An Executable Meta Model for Re-Engineering of Database Schemas*

Manfred A. Jeusfeld†, Uwe A. Johnen‡

†Informatik V, RWTH Aachen, 52056 Aachen, Germany
‡debis Systemhaus - Software Tools, 52076 Aachen, Germany

Abstract

A logical database schema, e.g. a relational one, is an implementation of a specification, e.g. an entity-relationship diagram. Upcoming new data models and the necessity of seamless integration of databases into application programs require a cost-effective method for mapping from one data model into the other. We present an approach where the mapping relationship is divided into three parts. The first part maps the input schema into a so-called meta model. The second part rearranges the intermediate representation, and the last part produces the schema in the target data model. A prototype has been implemented on top of a deductive object base manager for the mapping of relational schemas to entity-relationship diagrams. From this, a C++-based tool has been derived that will be part of a commercial CASE environment.

*This work was supported in part by the Commission of the European Communities under ESPRIT Basic Research Action 6810 (Compulog 2) and by the Ministry of Science and Research of Nordrhein-Westfalen. A version of this paper appears in Proc. 13th Intl. Conf. on the Entity-Relationship Approach (ER'94), Manchester, UK, Dec. 13-16, 1994.

1 Introduction

A database schema is a collection of statements about all possible database instances. The language for expressing these statement, the data model, is itself a collection of statements about all possible database schemata (belonging to this specific data model).

If there is only one data model, then two database schemata could be always investigated within the language of this data model. For example, the question whether the union of two relational database schemata is consistent can be answered by reasoning in the restricted logical theory of the relational data model. Unfortunately, there are many data models. One reason is the separation of a conceptual modeling and a logical design phase. The first abstracts physical and algorithmic details and is close to knowledge representation. The most prominent example is the entity relationship model. The latter has to be suitable for efficient storage management and query processing. Here, the relational data model has supposedly gained the largest impact.

Since the data models make statements about the same portion of the world, they have to be mapped to each other. The traditional mapping direction is from the conceptual model, usually an entity-relationship diagram (ERD), to the logical data model, usually a relational database schema (RDS). The reverse direction, from the logical data model to the conceptual model, is becoming more and more important due to the need re-engineering legacy information for systems where the original conceptual model is lost or is out of date.

The upcoming new data models, like object-oriented ones, make the need for a flexible mapping environment obvious. For example, a company may want to restructure a relational database into an object-oriented one.

We claim that the multitude of data models make individual solutions too costly. Instead, a generic mapping tool is required that can be parameterized by the properties of the source and target data models of the mapping. Our approach can be summarized as follows:

1. Classify the schema components of the source data model, e.g. a relational schema, into a so-called meta model. We use a deductive query language for this task. Generalization hierarchies are detected by analyzing foreign key dependencies.

2. Find "similar" concepts in the target data model. This is done by navigating in the meta model hierarchy to next neighbors.

3. Instantiate the components in the target data model.

The invariant of this method is the meta model. Ideally, the mapping from and to a new data model can be described by classifying it into the meta model.

The remainder is organized as follows. Section 2 reviews related work in database and software engineering. Section 3 presents the meta model as a hierarchy of concepts and elaborates on the mapping process. Section 4 presents the application of our approach to the mapping from the relational data model into the entity-relationship model. The input RDS is taken from the commercial database CASE environment ProMod-PLUS marketed by debis Systemhaus. The implementation was prototyped in ConceptBase [JARK94]. The paper concludes with a summary and a list of open questions.

2 Related Work

Early contributions to the reverse engineering of relational schemata were made by [CASA83] and [DUMP83]. Both present procedural algorithms that generate a simplified ERD from a RDS. [DAVI88] extended the method of [CASA83] by taking implicit constraints like referential integrity into account. In this approach, the RDS is required to be in third normal form. The presented algorithms for forward and reverse engineering are shown to be inverse to each other.

Mappings to extended ERDs are presented in [BATI92] and [NAVA88]. The latter chose an ERD language with generalization hierarchies. [KALM91] showed and corrected several errors in previous proposals (e.g., chains of foreign keys, attribute inheritance). One drawback remains, however: the mapping requires a lot of human interaction since the knowledge about generalization is missing (or hidden) in the RDS. Furthermore, the mapping rules are represented in a production rule fashion. This makes any kind of reasoning *about* the mapping nearly impossible.

[MARK92] uses first-order logic for restructuring an RDS enriched by key and inclusion dependencies in the presence of existing data. The method is based on a representation of (extended) ERDs in canonical relational schemas. However, it is not specifically a reverse engineering method but a method for giving relational semantics to ERDs.

The supposedly most popular system on the commercial market for reverse engineering of database schemas is the Bachmann toolkit [BACH88]. It completely avoids user interaction during reverse mapping. The trade-off is mapping errors due to missing information.

Meta models have been proposed mainly for CASE environments. GraphOR [MOGN91] is a tool built around a meta model of heterogeneous data and process models. The mapping between the different languages is done by so-called meta generator based on syntax definitions. The goal of our approach is less ambitious but we want to provide a more declarative description of the mapping.

[BATI92a] present an approach to represent heterogeneous schemas as ERDs, data flow diagrams, Pert diagrams and others. Properties of transformations *within* a fixed data model are described. Though the data models are organized in a simple meta model, this fact is not exploited for the reasoning.

[JANN92] designed a meta model hierarchy for relating different languages for requirements engineering. He concentrates on the mapping between ERD and SA (Structured Analysis). We re-use the top of this meta model for our purpose and the idea of navigating in the meta model to find suitable target constructs of the mapping. Different from [JANN92], we use a deductive query language to classify an input database schema. That idea can also be found in [ROSE92] for the purpose of classifying documents in a software repository. Our approach is more generic than [ROSE92] because it classifies over two instantiation levels.

The integration of heterogeneous databases is addressed in [KKM93] by representing relational and conceptual data models in an object-oriented framework. The representation language is similar to the one we propose. However, the purpose is the integration of existing database schemata for allowing global queries. In our approach, we are not concerned with discrepancies of different database schemata. Instead, the same schema has to represented in different data models.

[ASSE92] presents meta models for translating relational schemas into conceptual schemas. There are different meta models for the RDSs and for conceptual data models. The integration is performed within Telos and ConceptBase (like we do). We extend their approach by providing the classification of database schemas by means of declarative queries.

Finally, the IRDS standard [IRDS90] has to be mentioned. It proposes a framework for describing data models, database schemas, and database instances. The

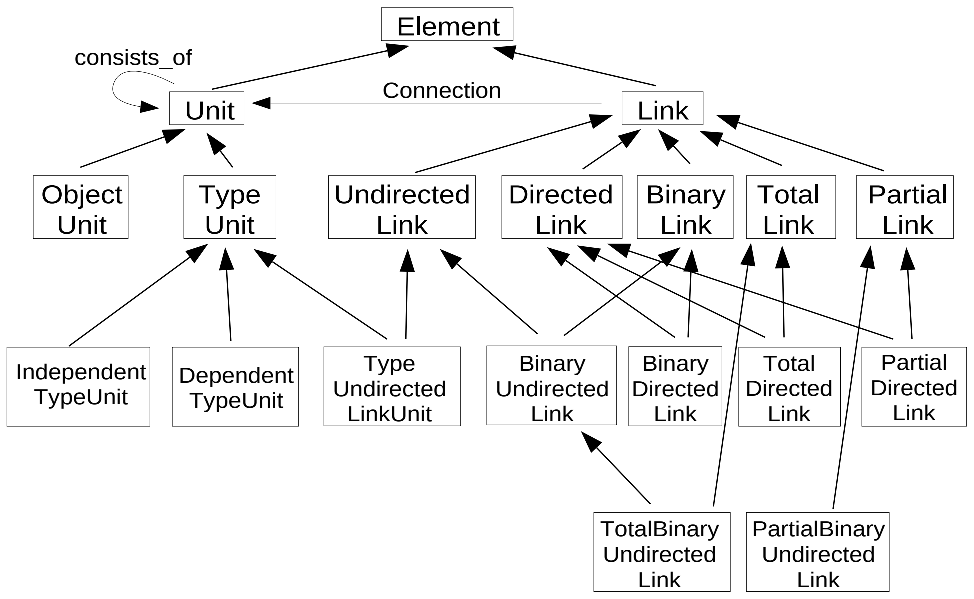


Figure 1: Meta model hierarchy

top three layers of IRDS are used in this paper when mapping from one data model into the other.

3 The Meta Model of Data Models

The term meta model has the flavor of ambiguity and unclear semantics. Here, we understand a meta model as a hierarchy of concepts which classify the building blocks of a data model. Figure 1 shows the specialization hierarchy of concepts that we use for classifying the constructs of a data model. The most general thing is an element. Then, units (e.g., entity sets in an ERD or relations in a RDS) are distinguished from links (e.g., relationship sets in an ERD or foreign keys in a RDS). By classifying a construct of a data model as a link we specify that this construct has a linkage character. The attribute `Connection` is used to describe the units participating in the linkage.

Links are separated with respect to their arity and direction. It should be noted that some concepts are defined as specialization of several other concepts. For example, a binary directed link is a directed link and a binary link. Some combinations are inconsistent, e.g., directed undirected link. Units may consist of other units. The two subconcepts, object unit and type unit, are used to distinguish units which may have instances in the database, e.g. tuples are instances of a relation, and units without explicit instances, e.g. domains of an attribute.

The meta model shown in Figure 1 reflects the expressiveness needed for our two example data models (ERDs and RDSs). It can be further refined to capture new concepts. Any data model participating in a mapping (either as source or as target) has to be classified into the hierarchy. Then, a schema can be mapped from one data model to the other by the following steps:

1. Represent the DB schema in the language of the source data model. This is a simple task provided the source data model is well-defined. For example, a statement `(Emp in Relation)` represents that `Emp` is a relation. For many data models this task can be automized by a program parsing the input DB

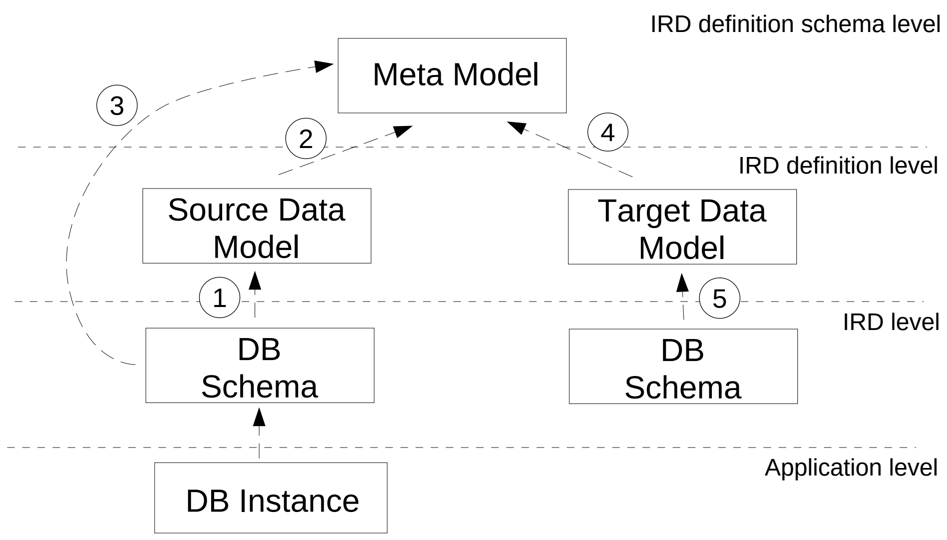


Figure 2: Mapping process via meta model

schema and representing it as instance of the source data model (see Section 4.1).

2. Classify the constructs of the source and target data models into the meta model. This step is done before the actual mapping and only once for each data model. For example, the statement `(Relation in TypeUnit)` represents that relations in the relational data model are type units, possibly in one of the subclasses of `TypeUnit`.

3. Classify the constructs of the input DB schema into the meta model. For example, `(Emp [in] IndependentTypeUnit)` expresses that `Emp` is a unit and a type which exists independently from other units. This is elaborated in Section 4.2.

4. Find a neighbor construct in the target data model. The simplest situation is when both constructs are instances of the same meta model concept. If not, one has to navigate in the meta model. Going up the hierarchy means forgetting details, going down means including new details. As an example, one may find that `(RelationshipSet in TypeUnit)` holds, i.e., a relationship set is a candidate for a relation to be mapped into. See Section 4.3 for more details.

5. As soon as the target data model construct has been found the downward instantiation into this data model is triggered. For example, one states that `(Emp in RelationShipSet)` holds. This step is not elaborated in this paper.

Figure 2 presents the steps in the diction of the IRDS standard. The database instance (not used in our approach) is a formal instance of a DB schema. This is an instance (dotted arrows) of the source target model, e.g., the relational data model. The source target model is an instance of the meta model. The mapping first follows the instantiation links upwards to classify the input DB schema into the source data model. Then, by interrogating the meta model, we find candidate target data model elements and follow the instantiation links downward to the output DB schema.

```
(Relation in TypeUnit)
(Attribute in DependentTypeUnit)
(CandidateKey in DependentTypeUnit)
(PrimaryKey in DependentTypeUnit)
(ForeignKey in BinaryUndirectedLinkTypeUnit)
(Domain in ObjectUnit)

(Relation has/Connection Attribute)
(ForeignKey occurs_in/Connection Relation)
(ForeignKey isA Attribute)
(Attribute value/Connection Domain)
(Relation identified_by/Connection PrimaryKey)
(PrimaryKey isA CandidateKey)
(CandidateKey composite_of/consists_of CandidateKey)
(ForeignKey from_r/Connection Relation)
(ForeignKey to_r/Connection Relation)

forall r1,r2,pk,fk
  (r1 identified_by pk) and (fk occurs_in r2) and
  (fk '= pk) ==> (fk source/from_r r1)

forall r1,r2,pk,fk
  (r1 identified_by pk) and (fk occurs_in r2) and
  (fk '= pk) ==> (fk destination/to_r r1)

[...]
```

Figure 3: Classification of the relational data model

For the specification of the meta model, the data models, and the database schemas we use the knowledge representation language Telos [MBJK90]. It has the advantage of offering multiple instantiation hierarchies (Figure 2) within a single framework, and a powerful deductive component that allows to specify queries over all levels of mapping process. Furthermore, Telos has been completely axiomatized in Datalog [JEUS92] and implemented in ConceptBase [JARK94] which allows the prototyping of a generic mapping tool within a short period of time.

4 Reverse Engineering of Relational Schemas

Without going into the details of deductive knowledge representation language Telos, we use predicates `(X in Y)`, `(X isA Y)`, `(X m Y)` to express that X is an instance of Y, or X is a specialization of Y, or X has an attribute of category m and value Y. A variant `(X n/m Y)` names the predicate `(X m Y)` by n. A deductive rule

```
forall n  (X n/m Y) ==> (X m Y)
```

makes sure that the original form can always be used. Additionally, a "meta" predicate `(X [in] Y)` is defined to be an abbreaviation for the formula `exists Z (X in Z) and (Z in Y)`. It may well be that `(X [in] Y)` is derivable without knowing the value for the existantially quantified variable Z. We will see such a case later.

4.1 Classification of the Data Models

With these definitions, the relational data model is instantiated from the meta model as shown in Fig. 3. Two deductive rules are included which translate foreign key occurences into the diction of connections between elements (here: relations). The predicate `(k1 '= k2)` expresses that two candidate key occurences have the same set of attributes. Each key occurence has its own identity. Thus, the rules can attach links to the referenced relations to them. This link is classified into the categories `from_r` (for the relation containing the foreign key) or `to_r` (for the relation having the foreign key as primary key), respectively.

```
(EntitySet in IndependentTypeUnit)
(RelationshipSet in TypeUndirectedLinkUnit)
(E-R-Link in BinaryUndirectedLink)
(Partial-E-R-Link in PartialBinaryUndirectedLink)
(Total-E-R-Link in PartialBinaryUndirectedLink)
(E-R-Attribute in DependentTypeUnit)
(Identifier in DependentTypeUnit)
(WeakEntitySet in DependentTypeUnit)
(SuperType in IndependentTypeUnit)
(SubType in DependentTypeUnit)
(Generalization in DirectedLink)
(PartialGeneralization in PartialDirectedLink)
(TotalGeneralization in TotalDirectedLink)
(Role in ObjectUnit)
(Cardinality in ObjectUnit)

(EntitySet identified_by/Connection Identifier)
(EntitySet has/Connection E-R-Attribute)
(Identifier isA E-R-Attribute)
(RelationshipSet has/Connection E-R-Attribute)
(E-R-Link to_r/Connection RelationshipSet)
(E-R-Link to_e/Connection EntitySet)
(E-R-Link name/Connection Role)
(E-R-Link range/Connection Cardinality)
(WeakEntitySet isA EntitySet)
(SuperType isA EntitySet)
(SubType isA EntitySet)
(Generalization super/Connection SuperType)
(Generalization sub/Connection SubType)
(TotalGeneralization isA Generalization)
(PartialGeneralization isA Generalization)
(TotalE-R-Link isA E-R-Link)
(PartialE-R-Link isA E-R-Link)

[...]
```

Figure 4: Classification of the entity-relationship data model

Similarily, the target data model for ERDs is specified within Telos. Due to its richness, it is more complex than the representation of the relational data model. The complete specification including constraints specifying the semantics of cardinality can be found in [JOHN93].

The ERD language described in Fig. 4 also has total and partial generalizations between entity sets which are not present in the relational data model. Therefore, the reverse engineer will have to provide some extra information.

4.2 Classification of the Input Database Schema

Having the above definitions, the classification of the elements of the input schema is done by so-called query classes [STAU94]. Query classes are frame-like notations of deductive rules which deduce membership to the query class. The classification of any schema element `e` into the meta model is expressed by a statement `(e [in] MMC)` where `e` is an element of a DB schema and `MMC` is a class in the meta model hierachy. Note that the label `[in]` states an instantiation of `e` into the meta model class `MMC` across the intermediate IRD definition level.

The set of solutions `MMC` for a given element `e` will never be empty since the meta model class `Element` always applies. In general there will be more than one solution. We discuss later which should one be chosen for the mapping. The following query definitions contain the deductive rules for deriving the classifification of elements into the meta model.

Figure 5 shows that the classification of elements like binary links must bridge the intermediate IRD definition level, i.e. the specific data model in which the binary link is encoded.

The query class definition of `BinaryLink` accomplishes the bridging by quantifying over the intermediate level (variable `blc` in Fig. 6). The condition states that the binary link class `blc` must have two distinct connection attributes `from`

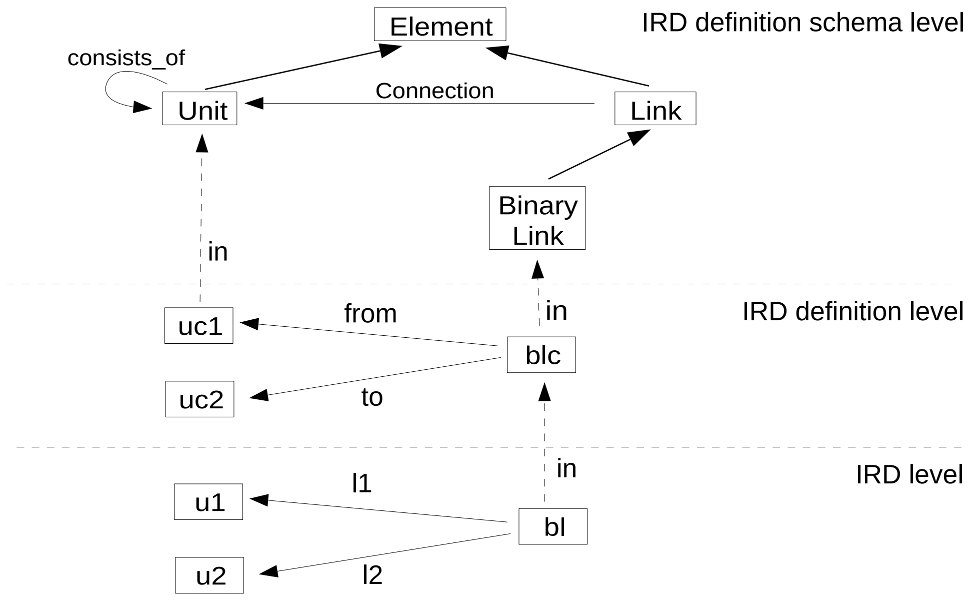


Figure 5: Classification of binary links

```
QueryClass BinaryLink isA Element with
  rule
    r: $ forall bl
          (exists blc,from,to,uc1,uc2
           (bl in blc) and (blc in Link) and
           (blc from/Connection uc1) and
           (blc to/Connection uc2) and
           (from \= to) and
            forall l1,u1
              (bl l1/from u1) and
              (not exists k1,v1
                (bl k1/from v1) and
                (l1 \= k1) ) and
            forall l2,u2
              (bl l2/to u2) and
              (not exists k2,v2
                (bl k2/to v2) and
                (l2 \= k2) ))
          ==> (bl [in] BinaryLink) $
end
```

Figure 6: Binary links defined in the meta model

and `to` to units. Any instance like `bl` of `blc` may fill these attributes at most once. Note that the condition quantifies over the possible names `from` and `to`. It is not important how they are named at the IRD definition level. It is only important that two connections can be distinguished for binary links.

Figure 7 refines the class `BinaryLink` for the two cases of directed and undirected links. Directed binary links can only be followed in one direction, i.e., a binary directed link `bl` which points by its `from` (or `to`) role `l` to some unit may not point to the same unit by the "inverse" `to` (or `from`, resp.) role. In other words: the source and destination of a binary link can be distinguished. Binary undirected links are defined as complementary concepts. Note that both classes are subclasses of `BinaryLink` and inherit its membership condition.

4.3 Determination of Target Data Model Concepts

As mentioned before, for a given input DB schema element `tt e` here can be more than one meta model class `MMC` for which `(e [in] MMC` holds. The definition of the

```
QueryClass BinaryDirectedLink isA BinaryLink with
  rule
    r: $ forall bl
          ( (bl [in] BinaryLink) and
            (forall l,u,from,to
              (bl l/from u) and (from \= to)
                ==> not (bl l/to u) ))
           ==> (bl [in] BinaryDirectedLink) $
end

QueryClass BinaryUndirectedLink isA BinaryLink with
  rule
    r: $ forall bl
          ( (bl [in] BinaryLink) and
            (forall l,u,from,to
              (bl l/from u) and (from \= to)
                ==> (bl l/to u) ))
           ==> (bl [in] BinaryUndirectedLink) $
end
```

Figure 7: Directed and undirected binary links in the meta model

meta model makes sure that all such `MMC` stand in a subclass relation[1]. Consequently, the uniquely defined most specific class `MMCo` represents the classification of the input DB schema element `e`.

The choice of the target data model concept for `e` is non-deterministic. Several cases are possible:

- The class `MMCo` has exactly one direct instance `T` in the target data model[2]. Then this instance shall be chosen as type for representing `e` in the output DB schema.
- The class `MMCo` has more than one direct instance in the target data model. Then, the user has to make a choice since all candidates apply equally well from the meta model view.
- The class `MMCo` has no direct instance in the target data model but super and/or subclasses do have such instances. Then the user has to decide wether he can provide additional information not encoded in the input DB schema which makes one of the subclasses applicable. Otherwise, a more generic concept has to be chosen (resulting in information loss!).

The result of this step is a candidate target data model type `T` for each input DB schema element `e`. To complete the transformation, one has to encode `e` as an instance of `T` (not shown in this paper).

The total specification of the meta model and the two example data models (ERD model and RDS model) take about 50 class and query definitions in Telos.

5 Implementation and Case Study

The prototype has been developed within ConceptBase. It is able to find appropriate target data model constructs for the elements of an input database schema in the source data model. The input database schema is described as an instance of the source data model, here: the relational data model. Since Telos allows the representation of all four layers of Figure 2, the process of finding the appropriate target constructs is almost completely defined by query classes. The "almost" refers to the missing information, e.g., whether a generalization is partial or total. As soon

[1]This is a consequence of the complementary defition of subclasses in the meta model. For example, binary directed links and binary undirected links are disjoint wrt. membership predicate `[in]`.

[2]Direct instance of a class are those object which are not instance of any subclass of this class.

```
class BinaryLink:Element
      {
      class_pointer   from_connection;  //source
      class_pointer   to_connection;    //target
public
      BinaryLink();
      ~BinaryLink();
      void set( class_pointer from, class_pointer to);//setting
      void print( identifier id);                   //printing
      class_pointer value( identifier);             //return values
      };
```

Figure 8: C++ code generated for query class BinaryLink

as the query classes for the meta model are encoded and the source and target data models are instantiated from the meta model as presented in the previous section, the prototype based on ConceptBase is ready for use.

After validating the prototype implementation, we have mapped the meta model and the query classes to C++ and integrated it into the commercial database CASE environment ProMod-PLUS. This environment provides a view on schemas of relational databases including foreign key dependencies (crucial for detecting generalizations). The C++-based mapping assistant called ProFace/Reverse is specialized for the mapping from RDS to ERD. Its output conforms to the syntax understood by the ERD editor of ProMod-PLUS. Since the environment already contains a forward mapping tool (from ERD to RDS) the cycle is closed, now.

Figure 8 presents the public part for the C++ class generated from the meta model concept `BinaryLink`. It provides test procedures for deciding wether an element of the input DB schema qualifies or qualifies not for this concept. In principle, the code generation can be automatic though we haven't yet implemented the generator.

The tool has been tested with database schemas of real applications. In one example, the relational schema contained 423 relations. The result of the reverse engineering produced 289 entity sets. It was identical to the original ERD of the application with the following exceptions:

1. The original role names between relationships and entities were replaced by system generated names. The reason is that role names are lost during forward engineering.

2. Four generalizations were detected which were not contained in the original ERD. The original ERD was incomplete in this respect.

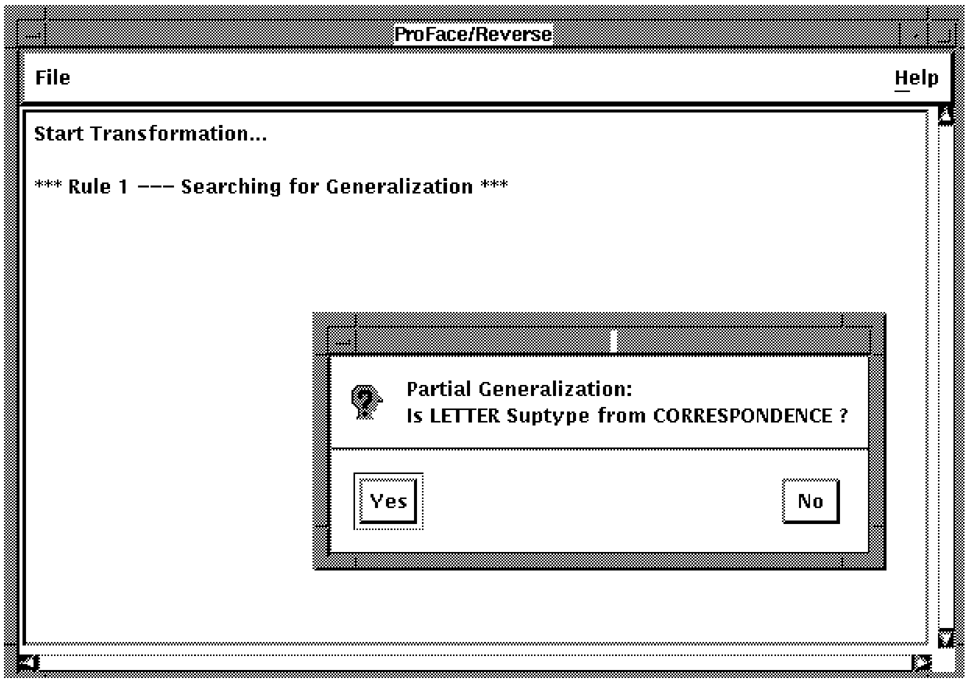
3. The mapping to many-to-many relationships of weak entities in the ERD is not yet supported by the tool.

4. Cardinalities other than "one" or "many", e.g. "2:5", are not supported by the tool because they are not derivable from a relational DB schema.

5. Some names of relationships are system-generated, esp. names of ISA relationships.

Of course, the resulting ERD depends on user decisions, esp. on partial and total generalizations. Wrong input from the user induces wrong results. In the the above example, only 38 yes/no decisions were necessary to build the ERD.

Figure 9 shows an intermediate situation in the mapping. The analysis of the input DB schema has exhibited a binary undirected link between a dependent and an independent type unit – represented by a primary key occuring in the two relations of the input DB schema which is also a foreign key between the two relations. Such situations can indicate that the mappings of the two type units stand in a partial



Start Transformation...

*** Rule 1 --- Searching for Generalization ***

Partial Generalization:
Is LETTER Suptype from CORRESPONDENCE ?

Figure 9: Retrieving missing information from the user

generalization relationship in the target data model (ERD) or that the two units stand in a normal relationship. Since the choice is not unique (see Section 4.3) the user is asked to supply the missing information.

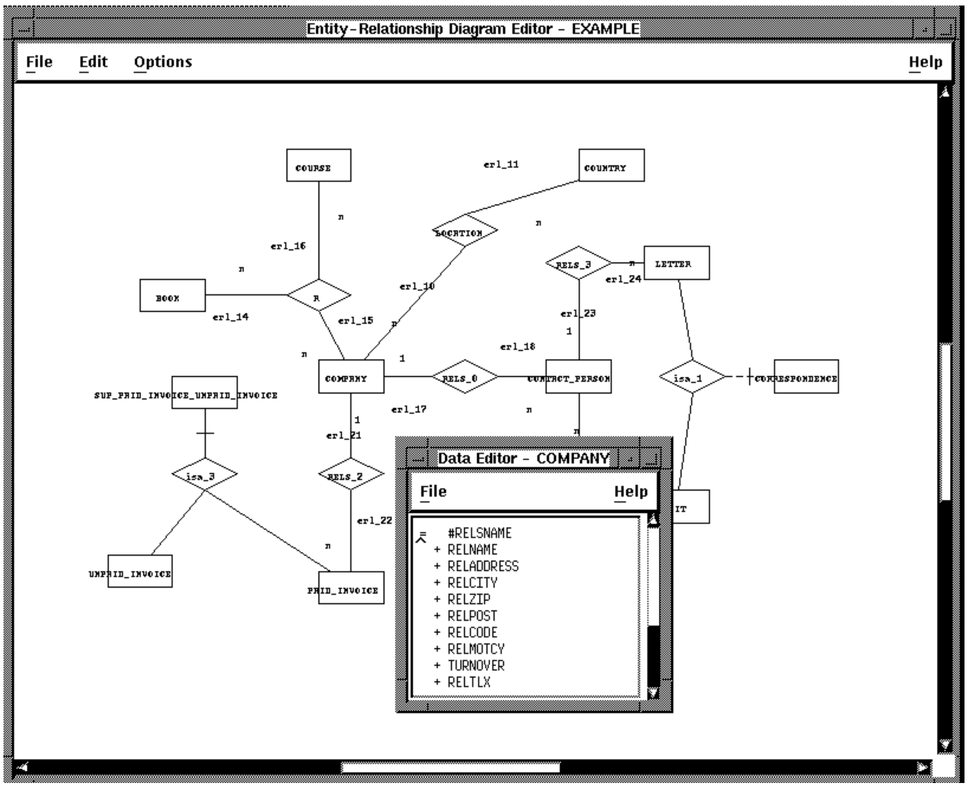
Figure 10 shows the result of the mapping of the first case study. Except for the points mentioned before it is identical to the original ERD schema from which the input RDS schema was generated. ProFace/Reverse uses the exchange data structures of ProMod-PLUS. Hence, the result of the reverse engineering can be evolved by the ERD editor of ProMod-PLUS and then mapped to the RDS schema.

The source of another case study was a relational data model with 80 relations and 408 attributes. There are 107 foreign key dependencies to the 80 primary keys. The schema was a small one, but a very complex one. During the transformation process back to a entity-relationship model there are 205 (!) interactive user decisions needed to decide if there is a generalization (partial or total) or not. The resulting entity-relationship diagram confirms the advantage of the entity-relationship model over the relational model for managing complexity. The generalizations became obvious after the transformation. Some relationships were not binary. This is a hint that such relationships are meaningful for conceptual modeling.

A final case study was on the meta model approach. Due to company decisions, the target data model had to be changed from ERD to a data model from object-oriented analysis. The modification took only about a week to be operational. The ease of the modification validates the statement that the detour via the meta model pays on the long run.

6 Summary and Outlook

This paper argues for the meta model approach for transforming database schemas into each other. Though meta models are not a new idea, the use of a deductive, object-oriented specification language, Telos, is new. This language makes auto-



```
#RELSNAME
+ RELNAME
+ RELADDRESS
+ RELCITY
+ RELZIP
+ RELPOST
+ RELCODE
+ RELMOTCY
+ TURNOVER
+ RELTLX
```

Figure 10: Result of the reverse engineering

matic execution of the specification possible because it is restricted to query classes as the means for describing the mapping. The main trick is the use of formulas quantifying over the intermediate data model layer. Thereby, the classification into the meta model is described entirely independent from the involved data models.

Since no specific assumptions wether a data model is used as source or target in a mapping, the method can well be applied to forward engineering, for example from ERD to RDS. So, at the end the title of the paper is justified. We did not investigate the forward engineering case because it is much simpler than reverse engineering. Moreover, the ProMod-PLUS tool already supplied such a facility.

The approach has been validated in realistic case studies. A number of research issues remain open.

- The meta model basically reflects the elements found in RDSs and ERDs. Whether object-oriented data models easily fit into the meta model has to be investigated. At least the behavioral part (methods) are completely new. Potentially more important, hierarchical and network data models should also fit into the meta model.

- The user interaction for supplying missing information should be integrated into the meta model. In the prototype based on ConceptBase, we assume that the necessary information is added to the input schema before the mapping (i.e., query evaluation) starts. A process model could formalize this and take some optimality criteria into account.

- It should be possible to generate the C++ code automatically from the Telos specification.

- The database instance is not taken into account. By querying it, one can validate or falsify assumptions about generalization not deducible from the schema. Additionally, re-engineering of database instances should be supported by the method, i.e., the generation of code that evolves the database instance whenever the conceptual schema is evolved.
- The granularity of mapping are atomic elements of the DB schemas. It is likely that some DB schemas require to map complex portions in a single complex step.

An attractive property of the uniform representation of the meta model and the data models within Telos is the possibility to reason about both. For example, the concepts of the meta model can be checked for satisfiability [BUCH94]. Furthermore, the data models can be queried hypothetically for possible mappings, i.e., without taking an example schema into account.

The C++-version of our reverse engineering tool is scheduled to be part of the next release of ProMod-PLUS.

Acknowledgements. We would like to thank Matthias Hallmann and Wolf Fischer from CAP debis SSP for helping us to evaluate our prototype within a commercial system. Many thanks to Matthias Jarke for his constant support.

References

[ASSE92] Assenova P., "Concept formation by reverse modelling", *Esprit NATURE Report Series*, 92-10, RWTH Aachen, 1992.

[BACH88] Bachmann C., "A CASE for reverse engineering", *Datamation*, July 1988.

[BATI92] Batini C., Ceri S., Navathe S., *Conceptual design - an entity-relationship approach*, Benjamin-Cummings, Redwood City, CA, 1992.

[BATI92a] Batini C., Di Battista G., Santucci G., "A formal framework for multi-level schema documentation in a data dictionary", In Falkenberg et al. (ed.): *Information Systems Concepts - Improving the Understanding*, Elsevier Science Publ. 1992.

[BUCH94] Buchheit M., Jeusfeld M., Nutt W., Staudt M., "Subsumption between queries to object-oriented databases." In *Information Systems*, 19, 1, pp. 33–54, 1994.

[CASA83] Casanova M.A., de Sa J.E.A., "Designing entity-relationship schemas for conventional information systems", *Proc. 3rd Intl. Conf. on Entity-Relationship Approach*, North-Holland, 1983.

[DAVI88] Davis K.H., Arora S.K., "Converting a relational database model into an entity relationship model", *Proc. 6th Intl. Conf. on Entity-Relationship Approach*, North-Holland, 1988.

[DUMP83] Dumpala S.R., Arora S.K., "Schema translation using the entity-relationship approach", In Chen P. (ed.): *Entity Relationship Approach to Information Modeling and Analysis*, Elsevier Science Publ., Amsterdam, 1983.

[IRDS90] ISO/IEC 10027, *Information technology – information resource dictionary system (IRDS) – framework*, ISO/IEC International Standard, 1990.

[JANN92] Janning T., *Integration of languages and tools for requirements engineering and programming-in-the-large* (in German), Dissertation, RWTH Aachen, 1992.

[KALM91] Kalman K., "Implementation and critique of an algorithm which maps a relational database to a conceptual model", *Proc. 3rd Intl. Conf. CAiSE'91*, *LNCS*, 498, Springer-Verlag, 1991.

[JARK94] Jarke M., Gallersdörfer R., Jeusfeld M.A., Staudt M., Eherer S., "ConceptBase - a deductive object base for meta data management", appears in *Journal on Intelligent Information Systems*, Special Issue on Advances in Deductive Object-Oriented Databases, 1994.

[JEUS92] Jeusfeld M., *Update control in deductive object bases* (in German), Infix-Verlag, St.Augustin, Germany.

[JOHN93] Johnen U.A., *A re-engineering approach for database modelling* (in German), diploma thesis, RWTH Aachen, 1993.

[KKM93] Keim D.A., Kriegel H.-P., Miethsam A., "Integration of relational databases in a multidatabase system based on schema enrichment", In *Proc. 3rd Intl. Workshop on Interoerability in Multidatabase System (RIDE-IMS)*, Vienna, Austria, 1993.

[MARK92] Markowitz V.M., Shoshani A., "Representing extended entity-relationship structures in relational databases - a modular approach", *ACM Trans. on Database Systems*, 17, 3, Sept. 1992.

[MOGN91] Morejon J., Oudrhiri R., de Gaudemont M., Negros P., "GraphOr – a meta design tool", In Kangassalo H (ed.): *Entity-Relationship Approach – The Core of Conceptual Modelling (Proc. ER'90)*, North-Holland, 1991.

[MBJK90] Mylopoulos J., Borgida A., Jarke M., Koubarakis M., "Telos – a language for representing knowledge about information systems", In *ACM Trans. Information Systems*, 8, 4, pp. 325–362, 1990.

[NAVA88] Navathe S.B., Awong A.M., "Abstracting relational and hierachical data with a semantic data model", *Proc. 6th Intl. Conf. on Entity-Relationship Approach*, North-Holland, 1988.

[ROSE92] Rose T., Jarke M., Mylopoulos J., "Organizing software repositories – modeling requirements and implementation experiences", In *Proc. 16th Intl. Computer Software & Applications Conf.*, Chicago, IL, Sept. 23-25, 1992.

[STAU94] Staudt M., Nissen H.W., Jeusfeld M.A., "Query by class, rule and concept", Appears in *Applied Intelligence*, Special Issue on Knowledge Base Management, 1994.