapping,we produce TQobj which represents the transformed query in the object schema.

CRrel[sub-rel $\leftrightarrow$ (R_{sup}, R_{sub});
reference $\leftrightarrow$ J;
comparison $\leftrightarrow$ C]

[7] For notational simplicity, we will replace the expression cons(nil,(R1,A1,R2,A2)) by just (R1,A1,R2,A2).
[8] This symbol is different from the one used in the original paper.

HTrel-objMapping(R, D)[key $\leftrightarrow$ K;
mapped-class $\leftrightarrow$ C[super-class $\leftrightarrow$ C'];
mappings $\leftrightarrow$ map(R,A,D)[object-attr-domain $\leftrightarrow$ (O,AN,OO)]]

TQobj(CRrel, D)[sub-class@R $\rightarrow$ O;
path@J $\rightarrow$ L;
join@J $\rightarrow$ K]

We have introduced the CRrel data structure in section 4. The second data structure, HTrel-objMapping, provides the mapping information from the relation R to an equivalent class C in the target schema D, and a relation attribute pair to an equivalent object attribute pair. The class hierarchy in D is represented by the method *super-class*. Its value C' represents the super classes of class C. We also store keys for a relation from the source schema. The last data structure TQobj represents the transformed query for the object schema. Its method *sub-class* will return the corresponding class label for the inclusive class R. The method *path* will produce an XSQL path expression, given an ordered list of joining pairs. The method *join* is invoked to produce an explicit join in XSQL format.

5.3 Mapping Rules

The mapping rules can be classified into three groups. The first group **Ident-Class-Hier** deals with *inclusive classes*. The second group **Object-Ptr** determines the appropriate path in the object schema corresponding to a given ordered list of joining pairs in the relational schema. The last group **Exp-Join** is used when an explicit join is needed. We show how these rules transform the joining pairs given in Query One in section 2 to an equivalent XSQL query for the object schema in Figure 2. We will call that schema d_2.

- **Group One: Ident-Class-Hier**

  The following rule determines the class name in the object schema corresponding to the inclusive class associated with a relation. We must check if the class hierarchy in the target object schema corresponds exactly to the inclusive classes in the relational schema. If so, we produce an optimal query for the relevant (most specific) class; otherwise, we generate an explicit join (see the rules is Group Three: Exp-Join).

  TQobj(CRrel,D)[sub-class@R1 $\rightarrow$ O2] $\leftarrow$
  CRrel[sub-rel $\leftrightarrow$ (R1,R2)] $\wedge$
  HTrel-objMapping(R1,D)[mapped-class $\rightarrow$ O1] $\wedge$
  HTrel-objMapping(R2,D)[mapped-class $\rightarrow$ O2 [super-class $\leftrightarrow$ O1]]

  We give an example to show how to apply this rule. Suppose the relation GradStudent is in the inclusive class denoted by CsGradStudent and the corresponding CRrel is:

  CRrel[sub-rel $\leftrightarrow$ (GradStudent, CsGradStudent)]

  Also, CsGradStudent maps to the class CsGrad-class, and CsGrad-class is a subclass of Grad-class. Thus, the inclusive class hierarchy of the relation schema corresponds exactly to the subclass hierarchy in the object schema d_2. The instantiation of the mapping rules will be as follows:

TQobj(CRrel,d_2)[sub-class@GradStudent → CsGrad-class] ←
CRrel[sub-rel ↔ (GradStudent, CsGradStudent)] ∧
HTrel-objMapping(GradStudent,d_2)[mapped-class ↔ Grad-class] ∧
HTrel-objMapping(CsGradStudent,d_2)[mapped-class ↔ CsGrad-class
[super-class ↔ Grad-Class]]

If R2 is the inclusive class for R1, R1 and R2 map to O1 and O2, respectively, but O2 is not a subclass of O1, we will produce an explicit join pair. This situation happens when the two schemas have a different grouping of data. One database may contain a class Student with two subclasses, GradStudent and UnderGradStudent, and the other database may have relations CsStudent and MathStudent without an inclusion dependency.

- **Group Two: Object-Ptr**

The rules in this group produce an XSQL-like path expression. The parameter of the method *path* of TQobj is an ordered list of quadruples. A quadruple (R1, A1, R2, A2) corresponds to a joining pair R1.A1 = R2.A2, where the ref-link is pointing from R1 to R2. If R1.A1 maps to O1.AN1, R2.A2 maps to O2.AN2, and the *inclusive class* of R1 maps to the subclass O3 of O1, and the *inclusive class* of R2 maps to the subclass O4 of O2, then the result of the method *path* will be the quadruple (O3,AN1,O2,O4). AN1 is an attribute of O3, which could be explicit or inherited attribute. O2 is the object referenced by the attribute AN1. O4 is a specific subclass of O2 and the XSQL query returns this subclass. The corresponding XSQL query is O3.AN1[O4]. Note that this rule uses the previous rule in group Ident-Class-Hier to get the value for the method sub-class of TQobj.

TQobj(CRrel,D)[path@(R1,A1,R2,A2) → (O3,AN1,O2,O4)] ←
CRrel[reference ↔ (R1,A1,R2,A2)] ∧
HTrel-objMapping(R1,D)[
mappings ↔ map(R1,A1,D) [object-attr-domain ↔ (O1,AN1,O11)]] ∧
HTrel-objMapping(R2,D)[key ↔ A2;
mappings ↔ map(R2,A2,D) [object-attr-domain ↔ (O2,AN2,O22)]] ∧
O1 ≠ O2 ∧
((O11 = O2) ∨ HTrel-objMapping(R2,D) [mapped-class ↔ O2[super-class ↔ O11]]) ∧
TQobj(CRrel,D)[sub-class@R1 → O3] ∧
TQobj(CRrel,D)[sub-class@R2 → O4]

We need to check that O1 is not equal to O2 and O11 is equal to O2 or a superclass of O2. The condition that O1 is not equal to O2 ensures that a pointer from O1 to O2 is needed in the object schema. The second condition ensures that the object referenced by AN1 of O1 is O2 or a superclass of O2. If one of these tests fails, we will produce an explicit join.

Consider the joining pair Register.Ssn = GradStudent.Ssn. Observe that the relations Register and the relation GradStudent correspond to the classes Register-class and Grad-class respectively, so the instantiation of the mapping rules will be as follows:

TQobj(CRrel,d_2)[path@(Register,Ssn,GradStudent,Ssn) →
(Register-class,Sid,Student-class,CsGrad-class)] ←

CRrel[reference $\leftrightarrow$ (Register,Ssn,GradStudent,Ssn)] $\wedge$
HTrel-objMapping(Register,d_2)[
mappings $\leftrightarrow$ map(Register,Ssn,d_2)
[object-attr-domain $\leftrightarrow$ (Register-class,Sid,Student-class)]]$\wedge$
HTrel-objMapping(GradStudent,d_2)[key $\leftrightarrow$ Ssn;
mappings $\leftrightarrow$ map(GradStudent,Ssn,d_2)
[object-attr-domain $\leftrightarrow$ (Grad-class,Sid,integer)]] $\wedge$
Register-class $\neq$ Grad-class $\wedge$
HTrel-objMapping(GradStudent,D) [
mapped-class $\leftrightarrow$ Grad-class[super-class $\leftrightarrow$ Student-class]] $\wedge$
TQobj(CRrel,d_2)[sub-class@Register $\rightarrow$ Register-class] $\wedge$
TQobj(CRrel,d_2)[sub-class@GradStudent $\rightarrow$ CsGrad-class]

The other joining pair CsGradStudent.WorksIn = Project.Pno will produce the following instantiation given the fact that the relation Project maps to the class Project-class.

TQobj(CRrel,d_2)[path@(CsGradStudent,WorksIn,Project,Pno) $\rightarrow$
(CsGrad-class,WorksIn,Project-class,Project-class)] $\leftarrow$
CRrel[reference $\leftrightarrow$ (CsGradStudent,WorksIn,Project,Pno)] $\wedge$
HTrel-objMapping(CsGradStudent,d_2)[
mappings $\leftrightarrow$ map(CsGradStudent,WorksIn,d_2) [object-attr-domain $\leftrightarrow$
(CsGrad-class,WorksIn,Project-class)]]$\wedge$
HTrel-objMapping(Project,d_2)[key $\leftrightarrow$ Pno;
mappings $\leftrightarrow$ map(Project,Pno,d_2) [object-attr-domain $\leftrightarrow$
(Project-class,nil,oid)]] $\wedge$
CsGrad-class $\neq$ Project-class $\wedge$
TQobj(CRrel,d_2)[sub-class@CsGradStudent $\rightarrow$ CsGrad-class] $\wedge$
TQobj(CRrel,d_2)[sub-class@Project $\rightarrow$ Project-class]

Note that the value of the attribute object-attr-domain of the object map(Project,Pno,d_2) is the triple (Project-class,nil,oid). This means the attribute Pno of the relation Project maps to the object identifier of the class Project-class.

The following rule is used to process a set of joining pairs and produce an ordered list following the pointers. This corresponds to the *path expression* in XSQL.

TQobj(CRrel,D)[path@cons(L,(R1,A1,R2,A2)) $\rightarrow$ cons(P,(O1,AN1,O11,O2))] $\leftarrow$
TQobj(CRrel,D)[path@(R1,A1,R2,A2) $\leftrightarrow$ (O1,AN1,O11,O2)] $\wedge$
TQobj(CRrel,D)[path@L $\leftrightarrow$ P]

This rule will combine the two paths we just produced as follows:

TQobj(CRrel,d_2)[path@cons((Register,Ssn,GradStudent,Ssn),
(CsGradStudent,WorksIn,Project,Pno)) $\rightarrow$
cons((Register-class,Sid,Student-class,CsGrad-class),

(CsGrad-class,WorksIn,Project-class,Project-class))] ←
TQobj(CRrel,d_2)[path@(Register,Ssn,GradStudent,Ssn) ↔
(Register-class,Sid,Student-class,CsGrad-class)] ∧
TQobj(CRrel,d_2)[path@(CsGradStudent,WorksIn,Project,Pno)↔
(CsGrad-class,WorksIn,Project-class,Project-class)]

- **Group Three: Exp-Join**

The following rules are used when an explicit join is needed. The first rule corresponds to the situation where the joining pairs do not have structural information in the relational schema.

TQobj(CRrel,D)[join@(R1,A1,R2,A2) → (O1,AN1,O2,AN2)] ←
CRrel[comparison ↔ ("=",R1.A1,R2.A2)] ∧
HTrel-objMapping(R1,D)[
mappings ↔ map(R1,A1,D) [object-attr-domain ⇒ (O1,AN1,O11)]] ∧
HTrel-objMapping(R1,D)[
mappings ↔ map(R2,A2,D) [object-attr-domain ⇒ (O2,AN2,O11)]]

The second rule is invoked when the object schema does not have the exact class hierarchy corresponding to the *inclusive class* hierarchy in the relational schema.

TQobj(CRrel,D)[join@(R1,A1,R2,A2) → (O1,AN1,O2,AN2)] ←
CRrel[sub-rel ↔ (R1,R2)] ∧
HTrel-objMapping(R1,D)[mapped-class → O1] ∧
not(HTrel-objMapping(R2,D)[mapped-class → O2 [super-class ↔ O1]]) ∧
HTrel-objMapping(R1,D)[key ↔ A1;
mappings ↔ map(R1,A1,D) [object-attr-domain ↔ (O1,AN1,O11)]] ∧
HTrel-objMapping(R2,D)[key ↔ A2;
mappings ↔ map(R2,A2,D) [object-attr-domain ↔ (O2,AN2,O11)]]

The last rule deals with the situation where the foreign key relationship in the relational schema does not correspond to the reference pointer in the object schema.

TQobj(CRrel,D)[join@(R1,A1,R2,A2) → (O1,AN1,O2,AN2)] ←
CRrel[reference ↔ (R1,A1,R2,A2)] ∧
HTrel-objMapping(R1,D)[
mappings ↔ map(R1,A1,D) [object-attr-domain ↔ (O1,AN1,O11)]] ∧
HTrel-objMapping(R2,D)[key ↔ A2;
mappings ↔ map(R2,A2,D) [object-attr-domain ↔ (O2,AN2,O22)]] ∧
O1 = O2 ∨
(O11 ≠ O2 ∧
not(HTrel-objMapping(R2,D)[mapped-class ↔ O2[super-class ↔ O11]])) ∧
TQobj(CRrel,D)[sub-class@R1 → O3] ∧
TQobj(CRrel,D)[sub-class@R2 → O4]

6 Summary

We have proposed a technique for interoperable query processing, where a user can pose a query against a local schema, without any knowledge of the other models/schema, such as mappings among schema and conflicts. We used the task of extracting knowledge from a SQL query, and mapping from a relational to two object schema with equivalent knowledge, to produce a transformed XSQL-like query. We use a high-level declarative language, F-logic, to represent the local and g lobal knowledge dictionaries and the mapping knowledge (rules). We are currently investigating how to extend the data structure CRrel so that i t can represent all possible SQL queries, and extend the HTrel-objMapping and HTobj-relMapping structures to include other kinds of schema conflicts. Other future work includes creating equivalent queries using available integrity constraints. This is especially useful when the mapping information is incomplete. The verification of the HTmapping information will be also part of our future research.

7 Bibliography

Ahmed, R., J. Albert, W. Du, W. Kent (1993) "An overview of Pegasus," *Proceedings of the International Conference on Data Engineering.*

Albert, J., R. Ahmed, M. Ketabchi, W. Kent, M. Shan (1993) "Automatic importation of relational schemas in Pegasus," *Proceedings of the International Conference on Data Engineering.*

Arens, Y. and C. Knoblock (1992) "Planning and reformulating queries for semantically-modeled multidatabase systems," *Proceedings of the International Conference on Knowledge Management.*

Batini, C., Lenzerini, M. and Navathe, S.B. (December 1986) "A comparative analysis of methodologies for database schema integration," *ACM Computing Surveys* 18:4, 323-364.

Dorr, Bonnie J. (1993) *Machine Translation: A View from the Lexicon*, MIT Press, Cambridge, MA.

Gardarin, G. and P. Valduriez (1992) "ESQL2: An object-oriented SQL with F-logic semantics," *Proceedings of the International Conference on Data Engineering.*

Jeusfeld, M. and M. Jarke (1991) "From relational to object-oriented integrity simplification," *Proceedings of the Second International Conference on Deductive and Object-Oriented Databases.*

Kent, W. (1991) "Solving domain mismatch and schema mismatch problems with an object-oriented database programming language," *Proceedings of the International Conference on Very Large Data Bases.*

Kifer, M. and G. Lausen (1989) "F-logic: A higher-order language for reasoning about objects, inheritance and scheme," *Proceedings of the ACM Sigmod Conference.*

Kifer, M., G. Lausen, and J. Wu (1990) "Logical foundations of object-oriented and frame-based languages," SUNY at Stonybrook, Technical report 90/14.

Kifer, M., W. Kim, and Y. Sagiv (1992) "Querying object-oriented databases," *Proc. of the ACM Sigmod Conference.*

Kim, W. and J. Seo (December, 1991) "Classifying schematic and data heterogeneity in multidatabase systems," *IEEE Computer*, 12–18.

Krishnamurthy, R., W. Litwin, and W. Kent (1991) "Language features for interoperability of databases with schematic discrepancies," *Proceedings of the ACM Sigmod Conference.*

Lawley, M. (1993) "A Prolog interpreter for F-logic," Griffith University, Technical report.

Lefebvre, A., P. Bernus and R. Topor (1992) "Query transformation for accessing heterogeneous databases," *Joint International Conference and Symposium on Logic Programming, Workshop on Deductive Databases.*

Qian, X. (1993) "Semantic interoperation via intelligent mediation," *Workshop on Research Issues in Data Engineering.*

Raschid, L., Chang, Y. and B. Dorr (1993) "Interoperable Query Processing with Multiple Heterogeneous Knowledge Servers," *Proceedings of the Second International Conference on Information and Knowledge Management.*

Raschid, L., Chang, Y. and B. Dorr (1994) "Query Transformation Techniques for Interoperable Query Processing in Cooperative Information Systems," *Proceedings of the Second International Conference on Cooperative Information Systems.*