Information Capacity Preserving Translations of Relational Schemas Using Structural Transformation*

November 1995
External Technical Report
ISSN 0836-0227-95-392

Patrick Martin, James R. Cordy and Rateb Abu-Hamdeh
Dept. of Computing and Information Science
Queen's University at Kingston
Kingston, Ontario
CANADA K7L 3N6

Abstract

Schema translation is one step in the process of schema integration in a multidatabase system. We describe an approach to the problem of schema translation which is based on structural transformation, or syntactic rewrite rules. There are two main advantages with our approach. First, the translation process can be directly automated using the transformation language TXL. Second, the correctness of the translation system can be proved using a straightforward analysis based on information capacity. We develop a set of translation rules to convert relational schemas to entity-relationship model schemas. We show that the translation rules are information capacity preserving and outline how they are automated with TXL.
Topic Area: Interoperability of Heterogeneous Information Systems

1 Introduction

Schema translation is the process of transforming a schema in one data model into a corresponding schema in a different data model. Historically, the main use of schema translation has been in the *view integration* phase of *conceptual schema design*, which takes several user views representing the information requirements of different users and integrates them into a single conceptual schema [5]. The user views may be expressed in different data models, in which case they must be translated to a common data model before performing integration.

Recent interest in the problem of dealing with legacy information systems has renewed interest in schema translation as it applies to the problems of database integration in multidatabase, or

*The authors thank IBM Canada Ltd., NSERC and ITRC for their support of the research.

federated database, systems [14], database migration [11], and schema evolution [16]. As part of the CORDS Multidatabase System (MDBS) project [2], our main use of schema translation is in *database integration* which takes the schemas from a collection of heterogeneous component database systems, and provides an integrated view of the available data.

The principles for defining a schema in a particular data model may be formal, in the sense that they are inherent to the model, or they may be part of standard design practice. In either case, the principle used is represented by a distinctive structure in the schema. The structure is in turn represented by a particular pattern in the DDL definition of the schema.

Structural transformation recognizes structures in a source object (schema, program, etc.) and transforms them into other structures in a target language to produce a translation of the original object. One can naturally apply it to the problem of schema translation by searching for the structures in a schema that represent the design rules followed in building the schema and then transforming them to corresponding structures in order to construct the translated schema. A structure in the source schema for which there is no corresponding structure in the target data model means there is a potential loss of information.

Structural transformations of schemas can be implemented as source-to-source syntactic transformations provided that, for each data model,

1. there exists a concrete data description language for expressing schemas in the model and
2. the syntactic structure of the data description language representation of a schema reflects the logical structure of the schema in some way.

When both these constraints hold, we can implement the structural transformation as a syntactic transformation from the data description language of the original data model to the data description language of the target data model. Our approach does exactly this, using the TXL source transformer [9] to implement the syntactic transformations.

In this paper we discuss the development of a set of structural transformation rules, which we collectively call a *translation system*, for translating relational schemas to Entity-Relationship schemas. Such a system may be used to extract the conceptual schema from an exisiting relational schema or as an intermediate step in the translation to other data models such as an object-oriented model [1].

The remainder of the paper is organized as follows. Section 2 presents a brief review of related work. Section 3 defines the source and target data models. Section 4 describes the rules in the translation system and section 5 presents a proof of the information capacity preserving properties of the system. Section 6 briefly discusses an implementation of the translation system using TXL and Section 7 presents our conclusions.

2 Related Work

Approaches have been developed to translate between a variety of data models. Most of the work related to conceptual schema design has focussed on translation of schemas in high-level conceptual models, most commonly the entity-relationship (ER) model, into schemas in the three traditional data models, namely the hierarchical, network, and relational models [12, 15, 22]. Dumpala and Arora [12] also present reverse mappings to translate relational, hierarchical, and network schemas to the ER model which can be used to extract the conceptual structure of existing schemas. Johannesson [16], Castellanos and Saltor [7], Markowitz and Makowsky [18], and Davis and Arora [10] all describe approaches to extracting the logical structure from relational schemas. Work has also been reported on mapping schemas between the traditional data models. The two approaches used have been either to map schemas directly between data models, for example Zaniolo [23], or to map to an intermediate representation. Biller [6], for example, uses a semantic data model as an intermediate form when mapping relational schemas to network schemas.

One problem with most of these approaches, except the one described by Davis and Arora [10], is that they require knowledge of the semantics of the source schema, and thus they are intended to be performed manually by a database designer. In practice, however, source schemas can be large and complex, which makes the translation process tedious, time consuming, and prone to error. Another problem with most of the proposed translation schemes is that, with the exception of Johannesson [16] and Markowitz and Makowsky [18], they are all informal and do not show that the schemas produced have the same information capacity as the original schemas.

The approach discussed in this paper overcomes both shortcomings. The translation systems produced can be automated and translate existing schemas without added information. Our approach does, however, accommodate extended schemas. It also lends itself to a formal analysis of

the relative information capacities of the source and target schemas.

3 The Data Models

A *data model* is a tool that provides an interpretation for data describing real-world situations. It consists of a set of constructs to describe the structure and constraints of the data, or *data definition language* (DDL), a set of operations to access the data, or *data manipulation language* (DML), and, at least informally, a set of principles for arranging the structures to represent the situation. The description of a particular database in the DDL is called the *database schema.*

Definition 1 *A* **database schema** *D of data model $\mathcal{M}$ is a pair $< S, C >$ where S is set of distinct structure descriptions, and C is a set of distinct constraint descriptions, and members of S and C are expressed in the DDL of $\mathcal{M}$.*

3.1 Source Data Model: Relational Model

We present the basic relational concepts and terminology used in the paper. Further details can be found in any text book, for example Atzeni and De Antonellis [3].

Definition 2 *A* **relation scheme** *is a relation name R together with a set of distinct attribute names, $X = \{A_1, A_2, ..., A_n\}$ where each A_i belongs to some valid domain D_i. To indicate the relation scheme we write $R(A_1, A_2, ..., A_n)$ or $R(X)$.*
A **relation instance** *of the relation scheme R is a finite set of* tuples *$r = \{t_1, t_2, ..., t_m\}$ where each tuple is an ordered list of values $t =< v_1, v_2, ..., v_n >$ and $v_i \in D_i$, $1 <= i <= n$, corresponds to the attribute A_i.*

Definition 3 *An* **integrity constraint** *γ on a relation scheme R is a function that associates each relation instance r of R with a boolean value $\gamma(r)$. We say r satisfies γ if $\gamma(r) = true$ and r violates γ if $\gamma(r) = false$.*

We consider the following integrity constraints in our translation:

1. A *unique constraint*, specified by a UNIQUE clause in the SQL table definition, identifies one or more candidate keys in addition to the primary key of a relation. Each attribute appearing in a UNIQUE clause must also be declared NOT NULL.

For a relation scheme $R(X)$, relation instance r of $R(X)$ and set of attributes $U \subseteq X$

$$unique(r, U) = true\ if\ \forall t_1, t_2 \in r, t_1 \neq t_2 \Rightarrow t_1[U] \neq t_2[U]$$

2. A *primary key constraint*, represented by the PRIMARY KEY clause in the SQL table definition, is a special case of the unique constraint. A candidate key is designated as the primary key of the relation. Each attribute appearing in a PRIMARY KEY clause must also be declared NOT NULL.

   For a relation scheme $R(X)$, relation instance r of $R(X)$ and set of attributes $PK \subseteq X$ called *prime* attributes,

   $$primaryKey(r, PK) = true\ if\ \forall t_1, t_2 \in r, t_1 \neq t_2 \Rightarrow t_1[PK] \neq t_2[PK]$$

   We also define the *nonprime* attributes of $R(X)$ to be the attributes in the set $X - PK$.

3. A *not null constraint*, indicated by the NOT NULL clause in the SQL table definition, states that a particular attribute may not take a null value.

   For a relation scheme $R(X)$, relation instance r of $R(X)$ and attribute $A \in X$

   $$notNull(r, A) = true\ if\ \forall t \in r, t[A] \neq null$$

4. A *referential constraint*, specified by a FOREIGN KEY clause in the SQL table definition, states that for all tuples in the referencing relation the specified attributes (that is the FOREIGN KEY attributes) either have a null value or contain a value found in the corresponding attribute in the referenced relation.

   For relation schemes $R_1(X_1)$ and $R_2(X_2)$, relation instances r_1 and r_2 of $R_1(X_1)$ and $R_2(X_2)$, respectively, and set of attributes $fk_2 \subseteq X_1 \wedge X_2$

   $$references(r_1, r_2, fk_2) = true\ if\ \pi_{fk_2}(r_1) \supseteq \pi_{fk_2}(r_2)$$

where $\pi_{fk_i}(r_i)$ is the *projection* of r_i on fk_i which is $\{t[fk_i]|t \in r_i, i \in \{1,2\}\}$.

For relation scheme $R(X)$ we define $FK = fk_1 \cup fk_2 \cup ... \cup fk_n$ to be the set of all foreign key attributes where n is the number of distinct foreign keys in R.

Definition 4 *A* **database schema** *D for the relational model is a pair $< R, C >$ where R is set of relation schemes with distinct names, and C is a set of integrity constraints.*
A **database instance** *(or simply* **database***) on a database schema D with $R = \{R_1(X_1), R_2(X_2), ..., R_n(X_n)\}$ is a set of relation instances $r = \{r_1, r_2, ...r_n\}$ where each instance r_i is defined on the corresponding scheme R_i and satisfies the corresponding integrity constraints in C.*

An example source relational schema is given in Appendix A.

3.2 Target Data Model: Entity-Relationship Model

The target data model for our translation is the Entity-Relationship model [8]. As explained earlier, we chose the ER model because it can serve as the basis for further work on extracting conceptual schemas or as an intermediate stage in translations to other data models.

The ER model does not have a generally accepted DDL. We chose to represent the ER schema information as a set of Prolog facts which we call a *schema factbase*. A *fact* represents a single piece of information and is defined as follows.

Definition 5 *A* **schema fact** *$p(A_1, A_2, ..., A_n)$ is composed of a predicate symbol p and a list of arguments $A_1, A_2, ..., A_n$ where $n >= 1$. The predicates correspond to constructs in the ER model and the arguments are either names from the source schema or constants.*

Definition 6 *A* **schema factbase** *F for the ER model is a triple $< E, R, C >$ where E, R and C are the sets of facts describing entity types, relationship types and integrity constraints, respectively.*

Definition 7 *A* **database** *on a schema factbase F is a collection of database facts which are triples $< s, l, t >$ where s and t are values and l is a label describing the connection between them. The database forms a semantic network [13] where the values are nodes and the connections are directed labelled edges.*

For example, consider a simple schema factbase and its ER diagram representation which are shown in Figures 1 and 2 respectively. Entity types, such as "EMPLOYEE", generate an "entity"

entity(EMPLOYEE)
entityAttribute(EMPLOYEE, SIN, INTEGER)
entityAttribute(EMPLOYEE, Fname, CHARACTER(20))
entityAttribute(EMPLOYEE, Lname, CHARACTER(20))
entityAttribute(EMPLOYEE, Address, CHARACTER(40))
key(EMPLOYEE, SIN)

relationship(WORKS_FOR, BINARY)
entityInRelationship(WORKS_FOR, EMPLOYEE, (1,1), employee)
entityInRelationship(WORKS_FOR, DEPARTMENT, (0,n), department)

entity (DEPARTMENT)
entityAttribute (DEPARTMENT, Number, INTEGER)
entityAttribute (DEPARTMENT, Name, CHARACTER (40))
multivaluedAttribute (DEPARTMENT, Locations, CHARACTER (40))
key (DEPARTMENT, Number)
key (DEPARTMENT, Name)

relationship (MANAGES, Binary)
entityInRelationship (MANAGES, DEPARTMENT, (1, 1), department)
entityInRelationship (MANAGES, EMPLOYEE, (0, 1), employee)

entity(PROJECT)
entityAttribute(PROJECT, Name, CHARACTER(40))
key(PROJECT, Name)

relationship (WORKS_ON, Binary)
relationshipAttribute (WORKS_ON, Hours, INTEGER)
entityInRelationship (WORKS_ON, EMPLOYEE, (0, n), employee)
entityInRelationship (WORKS_ON, PROJECT, (0, n), project)

Figure 1: Example Factbase

Figure 2: Example ER Schema

fact, an "entityAttribute" fact for each attribute which identifies the name and type of the attribute, and a "key" fact for each key of the entity type. We assume that, in the case of more than one key fact, the first fact represents the primary key.

Relationship types, such as "WORKS_ON", generates a relationship fact which contains the arity of the relationship, a "relationshipAttribute" fact for each attribute of the relationship, and an "entityInRelationship" fact for each entity type participating in the relationship which identifies the minimum and maximum cardinalities and the role of the entity type in the relationship. The participation (that is, total or partial) of the entity type in the relationship may be inferred from the minimum cardinality. Other types of facts, which are shown in the example, may be generated for other ER constructs such as weak entities and composite attributes. Examples of these types of facts appear in Appendix B.

A portion of a possible database on the schema factbase given in Figure 1 is shown in Figure 3. The nodes may be actual attribute values, unique identifiers for instances of entity types, relationship types and system types such as ROLES, and type names. The edges represent connections taken from the schema, for example attribute names or system-defined connections such as "instance" and "plays".

4 The Translation System

We express a translation system from relational schemas to ER schemas as the set of transformation rules

$$TS_{RER} = \{tr_p, tr_{wp}, tr_s, tr_{ws}\}$$

where the rules effectively partition the relations of a relational database schema D into four categories such that exactly one rule applies to each relation scheme of D. The derived classification is similar to one described by Batini, Ceri and Navathe [4].

1. tr_p - **Primary Relation Translation Rule**:

   A relation scheme $R(X)$ of D is a *primary relation scheme* if there are no foreign keys or, otherwise, $\forall fk_i \subseteq FK$ where fk_i is the primary key of some relation scheme R_i of D, $fk_i \not\subseteq PK$, that is the primary key does not contain the key of any other relation.

Figure 3: Example Database

The effects on the factbase of applying tr_p to relation scheme R are as follows:

- Add the entity fact R.
- For each attribute $A \in X - (PK \cup FK)$, add the attribute fact A.
- For primary key constraint $primaryKey(r, PK)$ where r is an instance of R and $PK \subseteq X$, add the composite attribute fact A_{PK} containing each attribute $A \in PK$ and the key constraint fact A_{PK}[1].
- For each unique constraint $unique(r, U)$ where r is an instance of R and $U \subseteq X$, add the composite attribute fact A_U containing each attribute $A \in U$ and the key constraint fact A_U.
- For each references constraint $references(r, r_i, fk_i)$ where r is an instance of R, r_i is an instance of another relation scheme R_i in D and $fk_i \subseteq FK$ such that fk_i is primary key of R_i, that is $fk_i = PK_i$, add the binary relationship fact $R : R_i$ and the entity-in-relationship facts R in $R : R_i$ and R_i in $R : R_i$[2]. The cardinality constraints are, by default $(0, 1)$ and $(0, n)$ for R and R_i, respectively. If $notNull(r, fk_i)$ is true then the cardinality constraints are $(1, 1)$ and $(0, n)$ for R and R_i, respectively, and if $unique(r, fk_i)$ is also true then the cardinality constraint for R_i is $(0, 1)$.

2. tr_{wp} - **Weak Primary Relation Translation Rule**:

   A relation scheme $R(X)$ is a *weak primary relation scheme* if $FK \subset PK$ and $\exists A \in X - PK$, that is the primary key contains the keys of one or more other relations and the relation scheme contains nonprime attributes. We use the second condition to differentiate between weak entities and multivalued attributes.

   The effects on the factbase of applying tr_{wp} to relation scheme R are:

   - add the weak entity fact R;
   - for each attribute $A \in X - PK$, add the attribute fact A;
   - for primary key constraint $primaryKey(r, PK)$ where r is an instance of R and $PK \subseteq X$, add the composite attribute fact A_{PPK} to match the partial key of the weak entity where

[1] We assume the first key constraint in an entity definition is always the primary key.
[2] We assume the first entity-in-relationship fact always represents the relation that contained the relationship.

A_{PPK} contains each attribute $A \in PK - FK$ and add the partial key constraint fact A_{PPK};

- add the identifying relationship fact $R : R_1 : R_2 : ... : R_n$ where each entity R_i, $1 \leq i \leq n$ has primary key fk_i such that $fk_i \in PK$ and add the entity-in-relationship fact R in $R : R_1 : R_2 : ... : R_n$.

3. tr_s - **Secondary Relation Translation Rule**:

   A relation scheme $R(X)$ is a *secondary relation scheme* if $FK = PK$, that is the primary key is formed by a concatenation of primary keys from other relation schemes. Such a relation scheme describes a relationship among those other relation schemes.

   The effects on the factbase of applying tr_s to relation scheme R are:

   - add the relationship R;
   - for each relation R_i such that its primary key is fk_i where $fk_i \subseteq PK$, add the entity-in-relationship fact R_i in R;
   - for each attribute $A \in X - PK$, add the relationship attribute fact A.

4. tr_{ws} - **Weak Secondary Relation Translation Rule**:

   A relation scheme $R(X)$ is a *weak secondary relation scheme* if PK contains the primary key fk_i of exactly one other relation scheme $R_i(X)$ and $X = PK$, that is the $R(X)$ corresponds to a multivalued attribute of entity R_i.

   The effects on the factbase of applying tr_{ws} to relation scheme R are:

   - if $|X - PK| > 1$ add multivalued attribute fact M for entity R_i and add multivalued attribute component fact for each attribute $A \in X - PK$, otherwise add multivalued attribute fact A for entity R_i.

5 Properties of the Translation System

An important consideration in developing a practical translation system from one data model to another is the potential for loss of information. Are there patterns in the source schemas that cannot be adequately represented in the target schema? We may evaluate the correctness of a

translation system based on the relative *information capacities* of the source and target schemas, that is, do the source and target schemas model the same real world information?

An analysis of the correctness of a translation system should also consider the operational goals of the schema translation. For example, if a multidatabase system must support querying of the source database through the target schema, then a correct translation system should ensure that all data in the source database maps to a representation under the target schema. If, however, the multidatabase system must also support updates of the source database through the target schema, a correct translation system must ensure that data maps to representations in both directions.

A strength of our approach is that the correctness of translation systems expressed as a set of syntactic rewrite rules may be evaluated in a rigorous manner. We present an analysis of the Relational-to-ER system TS_{RER} as an example. We first give a set of definitions to establish our notion of correctness based on information capacity. We next provide a collection of lemmas which, in combination, prove the main result of the section, namely that the translation system TS_{RER} is information capacity preserving and hence correct. The proofs of the lemmas are given in Appendix C.

The *information capacity* of a database schema D_1 determines the set of valid instances $I(D_1)$ of that schema [21]. Intuitively, we say that a database schema D_2 has a greater information capacity than a schema D_1 if every instance $i \in I(D_1)$ can be mapped to some instance $j \in I(D_2)$ without loss of information, that is we can recover the original instance from its image under the mapping. We define these concepts more precisely below.

Definition 8 *An* **information capacity preserving** *mapping between the instances of two database schemas D_1 and D_2 is a total, injective function $f : I(D_1) \rightarrow I(D_2)$.*

Definition 9 *If D_1 and D_2 are two database schemas such that $f : I(D_1) \rightarrow I(D_2)$ is an information capacity preserving mapping, then D_2* **dominates** *D_1 via f, denoted $D_1 \preceq D_2$.*

Definition 10 *Let $\boldsymbol{D_1}$ and $\boldsymbol{D_2}$ be sets of schemas specified in the different data models $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. A translation system $\mathcal{TS}$ from $\boldsymbol{D_1}$ to $\boldsymbol{D_2}$ induces the total function $F : \boldsymbol{D_1} \rightarrow \boldsymbol{D_2}$. $\mathcal{TS}$ is information capacity preserving if for all $D \in \boldsymbol{D_1}$, $D \preceq F(D)$.*

Definition 11 *A translation system $\mathcal{TS}$ is* **correct** *if it is information capacity preserving and if the operational goal of the system is that the entire database stored under the source schema may be*

viewed and queried through the target schema.

In our approach to schema translation, a translation system is expressed as a set of independent syntactic rewrite rules rather than as an algorithm as in most other approaches. The rules are functions in the mathematical sense, that is they do not produce side-effects, and they produce the same result independent of the order in which they are applied [17]. Thus the proof that a translation system is information capacity preserving can be reduced to proving that each individual rule is an information capacity preserving translation, and that the set of rules in the translation system partition the relation schemes in the source schema so that each relation scheme is translated by exactly one transformation rule. We assume that good design practices are followed in creating the source relational database schemas and do not consider poorly designed relation schemes in our analysis.

Lemma 1 *Given relational database schema* $D =< R, C >$*, the classification imposed on* R *by the translation system* TS_{RER} *is a partition.*

Lemma 2 *Given relational database schema* $D =< R, C >$*, the syntactic rewrite rule* tr_p *is an information capacity preserving translation on its partition of* R*.*

Lemma 3 *Given relational database schema* $D =< R, C >$*, the syntactic rewrite rule* tr_{wp} *is an information capacity preserving translation on its partition of* R*.*

Lemma 4 *Given relational database schema* $D =< R, C >$*, the syntactic rewrite rule* tr_s *is an information capacity preserving translation on its partition of* R*.*

Lemma 5 *Given relational database schema* $D =< R, C >$*, the syntactic rewrite rule* tr_{ws} *is an information capacity preserving translation on its partition of* R*.*

Claim 1 *The Relational-to-ER translation system* TS_{RER} *is an information capacity preserving schema translation.*

PROOF: The proof of the claim that the Relational-to-ER translation system TS_{RER} is an information capacity preserving schema translation based on the results shown in Lemmas 1 - 5. Lemma 1 ensures that exactly one rule in TS_{RER} applies to every relation scheme R_i in a relational database schema D. Lemmas 2 - 5 show that each rule in TS_{RER} is information capacity preserving over its

subset of the relation schemes in D. Therefore the total translation system is information capacity preserving.

6 Implementation of the Translation System

Our translation system has been implemented using the TXL transformation system [9]. TXL is a structural source transformer, which means that

1. both the input and output of the transformation must be in source text form, and
2. the meaning of the source text must somehow be encoded in its syntactic structure.

In practice these two restrictions simply mean that the input and output must be in some formal (machine processable) language.

TXL is well suited to transformations of the structure of the input rather than just its text. For example, the transformation between a cascade of nested if statements and an equivalent case statement is straightforward in TXL, whereas it can be very complex using text-based tools such as sed and awk. TXL achieves this simplicity of structural transformation by parsing the input and output into abstract syntax trees, and implementing the transformations on the trees rather than the source text itself (Figure 4).

TXL transformations are programmed using sets of rules, each of which specifies a *pattern* and a *replacement*. The pattern captures the components of the input structure we would like to transform, and the replacement shows how to rearrange those components into the output. Both are specified by example, which is to say, in source text form, but are implemented as manipulations of the corresponding syntax trees as shown in Figure 4.

The TXL implementation of our translation system represents relational schemas by the source text of their SQL DDL specifications, and entity-relationship schemas by the source text of a Prolog factbase for their ER diagram. The translation system itself is expressed as a set of TXL rules that transform the syntactic structures of the input DDL to the syntactic structures of the output DDL.

Each of the four main rules in the set has a pattern that syntactically matches the structure of one of the four kinds of table definitions corresponding to primary relations, weak primary relations, secondary relations or weak secondary relations as described in section 4, and a replacement that syntactically generates the corresponding Prolog facts (Figure 5).

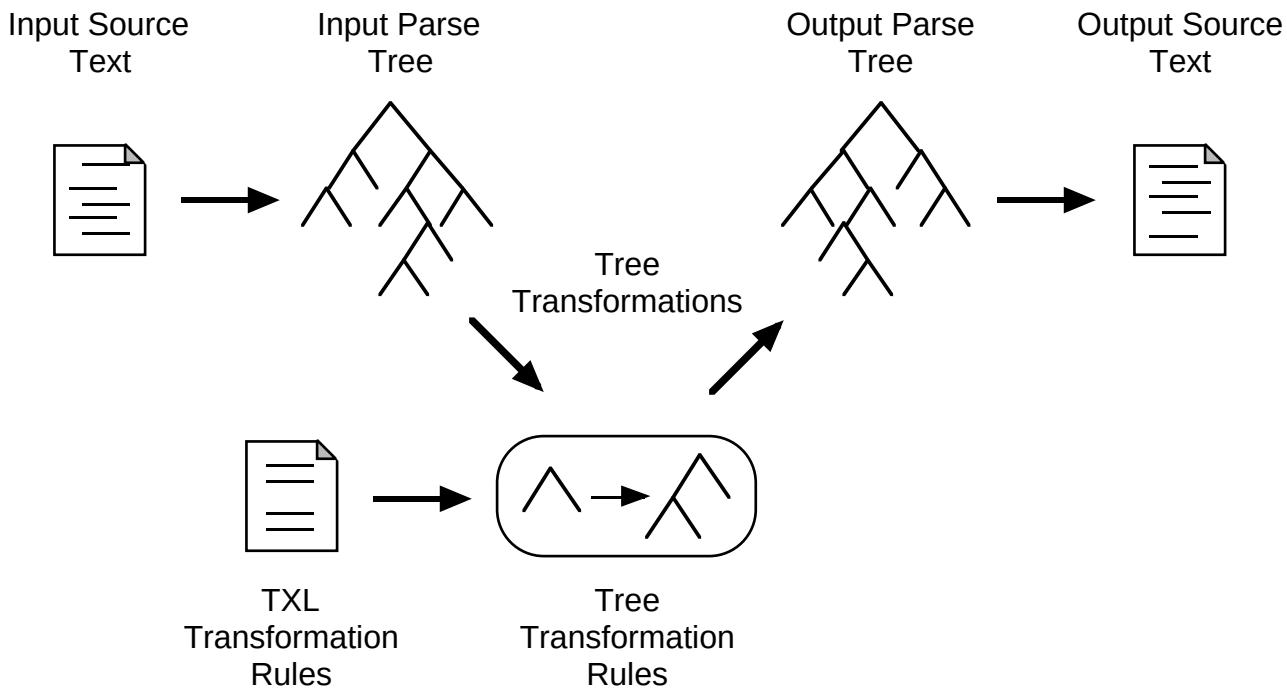

Figure 4: The TXL Transformation Process. Both source text of the input to be transformed and the transformation rules are parsed into tree form. The rules are then successively applied to the parsed input tree to create the tree form of the desired output. Finally, the transformed output tree is deparsed to form the final output text.

TXL's ability to directly encode the structural patterns on which our translations are based makes the implementation of new translation systems, once they are designed, relatively straightforward.

7 Summary

Schema translation is important for providing integration and interoperability in multidatabase systems. Two shortcomings of many of the existing approaches to the problem are that they are not easily automated and that they lack a formal basis for evaluating the correctness of the resulting translations. We have discussed an approach based on structural transformation that overcomes both of these problems.

Structural transformation is a technique for recognizing structures in a source language and translating them into structures in a target language. The structural transformations are implemented as source-to-source syntactic transformations. As we show in the paper, the technique can be successfully applied to the problem of schema translation.

Our approach expresses a translation system as a set of independent transformation rules. We indicate in the paper how one can prove the correctness of a system by showing that the rules preserve information capacity. We also describe how a translation system can be automated using the structural transformation language TXL.

We have developed translation systems to transform schema descriptions in representative relational, hierarchical, network, and object-oriented DDLs into an ER factbase representation, and vice versa. We can compose the translation systems to translate among the various data models.

We plan to continue to study the problem of schema translation and to apply our approach to automatic query translation. We are also incorporating the TXL implementations of the translation systems into a suite of tools to support database integration in a prototype multidatabase system [20].

References

[1] R. Abu-hamdeh, J. Cordy, and P. Martin. Schema translation using structural transformation. In *Proc. of CASCON'94, IBM Centre for Advanced Studies 1994 Conference*, pages 202–215, Toronto, November 1994.

[2] G. Attaluri, D. Bradshaw, N. Coburn, P.-Å. Larson, P. Martin, A. Silbershatz, J. Slonim, and Q. Zhu. The CORDS multidatabase project. *IBM Systems Journal*, 34(1):39–62, January 1995.

```
rule ProcessPrimaryRelations
      % If none of the foreign keys is contained in the primary key,
      % then the table is a primary relation

      replace [tableDefinition_or_predicateSequence]
            CREATE TABLE EntityName [id]
                  ( AttributeDefinitions [list attributeDefinition] )
            UniqueStatements [repeat uniqueConstraint]
            PRIMARY KEY ( PrimaryKey [attributelist] )
            ForeignKeyStatements [repeat foreignKeyDefinition]

      % Get all the foreign keys
      construct ForeignKeys [repeat attributelist]
            _ [^ ForeignKeyStatements]

      % Check the primary key does not contain any foreign key
      where not
            PrimaryKey [contains each ForeignKeys]

      % Get the foreign key attribute set
      construct ForeignKeyAttributes [repeat attribute]
            _ [^ ForeignKeys]

      % Get the primary key attribute set
      construct PrimaryKeyAttributes [repeat attribute]
            _ [^ PrimaryKey]

      % Compute (foreign key attribute set) - (primary key attribute set)
      construct ReducedForeignKeyAttributes [repeat attribute]
            ForeignKeyAttributes [removeAttribute each PrimaryKeyAttributes]

      % Compute (defined attribute set) - (reduced foreign key attribute set)
      construct ReducedAttributeDefinitions [repeat attributeDefinition]
            _ [. each AttributeDefinitions]
              [removeAttributeDefinition each ReducedForeignKeyAttributes]

      % Now construct the output predicates
      construct EntityPredicate [repeat predicate]
            entity ( EntityName )

      construct EntityAttributePredicates [repeat predicate]
            _ [addEntityAttributePredicate EntityName
                  each ReducedAttributeDefinitions]

      construct PrimaryKeyPredicates [repeat predicate]
            _ [addPrimaryKeyPredicates_Atomic EntityName PrimaryKey]
              [addPrimaryKeyPredicates_Composite EntityName PrimaryKey]

      construct UniquePredicates [repeat predicate]
            _ [addUniquePredicate_Atomic EntityName ForeignKeyAttributes
                  each UniqueStatements]
              [addUniquePredicate_Composite EntityName ForeignKeyAttributes
                  each UniqueStatements]

      construct BinaryRelationPredicates [repeat predicate]
            _ [addBinaryRelationPredicates EntityName AttributeDefinitions
                  UniqueStatements each ForeignKeyStatements]
      by
            EntityPredicate
                  [. EntityAttributePredicates]
                  [. PrimaryKeyPredicates]
                  [. UniquePredicates]
                  [. BinaryRelationPredicates]
end rule
```

Figure 5: TXL Encoding of the Main Rule for Translating Primary Relations. The pattern of the TXL rule (the part following the keyword 'replace') recognizes the syntactic form of a table corresponding to a primary relation. The next few clauses (those beginning with the keywords 'deconstruct' and 'where') check additional non-syntactic constraints on primary relations. Subsequent clauses (those beginning with the keyword 'construct') syntactically build the corresponding Prolog facts using other TXL rules not shown here. Finally, the replacement of the rule (the part following the keyword 'by') assembles these various facts into the final ER output for the table.

[3] P. Atzeni and V. De Antonellis. *Relational Database Theory.* The Benjamin/Cummings Publishing Company, Inc., 1993.

[4] C. Batini, S. Ceri, and S. Navathe. *Conceptual Database Design: An Entity-Relationship Approach.* The Benjamin/Cummings Publishing Company, Inc., 1992.

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

[6] H. Biller. On the equivalence of data base schemas – a semantic approach to data translation. *Inf. Syst.*, 4:35–47, 1979.

[7] M. Castellanos and F. Saltor. Semantic enrichment of database schemas: An object oriented approach. In *Proc. of First International Workshop on Interoperability in Multidatabase Systems*, pages 71–78, 1991.

[8] P.P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Trans. Database Syst.*, 1(1), 1970.

[9] J.R. Cordy, C.D. Halpern-Hamu, and E.M. Promislow. TXL: A rapid prototyping system for programming language dialects. *Computer Languages*, 16(1):97–107, 1991.

[10] K. Davis and A. Arora. Converting a relational database model into an entity-relationship model. In S. T. March, editor, *Entity-Relationship Approach*, pages 271–285, Amsterdam, 1988. North-Holland.

[11] P. Drew. On database technology for information system migration and evolution. In *Proc. of the Workshop on Interoperability of Database Systems and Database Applications*, pages 121–131, Fribourg, 1993.

[12] S. Dumpala and S. Arora. Schema translation using the entity-relationship approach. In P. P. Chen, editor, *Entity-Relationship Approach to Information Modeling and Analysis*, pages 337–356, Amsterdam, 1983. North-Holland.

[13] R. Frost. *Introduction to Knowledge Base Systems.* Macmillan Publishing Co., 1986.

[14] D. Hsiao. Tutorial on federated databases and systems (part I). *The VLDB Journal*, 1(1):127–179, 1992.

[15] J. Iossiphidis. A translator to convert the DDL of ERM to the DDL of system 2000. In P. P. Chen, editor, *Entity-Relationship Approach to Systems Analysis and Design*, pages 477–504, Amsterdam, 1980. North-Holland.

[16] P. Johannesson. A method for transforming relational schemas into conceptual schemas. In *Proc. of the International Conference on Data Engineering*, pages 190–201, Houston, 1994.

[17] A. Malton. The denotational semantics of a functional tree-manipulation language. *Computer Languages*, 19(3):157–168, 1993.

[18] V. Markowitz and J. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Trans. on Software Engineering*, 16(8):777–790, 1990.

[19] P. Martin, J. Cordy, and R. Abu-hamdeh. A general approach to schema translation using structural transformation. in preparation.

[20] P. Martin and W. Powley. Database integration using multidatabase views. In *Proc. of CASCON'93, IBM Centre for Advanced Studies 1993 Conference*, pages 779–788, Toronto, 1993.

[21] R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *Proc. of the International Conference on Very Large Data Bases*, pages 120–133, Dublin, Ireland, 1993.

[22] H. Sakai. A unified approach to the logical design of a hierarchical model. In P. P. Chen, editor, *Entity-Relationship Approach to Systems Analysis and Design*, pages 61–74, Amsterdam, 1980. North-Holland.

[23] C. Zaniolo. Design of relational views over network schemas. *Proc. ACM SIGMOD*, pages 179–190, 1979.

A Sample Source Relational Schema

```
CREATE TABLE CONSULTANT (
                Fname CHARACTER (20) NOT NULL,
                Lname CHARACTER (20) NOT NULL,
                Address CHARACTER (40))
PRIMARY KEY (Fname, Lname)

CREATE TABLE EMPLOYEE (
                SIN INTEGER NOT NULL,
                Fname CHARACTER (20),
                Lname CHARACTER (20),
                Address CHARACTER (40),
                Department_Number INTEGER NOT NULL)
PRIMARY KEY (SIN)
FOREIGN KEY (Department_Number)
                REFERENCES DEPARTMENT

CREATE TABLE DEPARTMENT (
                Number INTEGER NOT NULL,
                Name CHARACTER (40) NOT NULL,
                Manager_SIN INTEGER NOT NULL)
UNIQUE (Name)
UNIQUE (Manager_SIN)
PRIMARY KEY (Number)
FOREIGN KEY (Manager_SIN)
                REFERENCES EMPLOYEE

CREATE TABLE PROJECT (
                Name CHARACTER (40) NOT NULL,
                Controlling_dept_Number INTEGER NOT NULL)
PRIMARY KEY (Name)
FOREIGN KEY (Controlling_dept_Number)
                REFERENCES DEPARTMENT

CREATE TABLE SUPPLIER (
                Name CHARACTER (40) NOT NULL)
```

```
PRIMARY KEY (Name)

CREATE TABLE PART (
                Number INTEGER NOT NULL)
PRIMARY KEY (Number)

CREATE TABLE DEPENDENT (
                Relationship CHARACTER (40),
                Name CHARACTER (40) NOT NULL,
                Employee_SIN INTEGER NOT NULL)
PRIMARY KEY (Name, Employee_SIN)
FOREIGN KEY (Employee_SIN)
                REFERENCES EMPLOYEE

CREATE TABLE WORKS_ON (
                Hours INTEGER,
                Worker_SIN INTEGER NOT NULL,
                Project_Name CHARACTER (40) NOT NULL)
PRIMARY KEY (Worker_SIN, Project_Name)
FOREIGN KEY (Worker_SIN)
                REFERENCES EMPLOYEE
FOREIGN KEY (Project_Name)
                REFERENCES PROJECT

CREATE TABLE SUPPLY (
                Quantity INTEGER,
                Project_Name CHARACTER (40) NOT NULL,
                Supplier_Name CHARACTER (40) NOT NULL,
                Part_Number INTEGER NOT NULL)
PRIMARY KEY (Project_Name, Supplier_Name, Part_Number)
FOREIGN KEY (Project_Name)
                REFERENCES PROJECT
FOREIGN KEY (Supplier_Name)
                REFERENCES SUPPLIER
FOREIGN KEY (Part_Number)
                REFERENCES PART

CREATE TABLE DEPARTMENT_Locations (
                DEPARTMENT_Number INTEGER NOT NULL,
                Locations CHARACTER (40))
PRIMARY KEY (DEPARTMENT_Number, Locations)
FOREIGN KEY (DEPARTMENT_Number)
                REFERENCES DEPARTMENT

CREATE TABLE PROJECT_Locations (
                PROJECT_Name CHARACTER (40) NOT NULL,
                Building CHARACTER (40),
                Room CHARACTER (10))
PRIMARY KEY (PROJECT_Name, Building, Room)
FOREIGN KEY (PROJECT_Name)
                REFERENCES PROJECT
```

B Sample Target ER Schema

entity (CONSULTANT)
entityAttribute (CONSULTANT, Fname, CHARACTER (20))
entityAttribute (CONSULTANT, Lname, CHARACTER (20))
entityAttribute (CONSULTANT, Address, CHARACTER (40))
key (CONSULTANT, PrimaryKey)
entityAttribute (CONSULTANT, PrimaryKey, COMPOSITE, Fname, Lname)

entity (EMPLOYEE)
entityAttribute (EMPLOYEE, SIN, INTEGER)
entityAttribute (EMPLOYEE, Fname, CHARACTER (20))
entityAttribute (EMPLOYEE, Lname, CHARACTER (20))
entityAttribute (EMPLOYEE, Address, CHARACTER (40))
key (EMPLOYEE, SIN)

relationship (EMPLOYEE_DEPARTMENT_Rel1, Binary)
entityInRelationship (EMPLOYEE_DEPARTMENT_Rel1, EMPLOYEE, (1, 1), EMPLOYEE)
entityInRelationship (EMPLOYEE_DEPARTMENT_Rel1, DEPARTMENT, (0, n), DEPARTMENT)

entity (DEPARTMENT)
entityAttribute (DEPARTMENT, Number, INTEGER)
entityAttribute (DEPARTMENT, Name, CHARACTER (40))
key (DEPARTMENT, Number)
key (DEPARTMENT, Name)

relationship (DEPARTMENT_EMPLOYEE_Rel1, Binary)
entityInRelationship (DEPARTMENT_EMPLOYEE_Rel1, DEPARTMENT, (1, 1), DEPARTMENT)
entityInRelationship (DEPARTMENT_EMPLOYEE_Rel1, EMPLOYEE, (0, 1), EMPLOYEE)

entity (PROJECT)
entityAttribute (PROJECT, Name, CHARACTER (40))
key (PROJECT, Name)

relationship (PROJECT_DEPARTMENT_Rel1, Binary)
entityInRelationship (PROJECT_DEPARTMENT_Rel1, PROJECT, (1, 1), PROJECT)
entityInRelationship (PROJECT_DEPARTMENT_Rel1, DEPARTMENT, (0, n), DEPARTMENT)

entity (SUPPLIER)
entityAttribute (SUPPLIER, Name, CHARACTER (40))
key (SUPPLIER, Name)

entity (PART)
entityAttribute (PART, Number, INTEGER)
key (PART, Number)

weakEntity (DEPENDENT, DEPENDENT_IdentRel)
entityAttribute (DEPENDENT, Relationship, CHARACTER (40))
entityAttribute (DEPENDENT, Name, CHARACTER (40))

partialKey (DEPENDENT, Name)

identifyingRelationship (DEPENDENT_IdentRel, Binary)
entityInRelationship (DEPENDENT_IdentRel, DEPENDENT, (1, 1), DEPENDENT)
entityInRelationship (DEPENDENT_IdentRel, EMPLOYEE, (0, n), EMPLOYEE)

relationship (WORKS_ON, Binary)
relationshipAttribute (WORKS_ON, Hours, INTEGER)
entityInRelationship (WORKS_ON, EMPLOYEE, (0, n), EMPLOYEE)
entityInRelationship (WORKS_ON, PROJECT, (0, n), PROJECT)

relationship (SUPPLY, n_ary)
relationshipAttribute (SUPPLY, Quantity, INTEGER)
entityInRelationship (SUPPLY, PROJECT, (0, n), PROJECT)
entityInRelationship (SUPPLY, SUPPLIER, (0, n), SUPPLIER)
entityInRelationship (SUPPLY, PART, (0, n), PART)

multivaluedAttribute (DEPARTMENT, Locations, CHARACTER (40))

multivaluedAttribute (PROJECT, PROJECT_Locations, COMPOSITE)
multivaluedAttrComponent (PROJECT, MultiValuedAttribute, Building, CHARACTER (40))
multivaluedAttrComponent (PROJECT, MultiValuedAttribute, Room, CHARACTER (10))

C Proofs

Due to space limitations, complete proofs are not provided for Lemmas $2-5$. Detailed proofs may be found in [19].

C.1 Lemma 1

Given relational database schema $D =< R, C >$, the classification imposed on R by the translation system TS_{RER} is a partition.

PROOF: To prove that TS_{RER} induces a partition of the relation schemes in R we must show that each relation scheme falls into exactly one category. We provide a proof by contradiction by considering the two cases which violate the condition, namely that a relation scheme falls into more than one category and a relation scheme does not fall into any category.

Case 1: Relation scheme $R_i \in R$ falls into more than one of the categories induced by TS_{RER}.

If R_i is a primary relation scheme then $PK_i \cap FK_i = \phi \Rightarrow R_i$ is not in any other category since in all other categories $FK_i \subseteq PK_i$.

If R_i is a secondary relation scheme then $PK_i = FK_i \Rightarrow R_i$ is not a weak primary or weak secondary

relation scheme since in both cases $FK_i \subset PK_i$.

If R_i is a weak primary relation then $|X_i - PK_i| > 0 \Rightarrow R_i$ is not a weak secondary relation since in that case $X_i = PK_i$.

Therefore there is a contradiction for case 1.

Case 2: Relation scheme $R_i \in R$ is not in any of the categories induced by TS_{RER}.

Thus R_i is not a primary relation and its primary key must contain a foreign key. In other words, $\exists fk_1 \ldots fk_j$, $j \geq 1$ where fk_k is the primary key of relation scheme $R_k \in R$ $(1 \leq k \leq j)$ such that $fk_1 \cup \ldots \cup fk_j \subseteq FK_i$ and $fk_1 \cup \ldots \cup fk_j \subseteq PK_i$.

if PK_i does not contain any other attributes, that is $PK_i - (fk_1 \cup \ldots \cup fk_j) = \phi$, then R_i is a secondary relation scheme.

Otherwise, if PK_i contains other attributes and there are also nonprime attributes, that is $X_i - PK_i \neq \phi$, then R_i is a weak primary relation scheme.

Otherwise, the primary key must contain all the attributes, including the foreign keys, and R_i is a weak secondary relation scheme.

Therefore R_i is in one of the categories induced by TS_{RER} and so there is a contradiction for case 2.

Therefore the classification imposed on a set of relation schemes by the translation system TS_{RER} is a partition.

C.2 Lemma 2

Given relational database schema $D =< R, C >$, the syntactic rewrite rule tr_p is an information capacity preserving translation on its partition of R.

PROOF: Suppose that the function on the sets of instances induced by the rule is

$$f_p : I(R_p) \rightarrow I(T_{ER})$$

where R_p is the set of primary relations from the source relational schema D and T_{ER} is the target ER schema. f_p maps tuples from a primary relation into entities in an entity type of the same name as the relation. Attribute values of the tuple are mapped to values of corresponding attributes of the entity, which are established by the rules which create the attributes, *unique* and *primaryKey*

constraints on the tuple are reflected as *key* constraints on the entity and *references* constraints on the tuple are represented as relationships involving the entity.

The function f_p is total since every tuple in $I(R_p)$ can be mapped to a corresponding entity in $I(T_{ER})$. It is also an injection since its inverse, f_p^{-1}, is guaranteed to map an entity back to its original source tuple because all attribute values, and constraint information, are maintained by the mappings. In other words, for all $i \in I(R_p)$

$$i = f_p^{-1} \circ f_p(i)$$

Since the translation rule induces a function on the sets of instances that is both total and injective, we conclude that the rule is an information capacity preserving translation.

C.3 Lemma 3

Given relational database schema $D =< R, C >$, the syntactic rewrite rule tr_{wp} is an information capacity preserving translation on its partition of R.

PROOF: Similar argument to the proof of Lemma 2.

C.4 Lemma 4

Given relational database schema $D =< R, C >$, the syntactic rewrite rule tr_s is an information capacity preserving translation on its partition of R.

PROOF: Similar argument to the proof of Lemma 2.

C.5 Lemma 5

Given relational database schema $D =< R, C >$, the syntactic rewrite rule tr_{ws} is an information capacity preserving translation on its partition of R.

PROOF: Similar argument to the proof of Lemma 2.