Schema Translation Using Structural Transformation *

Rateb Abu-Hamdeh, James Cordy and Patrick Martin
Dept. of Computing and Information Science
Queen's University at Kingston

Abstract

This paper describes how structural transformation can be applied to the problem of translating schemas expressed in one data model into equivalent schemas expressed in another data model. We explain our approach to the problem, which involves translating a schema in the source data model into a set of facts in a knowledge base and from there into a schema in the target data model. We present an example transformation in detail and outline how one can analyze the information capacity preserving properties of the transformation.

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 [3]. 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 federated database, systems [13], database migration [11], and schema evolution [16]. As part of the CORDS Multidatabase System (MDBS) project, 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 [6, 20].

In this paper we discuss our approach to schema translation. Section 2 presents a brief review of related work. Section 3 defines the problem of schema translation and gives an explanation of structural transformation and how it can apply to the problem. Section 4 presents our approach in more detail and gives an example of how it may be used. Section 5 outlines how the notion of information capacity may be used to formally evaluate the correctness of the translation schemes produced using our approach. Section 6 presents our conclusions.

2 Related Work

Schema translation schemes have been developed to translate between a variety of data models. Most of the work related to conceptual schema design has focussed on schemes to translate 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 [5], Markowitz and Makowsky [19], and Davis and Arora [10] all describe approaches to extracting the logical structure from relational schemas. Work has also been reported on mapping schemas be-

*This paper appears in *Proc. of CASCON'94*, Toronto, Nov. 1994, pp.202-215

tween the traditional data models. The two approaches used have been either to map schemas directly between data models, for example Zaniolo [24], or to map to an intermediate representation. Biller [4], for example, uses a semantic data model as an intermediate form when mapping relational schemas to network schemas.

One problem with most of these translation schemes, 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 [19], 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 schemes produced can be automated and translate existing schemas without the 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 Schema Translation

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 rules for arranging the structures to represent the situation. The description of a particular database in the DDL is called the *database schema*. Schema translation produces a new schema in a different, or perhaps the same, data model, which consists of different structures but which provides the same interpretation of the database.

The rules 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. An example of the former type of rule is the representation of a one-to-many relationship in the entity-relationship model, for example the "works for" relationship in Figure 1. An example of the latter type of rule is to usually model the same relationship in the relational model by placing the foreign key of the "DEPARTMENT" relation in the "EMPLOYEE" relation. In either case, the rule 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.

Figure 1 shows a simple example of structural transformation. The source schema, in part A of the figure, is an ER diagram consisting of the entities "DEPARTMENT", "PROJECT", and "EMPLOYEE", and the relationships "works for", "controls", and "works on", which are a one-to-many mapping from "DEPARTMENT" to "EMPLOYEE", a one-to-many mapping from "DEPARTMENT" to "PROJECT", and a many-to-many mapping between "EMPLOYEE" and "PROJECT", respectively. The target data model is a hierarchical model and the target schema is shown in part C. The transformations of entities to segments and one-to-many relationships to parent-child links are straightforward. The transformation of the many-to-many relationship, as depicted in part B, involves the generation of virtual segments matching "EMPLOYEE" and "PROJECT" and an *unfolding* of the "works on" relationship into two one-to-many relationships.

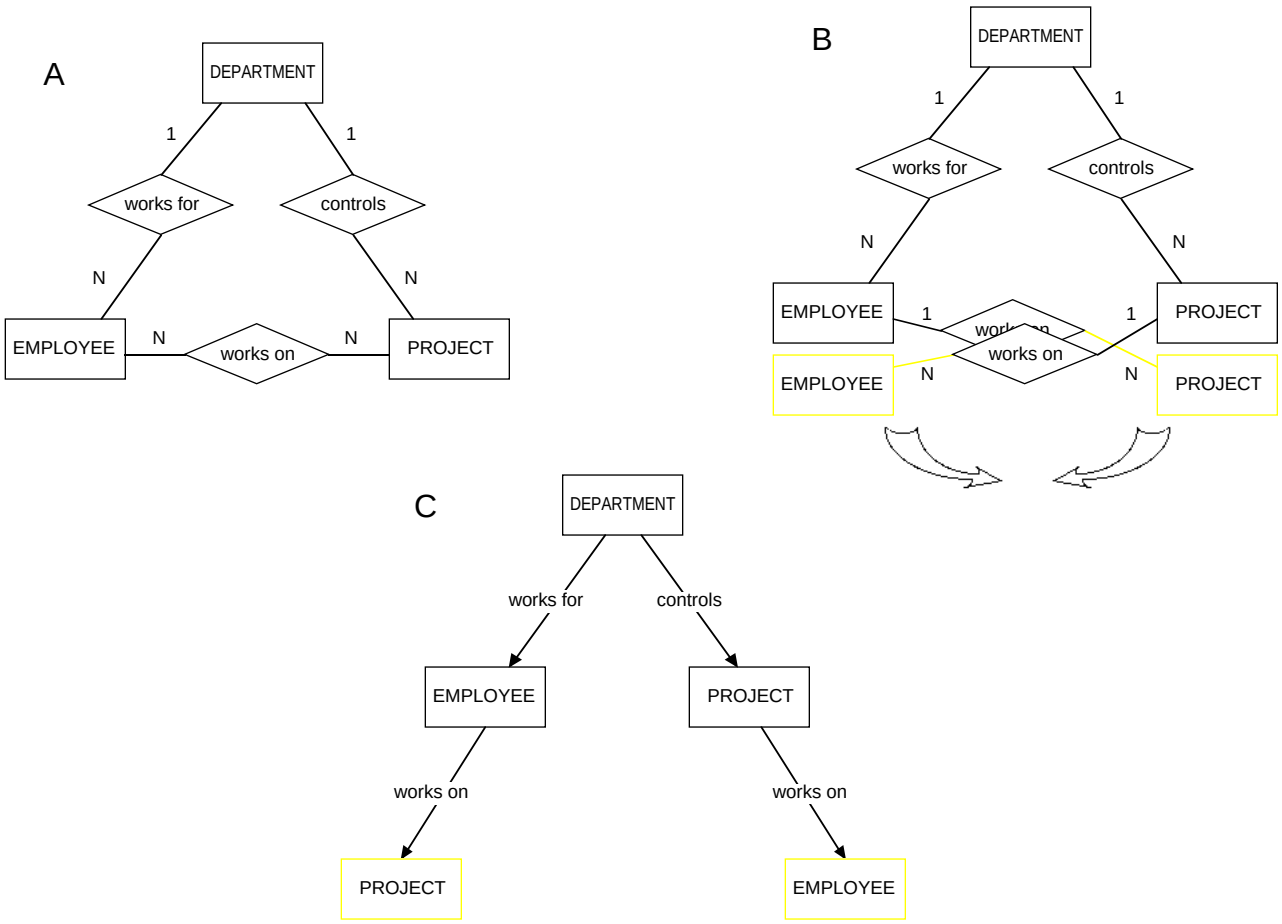

Figure 1: Sample Structural Transformation

As we noted earlier, a major concern in developing a translation scheme from one data model to another is potential loss of information, that is, there may be patterns in the source schemas that cannot be adequately represented in the target schema. We may evaluate the correctness of a translation scheme 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 evaluation of the correctness of a translation scheme 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 scheme 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 scheme must ensure that data maps to representations in both directions. The issues related to information capacity of schemas produced with our translation schemes are discussed in more detail in Section 5.

4 Structural Transformation

The approach we have taken consists of implementing the structural transformations between data models such as those shown in Figure 1 as syntactic transformations between the data description language (DDL) representations of the schemas to be transformed. To limit the number of transformation tasks, we have chosen to use the Entity-Relationship (ER) data model as a kind of "lingua franca" through which we can translate schemas from the data description language of any one data model to that of any other. Thus, for example, to translate a schema from the hierarchical to the relational model, we will first translate the hierarchical schema to the ER model and then translate the new ER schema to the relational model.

The Entity-Relationship model serves as an appropriate translation medium because it is a superset of the three traditional data models. The only practical difficulty is that, unlike other models, the ER model has no standard DDL for expressing schemas. Our solution to this problem is to use a Prolog factbase (i.e., sequence of predicates) to represent ER schemas as lists of facts about entities and their relationships.

Structural transformations can be implemented as source-to-source syntactic transformations provided that, for each data model, (i) there exists a concrete data description language for expressing schemas in the model and (ii) 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 [7] to implement the syntactic transformations.

TXL is a general purpose source-to-source translation system designed for implementing syntactic transformations. TXL has been used to prototype programming languages [8], to implement software engineering tasks such as remodularization of source code [23], metaprogramming [9], design recovery [17], and for many other tasks.

TXL "programs" consist of a context-free grammar describing the syntactic forms of the input and output languages and a set of by-example transformation rules to be applied to inputs parsed using the grammar. TXL operates in three phases: *parse*, *transform*, and *unparse*. The *parse* phase involves parsing the input according to the grammar defined in the program. The result is a labeled parse tree of the input. The *transform* phase involves applying the transformation rules provided in the program to the parse tree to produce a transformed parse tree that corresponds to the desired output. Finally, the *unparse* phase unparses the transformed tree to produce the source output.

In our case, the input will be a database schema described in the DDL of one data model, the output a database schema described

in the DDL of another model, and the transformation rules will describe the structural transformations between the models as syntactic transforms between the syntactic forms of the input DDL and the syntactic forms of the output DDL. In the following section, we demonstrate this technique using the relational to ER translation as an example.

4.1 Implementation of the Relational-to-ER Translation Scheme

In this section, we discuss as an example the implementation of a Relational-to-ER translation scheme as a representative source transformation task using TXL. The implementation of any other translation will be similar. The Relational-to-ER translation translates a relational schema to an ER schema. The translation is implemented as a TXL program which takes a relational schema written in the SQL data definition language [2] as input and produces the corresponding ER schema represented as a Prolog factbase as output. The general strategy of the transformation is

The main transformation rule of this TXL program is shown in Figure 2. The two **include** statements begin the program with grammars describing the syntactic forms of the input SQL data definition language and the output Prolog factbase. To allow the TXL program to transform from one to the other, the new syntactic type *program* combines these two grammars into one syntactic form defined as a sequence of relation definitions (that is, a relational schema) followed by a sequence of Prolog predicates (that is, a factbase representing the corresponding ER schema).

The transformation proceeds by adding ER facts for each relation definition to the initially empty factbase until all of the relation definitions have been translated and the entire schema is represented in the ER factbase. This same approach is used by all four TXL programs which implement our translation schemes.

The input to the program is defined as two possibly empty schemas; one schema corresponds to the input data model, whereas the other corresponds to the output data model. When an input schema is given to the program, it is parsed into a tree of type *program*, which is composed of two subtrees A and B; A is the parse tree of the input schema, while B corresponds to the output schema and is initially empty. In the transform phase, a parse tree of the output schema is iteratively added to B, while A is gradually removed. Finally, the result parse tree is unparsed to produce the source text form of the output schema.

We now describe some of the transformation rules used in the implementation of the Relational-to-ER translation scheme. The general strategy of the Relational-to-ER translation is to represent each table definition and its attributes as an *entity* fact and a set of associated *entityAttribute* facts. Each unique or primary key is translated into a *key* fact, and each foreign key is represented by a *relationship* fact along with associated *entityInRelationship* facts. In the following sections, we discuss the TXL main rule and subrules that implement these translations as syntactic transformations.

4.2 The Main Rule

The main transformation rule is shown in Figure 2. *mainRule* is a TXL function, which means it is a rule applied once to the entire parse tree of the input. TXL transformation rules may either be *functions*, which means that they are to be applied exactly once, or *rules*, which means that they are to be repeatedly applied until no more instances of their transformation can be made.

The *pattern* of function *mainRule* (the part following the keyword **replace**) consists of two variables RS and FB, of syntactic type [**repeat** tableDefinition] (that is, a sequence of relational table definitions) and [**repeat** predicate] (that is, a sequence of Prolog predicates), respectively. RS corresponds to the input relational schema, and FB corresponds to the initially empty factbase. The type of the pattern is [program], the root syntactic type of the combined grammars. The *replacement* of the function (the part following the keyword **by**) is the original empty factbase FB as modified by two other TXL functions, *ProcessRelationsWithoutFKs* and *ProcessRelationsWithFKs*, parameterized by the original relational

```
include "Relational_DDL.Grammar"
include "ER_Factbase.Grammar"

define program
        [repeat tableDefinition]
        [repeat predicate]
end define

function mainRule
    replace [program]
        RS [repeat tableDefinition]
        FB [repeat predicate]
    by
        FB [ProcessRelationsWithoutFKs each RS]
           [ProcessRelationsWithFKs each RS]
end function
```

Figure 2: mainRule TXL program Relational-to-ER

schema RS.

When *mainRule* is applied to the parse tree of the input, RS is bound to the sequence of relation definitions in the input relational schema, while FB is bound to an empty sequence of predicates. *mainRule* replaces the entire input with an empty sequence of relation definitions (implied by the absence of RS in the result) and the result of applying the TXL functions *ProcessRelationsWithoutFKs* and *ProcessRelationsWithFKs* applied to the initially empty factbase FB. These two functions construct the output ER factbase by successively adding predicates for each table definition in RS to FB.

RS is given as an actual parameter to each of the two functions. In both functions, the actual parameter RS is preceded by the TXL keyword **each**; this means that each transformation function is to be applied once for each relation definition in the input relational schema.

The two functions partition the transformation into two independent cases, relation definitions without foreign keys and those with foreign keys. If a table definition passed to the function *ProcessRelationsWithoutFKs* does not contain FOREIGN KEY statements, then the relation represents an entity; *ProcessRelationsWithoutFKs* constructs a sequence of predicates which represent the entity, and the predicates are appended to the factbase FB. If the relation definition passed does contain FOREIGN KEY statements, *ProcessRelationsWithoutFKs* fails (and does nothing).

ProcessRelationsWithFKs processes the relation definitions for which the first function fails, namely the relation definitions which contain FOREIGN KEY statements, which correspond to definitions of entities with relationships or multi-valued attributes for which corresponding facts must be generated. Thus every relation definition in the relational schema is transformed by exactly one of these two functions. In either case, a number of predicates representing the definition are added to the factbase FB.

4.3 The *ProcessRelationsWithoutFKs* function

The function *ProcessRelationsWithoutFKs* handles the first case in the Relational-to-ER translation scheme; it is shown in Figure 3. It has one formal parameter *TD* of type [tableDefinition], the syntactic type in the relational DDL grammar corresponding to a relational table definition. As explained above, the function is applied once for each table definition in the original relational schema.

The first statement in the function is a TXL

```
function ProcessRelationsWithoutFKs TD [tableDefinition]
    deconstruct TD
        'CREATE 'TABLE T [table] ( ADList [list attributeDefinition] )
        UniqueStatements [repeat uniqueConstraint]
        'PRIMARY 'KEY ( PKAList [list attribute] )

    construct ENT [entityName]
        T
    construct InitialPred [repeat predicate]
        'entity ( ENT )

    construct NewPredicates [repeat predicate]
        InitialPred [ProcessAttributeDef ENT each ADList]
                    [AddPKPreds_Atomic ENT PKAList]
                    [AddPKPreds_Composite ENT PKAList]
                    [AddUniquePreds_Atomic ENT each UniqueStatements]
                    [AddUniquePreds_Composite ENT each UniqueStatements]
    replace [repeat predicate]
        FB [repeat predicate]
    by
        FB [. NewPredicates]
end function

function ProcessAttributeDef ENT [entityName]
                         AttrDef [attributeDefinition]
    deconstruct AttrDef
        Attr [attribute] Type [dataType] _ [opt notNull]
    construct EAPred [predicate]
        entityAttribute ( ENT, Attr, Type )

    replace [repeat predicate]
        Predicates [repeat predicate]
    by
        Predicates [. EAPred]
end function
```

Figure 3: The function *ProcessRelationsWithoutFKs* and one of the functions that it applies.

deconstruct statement that splits the table definition TD into its constituent parts using a syntactic pattern. TXL patterns attempt to match the parse tree they are given (in this case the parse tree of the table definition bound to TD) to a particular syntactic form, in this case that of a table definition with a PRIMARY KEY statement. In this case, the pattern of the deconstruct matches only table definitions that do not have FOREIGN KEY statements and thus will fail to match if TD contains any FOREIGN KEY statements. In TXL this means that the function will fail and thus do nothing. If TD does not contain any FOREIGN KEY statements, then the pattern match succeeds and the function proceeds. According to our translation, relations which do not contain foreign keys represent entity types. Thus the function maps the table definition TD to a set of predicates which represent a corresponding entity type in the ER factbase.

The two **construct** statements following the deconstruct create the TXL variable *InitialPred* and bind it to the predicate *entity (ENT)*, where ENT is a variable of syntactic type [entityName] bound to the name of the original relation. TXL **construct** statements construct a new parse tree by parsing the sequence of symbols and TXL variables given in the construct body into a parse tree of the specified type, and bind the result to a new TXL variable. In this case, the first constructor makes the table name bound to T into an [entityName] bound to variable ENT and the second one constructs the entity predicate for the table using ENT. The "**construct** NewPredicates" statement constructs the actual sequence of predicates that correspond to the table definition and binds them to the TXL variable *NewPredicates*. *NewPredicates* is constructed by applying a number of transformation rules to the variable *InitialPred*. Each rule is passed a part of the table definition as a parameter; it then constructs one or more predicates to represent that part of the definition and adds them to the sequence of predicates bound to *InitialPred*.

As an example, the function *ProcessAttributeDef* constructs the predicates which represent the attributes of the entity type. It is passed each attribute definition in the relation by passing the variable ADList (attribute definition list) preceded by the TXL keyword **each**. (Thus the function is applied once for each attribute definition in ADList.) For each attribute, it creates a corresponding entityAttribute predicate and appends it to the predicate sequence.

AddPKPreds_Atomic and *AddPKPreds_Composite* construct the predicate(s) that correspond to the primary key of the relation definition. Only one of the two functions succeeds: *AddPKPreds_Atomic* succeeds if the primary key of the relation contains one attribute, whereas *AddPKPreds_Composite* succeeds if the primary key of the relation contains more that one attribute, in which case the primary key of the entity type becomes a composite attribute.

The functions *AddUniquePreds_Atomic* and *AddUniquePreds_Composite* add a key predicate for each UNIQUE statement in the relation definition. Again, one rule handles UNIQUE statements which refer to one attribute, while the other handles statements which refer to multiple attributes.

Once the sequence of predicates representing the entity type is constructed, it is appended to the sequence of predicates constructed so far (FB). This is done by the **replace** and **by** statements at the end of the function. The sequence FB is replaced by FB [. NewPredicates]. The rule [.] is a built-in TXL rule that appends the sequence passed to it as a parameter to the sequence to which it is applied.

To illustrate the operation of the function *ProcessRelationsWithoutFKs*, consider the relation table definition

```
CREATE TABLE DEPARTMENT (
    Dnumber INTEGER NOT NULL,
    Dname CHARACTER (20) NOT NULL,
    Location CHARACTER (20))
UNIQUE(Dname)
PRIMARY KEY (Dnumber)
```

This table definition does not contain any FOREIGN KEY statements, and therefore it represents an entity type according to the Relational-to-ER translation scheme. If this table definition is passed to the function *ProcessRelationsWithoutFKs*, the function creates the following predicates:

```
function ProcessRelationsWithFKs TD [tableDefinition]
    deconstruct TD
        'CREATE 'TABLE T [table] ( ADList [list attributeDefinition] )
        UniqueStatements [repeat uniqueConstraint]
        'PRIMARY 'KEY ( PKAList [list attribute] )
        FKStatements [repeat foreignKeyDefinition]

    construct EmptyFKStatements [repeat foreignKeyDefinition]

    where not
        FKStatements [= EmptyFKStatments]

    construct FKAttributes [repeat attribute]
        _ [↑ FKStatements]

    replace [repeat predicate]
        FB [repeat predicate]
    by
        FB [ProcessManyToManyRelationship T ADList PKAList
                FKStatements FKAttributes]
           [ProcessEntity T ADList UniqueStatements PKAList
                FKStatements FKAttributes]
           [ProcessWEorMVA T ADList PKAList FKStatements FKAttributes]
end function
```

Figure 4: The function *ProcessRelationsWithFKs* applied by the mainRule of the TXL program Relational-to-ER

```
entity (DEPARTMENT)
entityAttribute (DEPARTMENT, Dnumber,
          INTEGER)
entityAttribute (DEPARTMENT, Dname,
          CHARACTER (20))
entityAttribute (DEPARTMENT, Location,
          CHARACTER (20))
key (DEPARTMENT, Dnumber)
key (DEPARTMENT, Dname)
```

This sequence of predicates is bound to the variable NewPredicates, which is then appended to the whole sequence of predicates created so far as explained above.

The function *ProcessAttributeDef*, which is applied by *ProcessRelationsWithoutFKs*, is also shown in Figure 3. It is passed the entity name and an attribute definition as parameters. The attribute definition is then split into its constituent parts using a deconstruct pattern; these parts are used along with the entity name to construct a predicate that corresponds to the attribute definition. The predicate constructed is then appended to the sequence of predicates constructed so far.

4.4 The *ProcessRelationsWithFKs* function

The function *ProcessRelationsWithFKs* performs the second step in the Relational-to-ER translation scheme; it is shown in Figure 4. It is similar in structure to the function *ProcessRelationsWithoutFKs*, and is also applied once for each relation definition in the relational schema. However, it succeeds only if the table definition passed to it contains one or more FOREIGN KEY statements.

The function begins with a TXL **deconstruct** statement that breaks up the tables definition TD into its constituent parts. In particular, the foreign key statements in the table, if any, are bound to the TXL variable *FKStatements*.

The application of the function is guarded by a TXL **where** condition that insists that *FKStatements* is not equal to *EmptyFKStatements*, a constructed example of an empty sequence of foreign keys. **where** conditions allow a TXL rule or function to place additional conditions under which the rule or function is to be applied. In this case, the **where** condition will fail if and only if *FKStatements* is empty - that is, there are no foreign key statements in the table definition. If that is the case, then the function fails (and does nothing). Hence the function proceeds only if there is one or more FOREIGN KEY statements in the table definition passed to it.

Once the function ensures that *FKStatements* is not empty, it adds the appropriate set of predicates to FB by applying three subrules to it. *ProcessManyToManyRelationship* handles the case where *FKStatements* defines a many-to-many relationship, *ProcessEntity* handles the case where it contains the binary relationships defining an entity, and *ProcessWEorMVA* handles the case where it represents a weak entity or multi-valued attribute.

The components of the table definition to be transformed are passed as parameters to each of the subrules. Each subrule corresponds to one of the three cases above and is guarded by a **where** condition corresponding to the case it handles. Since exactly one of the three conditions is satisfied for a given table definition, only one rule out of the three subfunctions succeeds and adds the appropriate predicates.

To illustrate the operation of the function *ProcessRelationsWithFKs*, consider the relation table definition

```
CREATE TABLE EMPLOYEE (
    Fname CHARACTER (20) NOT NULL,
    Lname CHARACTER (20) NOT NULL,
    Ssn INTEGER NOT NULL,
    Bdate CHARACTER (8),
    Address CHARACTER (40)
    Dnumber INTEGER NOT NULL)
UNIQUE(Ssn)
PRIMARY KEY (Fname, Lname)
FOREIGN KEY (Dnumber)
    REFERENCES DEPARTMENT
```

The presence of a FOREIGN KEY statement in the definition implies that one of several translations is possible: a separate entity and a one-to-many relationship to another entity; a many-to-many relationship; a multi-valued attribute, or a weak entity and a relationship. In this case, since the foreign key is not part of the primary key, *ProcessRelationsWithFKs* translates the EMPLOYEE table into an entity-type

EMPLOYEE and a relationship-type between the entity-types EMPLOYEE and DEPARTMENT which are represented by the following predicates:

```
entity (EMPLOYEE)
entityAttribute (EMPLOYEE, primaryKey,
          COMPOSITE, Fname, Lname)
entityAttribute (EMPLOYEE, Fname,
          CHARACTER (20))
entityAttribute (EMPLOYEE, Lname,
          CHARACTER (20))
entityAttribute (EMPLOYEE, Ssn,
          INTEGER)
entityAttribute (EMPLOYEE, Bdate,
          CHARACTER (8))
entityAttribute (EMPLOYEE, Address,
          CHARACTER (40))
key (EMPLOYEE, primaryKey)
key (EMPLOYEE, Ssn)

relationship (D-E, Binary)
entityInRelationship (D-E, EMPLOYEE,
          (1,1), employee)
entityInRelationship(D-E, DEPARTMENT,
          (1,n), department)
```

In this section, we have discussed the implementation of only one of our six translation schemes as source transformation tasks using TXL. We have so far implemented translations between the ER model and the relational, hierarchical, and network models. The implementations of the other schemes are all similar, iteratively taking each part of the input schema definition and translating it to its representation in the output data model syntax.

5 Information Capacity

We observed earlier in the paper that one of the problems with much of the previous work on schema translation is that it has been based on an intuitive, rather than a formal, notion of correctness. Some recent work, however, has succesfully used information capacity as a basis for judging the correctness of transformed schemas [14, 16, 21].

The *information capacity* of a schema S, as defined by Miller, Ioannidis and Ramakrishnan [21], determines the set of valid instances $I(S)$ of that schema. Intuitively, we say that a schema T has a greater information capacity than a schema S if every instance $i \in I(S)$ can be mapped to some instance $j \in I(T)$ 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 1 *An information capacity preserving mapping between the instances of two schemas S and T is a total, injective function $f : I(S) \rightarrow I(T)$.*

Definition 2 *If $f : I(S) \rightarrow I(T)$ is an information capacity preserving mapping, then T dominates S via f, denoted $S \preceq T$.*

Definition 3 *Let $\boldsymbol{S}$ and $\boldsymbol{T}$ be families of schemas specified in different data models. A* schema translation *is a total function $F : \boldsymbol{S} \rightarrow \boldsymbol{T}$. A schema translation is information capacity preserving if for all $S \in \boldsymbol{S}$, $S \preceq F(S)$.*

The important practical implication of an information capacity preserving translation is that the entire database stored under the source schema may be viewed and queried through the target schema [21].

We discuss the Relational-to-ER scheme as an example.

Proposition 1 *The Relational-to-ER translation scheme is an information capacity preserving schema translation.*

Sketch of Proof:
A sketch of the proof of the proposition is provided here. A more detailed discussion is given in Abu-hamdeh's M.Sc. thesis [1].

In our approach to schema translation, a translation scheme is expressed as a set of independent transformation 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 [18]. Thus the proof of the proposition can be reduced to proving that each individual rule is an information capacity preserving translation, and that the set of rules in the translation scheme

is complete, that is, that all syntactic structures in the source schema are translated to a corresponding structure in the target schema.

The proof of the claim that the Relational-to-ER translation scheme is complete is based on the observation that table definitions are partitioned into two disjoint sets, namely those tables with foreign keys and those tables without foreign keys, and that this partitioning obviously includes all table defintions. As an example of proving that a rule is information capacity preserving, consider the rule *ProcessRelationsWithoutFKs* described in Section 4.

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

$$f_{RWFK} : I(S_R) \rightarrow I(T_{ER})$$

where S_R is a source relational schema and T_{ER} is the target ER schema. f_{RWFK} maps tuples from a relation without a foreign key 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, and "UNIQUE" and "PRIMARY KEY" constraints on the tuple are reflected as "KEY" constraints on the entity.

One piece of information that cannot be captured in the syntax of the ER factbase is, in the case of multiple candidate keys, which key is the primary key since they are all mapped to a fact of type "KEY". The translation scheme annotates the fact with a comment that it represents a primary key which the reverse translation scheme can then exploit in reconstructing the original schema. If there is no annotation, the first "KEY" fact is assumed to be the primary key.

The function f_{RWFK} is total since every tuple in $I(S_R)$ from a relation without a foreign key can be mapped to a corresponding entity in $I(T_{ER})$. It is also an injection since its inverse, f^{-1}_{RWFK}, is guaranteed to map an entity back to its original source tuple because all attribute values, and the primary key information, are maintained by the mappings. In other words, for all $i \in I(S_R)$

$$i = f^{-1}_{RWFK} \circ f_{RWFK}(i)$$

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.

We have been able to show, using this form of proof, that all of our translation schemes between the various data models are information capacity preserving. These results mean that we are able to query the data represented by the source schema through the target schema and obtain valid results. We have also obtained even stronger results for the transforms mapping hierarchical to relational and to ER, and mapping relational to ER, namely, that the transformation schemes are *equivalence preserving*, which implies that the source database may be updated through the target schema.

6 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.

We have developed translation schemes to transform schema descriptions in representative relational, hierarchical, and network DDLs into an ER factbase representation, and vice versa. We can compose the translation schemes to translate among the three traditional data models.

Our approach expresses a translation scheme as a set of independent transformation rules. We indicated in the paper how one can prove the correctness of a scheme by showing that the rules preserve information capacity. We

have been able to prove that all of our schema translation schemes among the models are information capacity preserving and that some are even equivalence preserving. This latter property means that one can update the source database through the target schema.

We plan to continue to study the problem of schema translation and, in particular, to look at translation from an object-oriented data model and at automatic query translation. We are also incorporating the TXL implementations of the translation schemes into a suite of tools to support database integration in a prototype multidatabase system [20].

Acknowledgments

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

About the Authors

Rateb Abu-hamdeh received his M.Sc degree from Queen's University at Kingston in 1994. He is currently working at Newbridge Networks Corporation and may be contacted at Newbridge Networks Corporation, 600 March Road, P.O. Box 13600, Kanata, Ontario, K2K 2E6, or by email at ratebh@Newbridge.com.

James R. Cordy is associate professor of computing and information science at Queen's University at Kingston, Canada. Dr. Cordy is the co-designer of the programming languages Concurrent Euclid, Turing and Turing Plus, the S/SL compiler specification language, the TXL transformation language and the visual language GVL. He is a co-author of the books "The Turing Programming Language: Design and Definition" (1988) and "Languages for Developing User Interfaces" (1992). Dr. Cordy was program chair of ICCL'92, the IEEE 1992 International Conference on Computer Languages, serves on the program committees of ICCL'94 and CASE'95, and is a member of the editorial board of the Journal of Programming Languages. He may be reached at (613) 545 6054, email cordy@qucis.queensu.ca.

Patrick Martin is an associate professor at Queen's University at Kingston and is a principal investigator in the CORDS project. He may be contacted at the Department of Computing and Information Science, Queen's University, Kingston, Ontario, K7L 3N6 or by email at martin@qucis,queensu.ca.

References

[1] R. Abu-Hamdeh. Automatic schema translation using structural transformation. Master's thesis, Department of Computing and Information Science, Queen's University at Kingston, February 1994.

[2] ANSI, Document X3.135-1989. *Database Language–SQL with integrity enhancement*, October 1989.

[3] 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.

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

[5] 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.

[6] N. Coburn, P.-Å Larson, P. Martin, and J. Slonim. Cords multidatabase project: Research and prototype overview. In *Proc. of CASCON '93*, pages 767–778, Toronto, 1993.

[7] 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.

[8] J.R. Cordy and E.M. Promislow. Specification and automatic prototype implementation of polymorphic objects in turing using the TXL dialect processor. In *Proc. IEEE 1990 International Conference on Computer Languages*, pages 145–154, New Orleans, 1990.

[9] J.R. Cordy and M. Shukla. Practical metaprogramming. In *Proc. CASCON '92*, pages 215–224, Toronto, 1992.

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

[14] R. Hull. Relative information capacity of simple relational database schemata. *SIAM Journal of Computing*, 15(3), 1986.

[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] D.A. Lamb and K.A. Schneider. Formalization of information hiding design methods. In *Proc. CASCON '92*, pages 201–214, Toronto, 1992.

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

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

[20] P. Martin and W. Powley. Database integration using multidatabase views. In *Proc. of CASCON '93*, 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] R. Srinivasan. Automatic software design recovery and re-modularization using source transformation. Master's thesis, Department of Computing and Information Science, Queen's University at Kingston, April 1993.

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