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

Manfred A. Jeusfeld
University of Science & Technology, ISMT, Hong Kong
jeusfeld@usthk.ust.hk

and

Uwe A. Johnen
debis Systemhaus - Software Tools, 52076 Aachen, Germany
johnen@gei-aachen.de

Received May 11, 1994
Revised February 28, 1995

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 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 reformulates the source and target data models into a so-called meta model. The second part classifies the input schema into the meta model, yielding a data model independent representation. The third part synthesizes the output schema in terms of the target data model. A prototype has been implemented on top of deductive object base manager ConceptBase for the mapping of relational schemas to entity-relationship diagrams. From this, a C++-based tool has been derived as part of a commercial CASE environment for database applications.

Keywords: database design, reverse engineering, meta model, IRDS, Datalog

1. Introduction

Re-engineering [8] is investigation, analysis, and evolution of an existing system in order to create a new version. Applied to database schemas it re-engineering translates to methods for restructuring of the schema elements and possibly database objects. Database schemas appear in different representation languages depending on the task to perform on them. Conceptual database schemas (example: an entity-relationship-diagram) are used in the design phase and are usually represented as graphs. Logical database schemas (example: a relational schema) implement the conceptual schemas. They describe data in a way that is suitable for query languages and for interfacing to programming language type systems. For re-engineering purposes, the conceptual schema is crucial for understanding the meaning of a database. In many cases it

*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.

is missing and has to be reverse-engineered from the logical schema. Then, the restructuring and forward-engineering to the new logical schema can take place. If no information losses would occur then forward- and reverse-engineering would be inverse operations. However, the logical schema typically contains less information than the conceptual schema is has been produced from. Therefore, the reverse-engineering is particularily hard to solve.

This paper presents an approach that is suitable for both reverse- and forward-engineering of database schemas. The idea is to create an abstraction level, the *meta model*, on top of the heterogeneous representation languages for database schemas. The concepts of the meta model deliver the key words for formulating queries, rules on constraints on input/output schemas. Thereby it supports actively the creation of a mapping relationship between them.

The upcoming new data models, like object-oriented ones, make the need for a flexible mapping environment obvious. A considerable amount of concepts are similar among data models. Knowledge about them should be kept in a representation that makes reuse (of mapping rules) simple. For example, there are about a dozen dialects of entity-relationship data model. We claim that this 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. It is in fact a language for describing data models. The mapping from and to a new data model can be derived from its classification into the meta model – modulo information loss.

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 relevant for database schemas 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 method has been prototype with the ConceptBase system[14]. A full-scale implementation was carried out in C++ and integrated into the commercial database CASE environment ProMod-PLUS from debis Systemhaus. Fionally, section 5 proposes extension of the approach to query database instances.

2. Related Work

Early contributions to the reverse engineering of relational schemata were made by Casanova and de Sa [6] and by Dumpala and Arora [10]. Both present procedural algorithms that generate a simplified ERD from a RDS. Davis and Arora [9] extended the method of [6] 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 by Batini et al. [4] and Navathe and Awong [21]. The latter chose an ERD language with generalization hierarchies. Kalman [13] 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.

Markowitz and Shoshani [18] use 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.

Petit et al. [22] and Andersson [1] propose to extract schema knowledge from SQL query definitions and from a snapshot database instance. Certain join conditions in queries establish links between relations. Inclusion of key sets of two relation instances indicates generalization relationships.

Premerlani and Blaha [23] differ from the above approaches by mapping to OMT (object-modeling technique) notation instead of ER diagrams. Special attention is devoted to candidate keys (opposed to primary keys) and to 'bad' habits in practice like special constants as null values. Investigation of application programs and the database instance is also included in their method.

The supposedly most popular system on the commercial market for reverse engineering of database schemas is the Bachmann toolkit [3]. 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 [19] 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.

Batini et al. [5] 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.

Janning [12] 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 the approach in [12], we use a deductive query language to classify an input database schema. That idea can also be found in Rose et al. [24] for the purpose of classifying documents in a software repository. Our approach is more generic because it classifies over two (even three) instantiation levels.

The integration of heterogeneous databases is addressed by Keim et al. [17] via 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.

Assenova [2] 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 O-Telos and ConceptBase (like we do). We extend their approach by providing the classification of database schemas by means of declarative queries.

3. The Meta Model of Data Models

In the framework of the IRDS standard [11], a database schema is a collection of statements constraining all possible database instances. The language for expressing these statement, the data model, is itself a collection of statements constraining all possible database schemata (belonging to this specific data model). At the top level of abstraction, we have the meta model of data models which defines how data models can be described.

3.1. Notations and Representation Language

The standard introduces so-called level-pairs to formalize the relationship between two neighbor levels by SQL und Pascal statements. We use a formal logic representation based on O-Telos [15], a Datalog-based variant of Telos [20]. The predicate `(x in c)` of O-Telos represents the level-pair relationship "x is an instance of of the concept c" of IRDS. In an abstract way, it is sufficient to formalize the idea of IRDS: Let `x` be an element of a database, `c` an element of an database schema, `DM` an element of a data model, and `MMC` a concept of the meta model. Then the four abstraction hierarchies of IRDS melt down to the statement:

```
x in c, c in DM, DM in MMC
```

Two more predicates are used in the logical representation language. Specialization between concepts is expressed by `(c isA d)` where `c` is the more specific concept (sometimes called *subclass*). The predicate `(x m/n y)` states a *reference* from `x` to `y`. The reference has the *label* `n` and the *category* `m`[1]. The label `n` is used to distinguish references which agree on source and destination components, for example `(peter attribute/boss mary)` and `(peter attribute/friend mary)` The category `m` is supposed to occur as label of a reference between the classes of `x` and `y` (see IC1 in fig. 1).

The three predicates are sufficient to encode the factual knowledge about data, schemas, and data models. The rest is expressed via logical formulas, interpreted either as deductive rules or integrity constraints over the factual knowledge. The set of all facts and logical formulas is called *meta database*[2].

Figure 1 lists the predefined content of the meta database. Rules R1, R2 and integrity constraints IC1, IC2 specify the semantics of the `isA` predicate, an abbreviation for the relationship predicate, and a typing condition for references between

[1] The placeholder `m` may not take values `in` or `isA`.

[2] Any deductive database system implementing Datalog with negation [7] can be used to manage and query such a meta database. We chose ConceptBase [14] because it is optimized for the predicates and formulas used in this paper.

```
Object in Object, Object references/references Object

R1: forall c,d,x  (x in c) and (c isA d) ==> (x in d)
R2: forall x,y,n,m (x m/n y) ==> (x m y)
R3: forall x,c,mc  (x in c) and (c in mc) ==> (x [in] mc)
R4: forall x,y,m,n,c,d
      (x in c) and (c m/n d) and (x n y) ==> (x [m] y)
IC1: forall x,m,y (x m y) ==>
       (forall c,d,k (c k/m d) ==> (x in c) and (y in d))
IC2: forall x,m,y (x m y) ==> exists c,d,k (c k/m d)
```

Figure 1: Predefined facts, rules and constraints

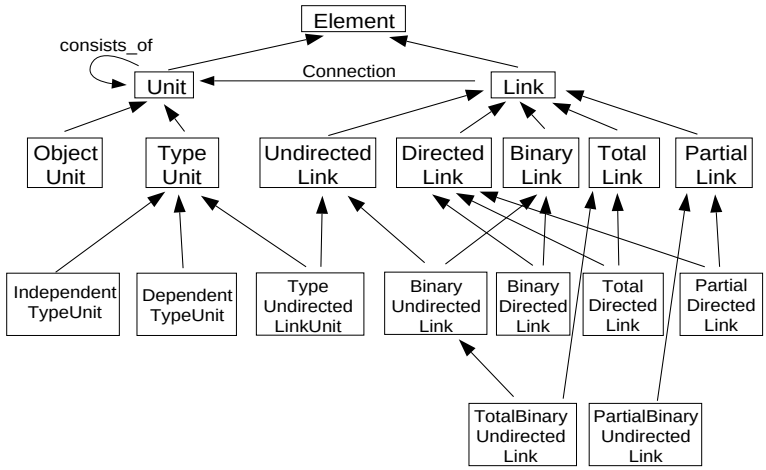

Figure 2: Meta model hierarchy

objects. The meta instantiation predicate `(x [in] mc)` is used to express that `x` is instance of `some` class `c` which itself is instance of the meta class `mc`. The deductive rule R3 defines the relationship to predicate `(x in c)`. An analogous abbreviation for references between objects is introduced by rule R4.

We assume that predicate for testing equality of constants is predefined. The formal semantics of a O-Telos meta database is the uniquely defined perfect model computed by fixpoint operator with stratified negation [7].

3.2. The Meta Model

Figures 2 and 3 present the meta model as a specialization hierarchy of concepts 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

```
Unit in Object, Link in Object
Link references/Connection Unit, Unit references/consists_of Unit

Unit isA Element, Link isA Element, ObjectUnit isA Unit, TypeUnit isA Unit
UndirectedLink isA Link, DirectedLink isA Link, BinaryLink isA Link
TotalLink isA Link, PartialLink isA Link, IndependentTypeUnit isA TypeUnit
DependentTypeUnit isA TypeUnit, TypeUndirectedLinkUnit isA TypeUnit
TypeUndirectedLinkUnit isA UndirectedLink
BinaryUndirectedLink isA UndirectedLink
BinaryUndirectedLink isA BinaryLink, BinaryDirectedLink isA DirectedLink
BinaryDirectedLink isA BinaryLink, TotalDirectedLink isA DirectedLink
TotalDirectedLink isA TotalLink, PartialDirectedLink isA DirectedLink
PartialDirectedLink isA PartialLink
TotalBinaryUndirectedLink isA BinaryUndirectedLink
TotalBinaryUndirectedLink isA TotalLink
PartialBinaryUndirectedLink isA BinaryUndirectedLink
PartialBinaryUndirectedLink isA PartialLink
```

Figure 3: Factual representation of the meta model

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 2 reflects the expressiveness needed for our two example data models (ERDs and RDSs). It can be extended to capture more 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 principle 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 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.

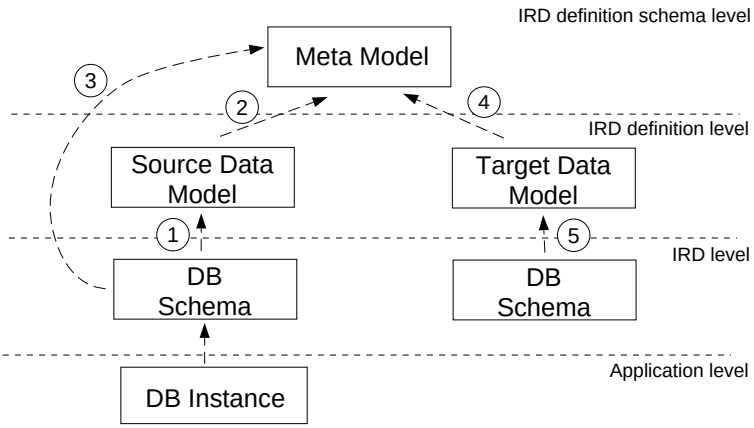

Figure 4: Mapping process via meta model

Figure 4 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.

4. Reverse Engineering of Relational Schemas

The meta model is a O-Telos meta database of statements about possible data models. In this section the meta database is extended by the representation of two instances of the meta model, the relational data model and the entity-relationship model. The representation translates the definition of the data models into the diction of the meta model, esp. by deriving connections from foreign key dependencies. After that rules define in terms of the meta model which schema element (IRD level in fig. 4) classifies into which concept of the meta model (IRD definition schema level). The classification is expressed *across* the data model level (IRD definition level) by means of logical quantification. Schema elements are accessible since they are represented as formal instances of concepts in their data model.

4.1. Classification of the Data Models

The relational data model is instantiated from the meta model as shown in fig. 5. The classification of `Relation` shows the two faces of relations: on the one hand the can express (type) units, on the other hand they may encode link between units. Rule R6 identifies certain relations as links: if the primary key of a relation consists of foreign key occurrences then it establishes a connection (`linkTo`) to the relation to which the foreign key is referring to (`refToPK`). Each key occurence has its own identity. Rule R6 uses this feature to discriminate different links from the same relation. Those foreign keys which are not covered by rule R6 are automatically classified by rule R7 into `ForeignKeyLink`. They constitute links on their own right: rule R8 derives the

```
Relation in Link, Relation in TypeUnit
Attribute in DependentTypeUnit
AttributeSet in ObjectUnit
PrimaryKey in ObjectUnit
ForeignKey in ObjectUnit, Domain in ObjectUnit
ForeignKeyLink in Link

Relation consists_of/attr Attribute
Relation consists_of/hasForeignKey ForeignKey
ForeignKey isA AttributeSet, ForeignKeyLink isA ForeignKey
ForeignKey consists_of/refToPK PrimaryKey
Attribute consists_of/value Domain
Relation consists_of/identified_by PrimaryKey
PrimaryKey isA AttributeSet
AttributeSet consists_of/partAttr Attribute
Relation Connection/linkTo Relation
ForeignKeyLink Connection/from_r Relation
ForeignKeyLink Connection/to_r Relation

R5: forall k1,k2
       (k1 in AttributeSet) and (k2 in AttributeSet) and
       (forall a (k1 partAttr a) ==> (k2 partAttr a))
    ==> (k1 subsetOf k2)

R6: forall r1,r2,pk1,pk2,fk
      (r1 identified_by pk1) and (r2 hasForeignKey fk) and
      (fk refToPK pk1) and (r2 identified_by pk2) and
      (fk subsetOf pk2) and not (pk2 subsetOf fk)
    ==> (r2 linkTo/fk r1)

R7: forall r1,r2,fk  (fk in ForeignKey) and
      not (r1 linkTo/fk r2) ==> (fk in ForeignKeyLink)

R8: forall r1,r2,pk1,pk2,fk  (fk in ForeignKeyLink) and
      (r1 identified_by pk1) and (r2 hasForeignKey fk) and
      not (r2 linkTo/fk r1) and (fk refToPK pk1) and
      (r2 identified_by pk2) ==> (fk from_r/pk2 r2)

R9: forall r1,r2,pk1,fk  (fk in ForeignKeyLink) and
      (r1 identified_by pk1) and (r2 hasForeignKey fk) and
      not (r2 linkTo/fk r1) and (fk refToPK pk1)
    ==> (fk to_r/pk1 r1)

R10: forall r,pk,fk  (fk in ForeignKeyLink) and
       (r identified_by pk) and (r hasForeignKey fk) and
       (pk subsetOf fk) and (fk subsetOf pk)
     ==> (r [in] DependentTypeUnit)
```

Figure 5: Classification of the relational data model

source of the link and R9 its destination(s). The connections are 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. The association of the foreign key link to the meta model is completed by the instantiation of the two categories to the `Connection` reference. The model makes heavy use of knowledge about foreign keys for translating the relational data model into the meta model dictions.

Foreign keys to candidate keys can be taken into account by extending the conditions of rules R6, R8, and R9 appropriately. Though common in practice [23] such foreign keys constitute a bad design because the key property of such keys is not guaranteed over lifetime of a database. If a foreign key is identical to the primary key of the relation, then the objects in this relation depend on the objects of the relation the foreign key is pointing to. Rule R10 classifies those relations as dependent type units in the sense of the meta model. Such relations are candidates for generalizations.

Similarily, the target data model for ERDs is specified within O-Telos. Due to its richer set of concepts, it requires no 'translation' rules (opposed to the relational

```
EntitySet in TypeUnit
RelationshipSet in TypeUndirectedLinkUnit
ER-Role in BinaryUndirectedLink
Partial-ER-Role in PartialBinaryUndirectedLink
Total-ER-Role in PartialBinaryUndirectedLink
ER-Attribute in DependentTypeUnit
WeakEntitySet in DependentTypeUnit
Generalization in DirectedLink
PartialGeneralization in PartialDirectedLink
TotalGeneralization in TotalDirectedLink
Cardinality in ObjectUnit, ER-Domain in ObjectUnit

EntitySet consists_of/e_attr ER-Attribute
Identifier isA ER-Attribute
RelationshipSet Connection/r_attr ER-Attribute
ER-Attribute consists_of/value ER-Domain
ER-Role Connection/to_r RelationshipSet
ER-Role Connection/to_e EntitySet
ER-Role Connection/range Cardinality
WeakEntitySet isA EntitySet
Generalization Connection/super EntitySet
Generalization Connection/sub EntitySet
TotalGeneralization isA Generalization
PartialGeneralization isA Generalization
TotalER-Role isA ER-Role
PartialER-Role isA ER-Role
```

Figure 6: Classification of the entity-relationship data model

data model). Otherwise, its role as target data model is purely pragmatic. Both the relational and the entity-relationship data model are instances of the one meta model. The complete specification including constraints specifying the semantics of cardinality can be found in [16]. The ERD language described in fig. 6 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 deductive rules derving meta instantiation relationships. 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 classification of elements into the meta model.

Figure 7 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. 8). The condition states that the binary link class `blc` must have two distinct connection attributes `from` 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

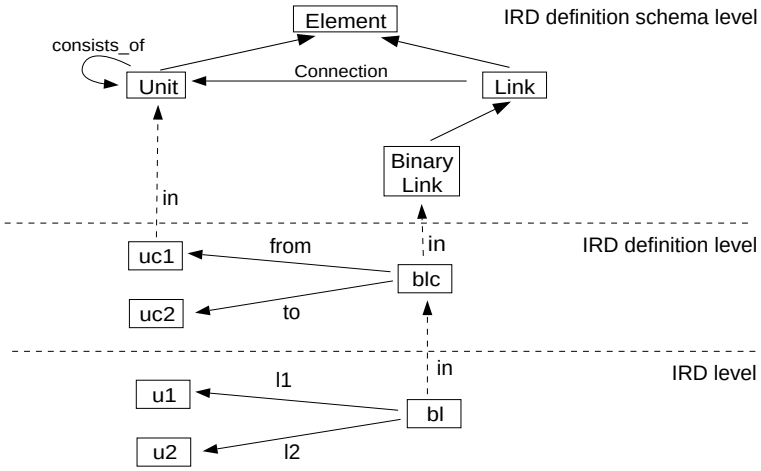

Figure 7: Classification of binary links

```
C1: forall bl,blc,from,to,uc1,uc2  (bl in blc) and
      (blc in Link) and (blc Connection/from uc1) and
      (blc Connection/to uc2) and not (from = to) and
      (forall l1,u1
         (bl from/l1 u1) and (not exists k1,v1
               (bl from/k1 v1) and not (l1 = k1) )) and
      (forall l2,u2
         (bl to/l2 u2) and (not exists k2,v2
               (bl to/k2 v2) and not (l2 = k2) ))
     ==> (bl [in] BinaryLink)
```

Figure 8: Binary links defined in the meta model

they are named at the IRD definition level. It is only important that two connections can be distinguished for binary links.

Figure 9 classifies those units that are also links connecting other units. The definition covers the relations treated by rule R6 in fig. 5. Note that a connection to a unit `u` is required. In principle, one can model a RDS where a relation is classified as type undirected link unit with connection to another type undirected link unit. That would be a role link between two relationship sets if the ER model wouldn't forbide such links. The meta model is less restrictive because of its task to cover more than one data model.

Dependent type units contain objects which necessarily depend on the existence of other objects in type units. For example, an budget account in a company may necessarily depend on the existence of a project. Figure 10 identifies those units by relying on a predicate `(x ?DEPENDSON y)`. This "oracle" predicate simulates knowledge acquired from the user. All type units that are not dependent are classified

```
C2: forall ul,lc,link,l,u  (ul [in] TypeUnit)
      (ul in lc) and (lc in Link) and
      (lc Connection/link u) and (ul link/l u)
    ==> (ul [in] TypeUndirectedLinkUnit) $
```

Figure 9: Detection of type undirected link units

```
C3: forall dtu  (dtu [in] TypeUnit) and
      (exists tu,link,r1,r2 (tu [in] TypeUnit) and
         (link r1 tu) and (link r2 dtu) and
         (dtu ?DEPENDSON tu))
    ==> (dtu [in] DependentTypeUnit)

C4: forall tu  (tu [in] TypeUnit) and
      not (tu [in] DependentTypeUnit)
    ==> (tu [in] IndependentTypeUnit)
```

Figure 10: Classification of type units by consulting oracle

```
SourceDataModelConcept references/mapTo TargetDataModelConcept

RDM-Element in SourceDataModelConcept
Relation isA RDM-Element, Attribute isA RDM-Element
...
ForeignKeyLink isA RDM-Element

ER-Element in TargetDataModelConcept
EntitySet isA ER-Element, RelationshipSet isA ER-Element
...
ER-Domain isA ER-Element

RDM-Element mapTo/generatedERelem ER-Element

MIC1: forall x  (x [in] SourceDataModelConcept)
      ==> exists y (x [mapTo] y))

MIC2: forall x,y,MMC  (y [in] TargetDataModelConcept) and
      (x [mapTo] y) and (x [in] MMC) and (MMC isA Element)
      ==> (y [in] MMC)
```

Figure 11: The mapping relationship between data models

into `IndependentTypeUnit`. Note that the two rules classify objects at the database schema level. The condition of the rule C3 without the oracle predicate is necessary for dependent type units. Pragmatically, it limits the search space for the oracle.

4.3. Synthesis of Output Database Schema

The previous subsections present a method to *analyze* the elements of an input database schema into the meta model by purely deductive rules. By construction, the result is uniquely defined by the perfect model semantics [7]. It contains statements of the form `(e [in] MMC)` where `e` is an input database element and `MMC` is a concept of the meta model.

The problem of generating the output database schema in terms of the target data model is different in nature: it is the *synthesis* of a set of statements which fulfill certain constraints. While the analysis is unique, the synthesis isn't. In general, there are infinitely many *suitable* output schemas (here: ER diagrams). Since there is not a unique solution, we present a set of constraints on all solutions and then add some preference order.

The statements in fig. 11 express the mapping relationships between source and target data model. They are used to express generic constraints on the mapping. These statements allow to quantify over input/output schema elements as well as over source and target data model concepts.

The constraint MIC1 ensures that each element in the input database schema finds its counterpart in the output database schema. The second constraint, MIC2,

demands that the output schema element has the same meta model class as the input schema element.

Additional constraints on the output database schema are imposed via the definition of the target data model. In particular, constraint IC1 of fig. 1 demands that the references are correctly typed. For example, the `to_r` reference of `ER-Role` must point to an instance of `RelationshipSet`.

Within this framework, the actual generation of the target database schema accounts to the insertion of statements of instances of the target data model. Constraint MCC1 serves as a bookkeeping 'watchdog': it can be used to count the number of elements that haven't yet been mapped. Constraint MIC2 guides the search for suitable target model concepts. The following rule will deliver all immediate candidates. The parameter `x` is the input database schema element and `c` is a concept of the target data model.

```
forall x,MMC
  (x [in] MMC) and (MMC isA Element) and
  (x [in] SourceDataModelConcept) and
  (c in MMC) and (c in TargetModelConcept)
==> (x CANDIDATE-CONCEPT c)
```

Note that `c` may be an instance of a more specific meta model class than `MMC`. In such case the user has to confirm that the element `x` fulfills the properties of the more specific class (see example in fig. 15). If more than one `c` appears as candidate concept then the one whose class is closest to `MMC` should be preferred. Note that the greater the distance to `MMC`, the greater will be the information loss in the forward-engineering direction.

The more difficult case is when no `c` appears as candidate key. Then one has navigate upwards in the meta model hierachy until the first solution pops up. Each step upwards means information loss. Formally, the result would violate MIC2. However, one has to tolerate this situation that arises from heterogeneity of the data models. One may try to find 'optimal' solutions which minimize the weighted of violations of MIC2 ranging over all mapping relationships. This is a hard combinatorial problem and not elaborated here.

The second group of constraints, the target data model specific ones, has to be fulfilled. Otherwise, the result would not be a legal instance of the target data model. In theory, this may exclude any solution. In our case, one can assume that there is at least one solution: the ERD from which the RDS was forward engineerd from.

4.4. A Small Example

Suppose there is a RDS with three relations `CORRESPONDENCE`, `LETTER` and `VISIT`. The primary key of all relations shall be the attribute `CONTACTID`. For `LETTER` and `VISIT` the primary key is also a foreign key occurence. This accounts to the representation in fig. 12. It is a formal instance of the class definitions in fig. 5.

The names `pkC` etc. for the primary and foreign keys have to be generated when extracting the schema from the data dictionary. The analysis of the above statements by the rules of fig. 5 yields that R6 is not applicable since no foreign key is a proper subset of a primary key. Thus, rules R7 to R10 derive the following statements:

```
fkL from_r/pkL LETTER, fkL to_r/pkC CORRESPONDENCE
```

```
CORRESPONDENCE in Relation
LETTER in Relation, VISIT in Relation
pkC in PrimaryKey, pkL in PrimaryKey, pkV ikn PrimaryKey
CONTACTID in Attribute, pkC partAttr/a1 CONTACTID
pkL partAttr/a1 CONTACTID, pkV partAttr/a1 CONTACTID
CORRESPONDENCE identified_by/k pkC
LETTER identified_by/k pkL, VISIT identified_by/k pkV
CORRESPONDENCE attr/a1 CONTACTID
LETTER attr/a1 CONTACTID, VISIT attr/a1 CONTACTID

fkL in ForeignKey, fkV in ForeignKey
fkL partAttr/a1 CONTACTID, fkV partAttr/a2 CONTACTID
LETTER hasforeignKey/k1 fkL, VISIT hasforeignKey/k1 fkV
```

Figure 12: Example of a relational schema

```
fkV from_r/pkV VISIT, fkV to_r/pkC CORRESPONDENCE
LETTER [in] DependentTypeUnit, VISIT [in] DependentTypeUnit
```

No links to `LETTER` or `VISIT` are established since they classify as dependent type units. Next the rule R3 and C1 applies and yields:

```
CORRESPONDENCE [in] TypeUnit
fkL [in] BinaryLink, fkV [in] BinaryLink
```

The classification of the entity-relationship model into the meta model exposes `EntitySet` as target data model concept for `CORRESPONDENCE`, and `WeakEntitySet` for `LETTER` and `VISIT`. It remains to decide how to map the binary links. There is no direct instance of `BinaryLink` in the entity-relationship model. The designer may decide that the binary link is directed, and that `Generalization` is a correct target model concept. Another possibility is to select `RelationshipSet`, an undirected link that is a also a type unit. The alternative `ER-Role` does not apply since none of the relations was mapped to `RelationshipSet`.

5. Implementation and Case Study

A prototype of the reverse engineering has been developed with ConceptBase. The O-Telos definitions of the meta model, the two data models and the RDS are loaded in the meta database of ConceptBase. Then, classification of into the meta model is computed by evaluating queries `(e [in] MMC)` for each schema element. The evaluation combines rules from the classification of the relational data model (fig. 5) with the rules at the meta model level. Candidate concepts of the target data model appear as answers to the query `(C in MMC)`. The actual generation of the target data base schema is not supported by the ConceptBase prototype.

After validating the prototype, the O-Telos specifications were implemented in C++ and integrated 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 links). The C++-based mapping assistant called ProFace/Reverse (fig. 13) 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.

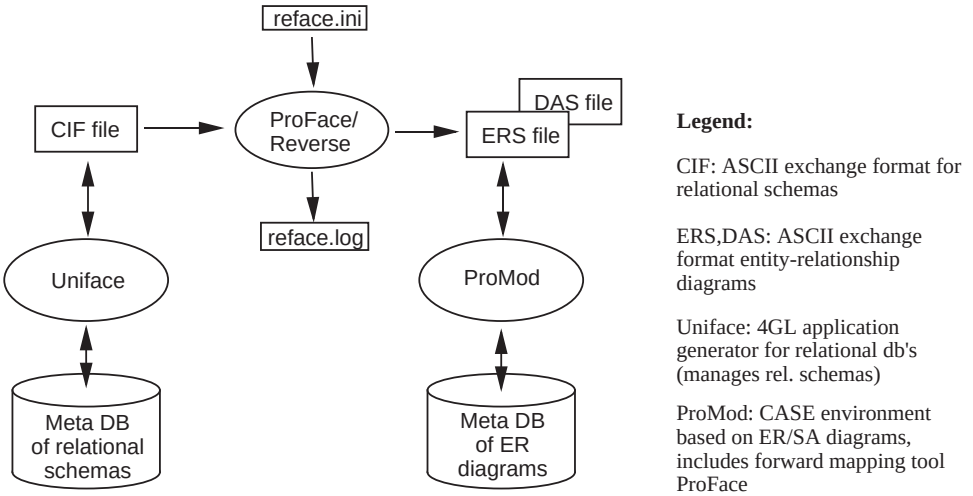

Figure 13: ProFace/Reverse data flow

```
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 14: C++ code generated for query class BinaryLink

Figure 14 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. The rules of fig. 5 are implemented as filters reading and analyzing the input database schema. The 'oracle' rules re-appear as yes/no menues presented to the user.

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.

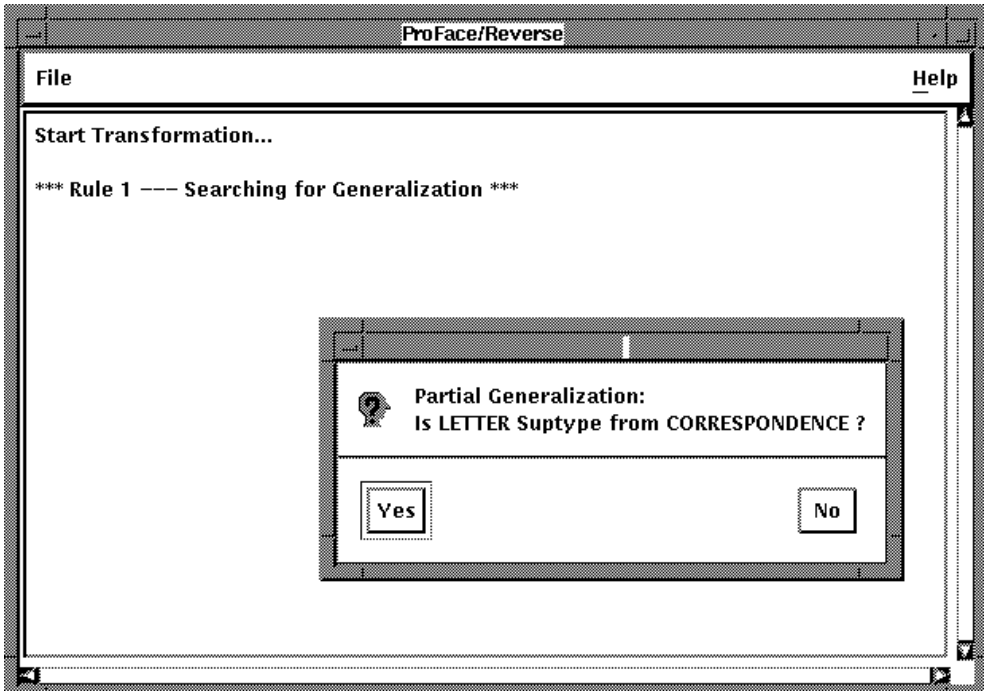

Figure 15: Retrieving missing information from the user

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

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

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 15 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 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 16 shows the result of the mapping of the first case study. Except for the points mentioned before it is identical to the original ERD from which the input RDS 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.

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

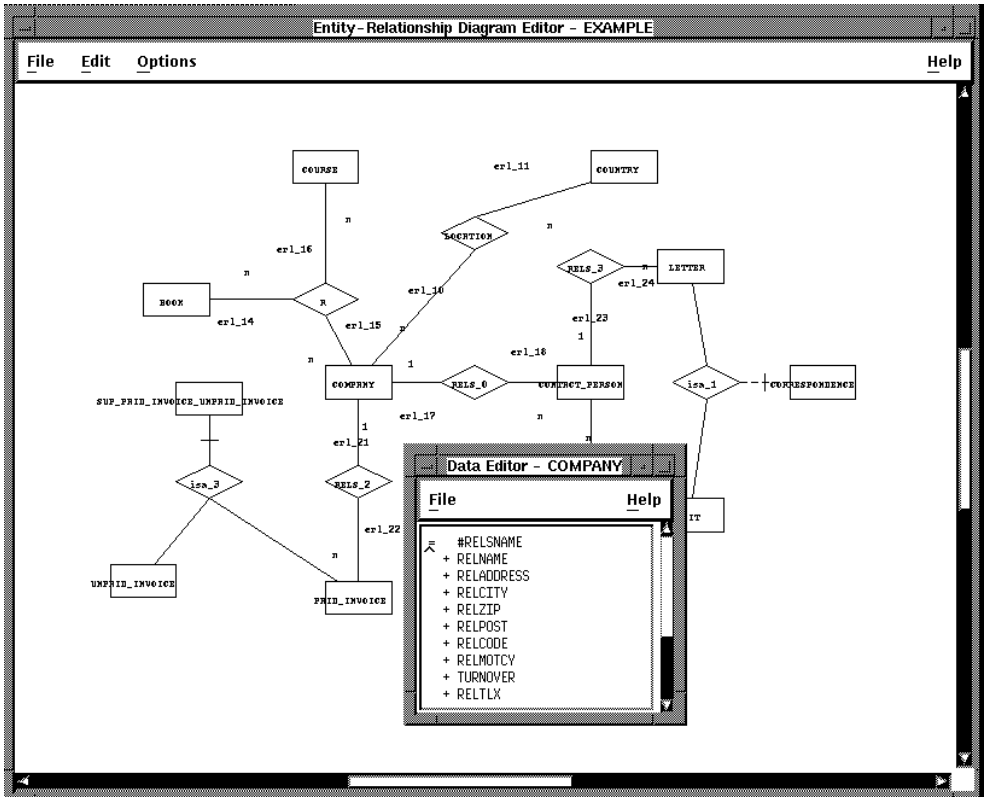

Figure 16: Result of the reverse engineering

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. Extensions

The approach presented so far takes a database schema A as input and produces an "equivalent" schema B in another representation language. For the relational data model it was indicated that missing information is asked from the user. There are however alternatives as proposed by [1, 22]. The database instance can be employed for various tests for validating assumptions like generalization dependencies between

two relations. Because of the nature of instances, only negative information can be obtained, a database instance can be used to rule out certain possibilites.

The typical case is detection of generalizations. We say a type unit `u1` is a generalization of a type unit `u2` if all objects in `u2` are also in `u1`, or formally:

```
forall u1,u2
   (u1 [in] TypeUnit) and (u2 [in] TypeUnit)
   (u1 generalizationOf u2) ==>
   (forall x (x in u2) ==> (x in u1))
```

Note that the variable `x` ranges over objects in the database instance. A hidden quantification is introduced by the meta class predidates ranging over the concepts of the data model. Assume that the type units `PERSON` and `STUDENT` are two instances of `Relation` which are supposed to be generalizations. Then the following statement must be true for any database instance:

```
forall x (x in STUDENT) ==> (x in PERSON)
```

Further assume that `ID` is the primary key of both relations. Then, the test (adapted from [22]) can be performed by a SQL query:

```
select count(STUDENT.ID) from STUDENT
where not (STUDENT.ID in select ID from PERSON)
```

If the result is different from zero then the two relations will formally not qualify for the generalization hypothesis. Even then, the engineer may decide that there is a generalization and the database instance is inconsistent. The ratio of the result to the number of tuples in `STUDENT` may serve as an indicator.

Note that the concept of generalization is defined at the meta model level and that the test is described in terms of the database schema (two levels below the meta model). The concept is in fact independent from the data model. Specialization to a specific data model (here: relational data model) and a specific database schema is performed by substituting certain variables in the formula by constants.

7. Summary and Outlook

Meta models are a suitable way to represent knowledge common to different data models. It makes data models comparable to each other and, by choice of a deductive language, take a considerable portion of the transformation between data models. This paper applied the concept to the reverse engineering of relational database schemas to entity-relationship diagrams. The novelties are summarized as follows:

- An extensible meta model at the top level of the IRDS hierarchy with classification rules that quantify over data model concepts. Thereby, the rules are independent from a specific data model and reusable for transformation between all data models. This is new compared to previous approaches which describe transformation in term of the specific data models.
- A uniform representation of heterogeneous data models and database schemas as O-Telos facts and rules. Translation of data model specific dictions like

foreign keys into meta model dictions is achieved by deductive rules. Even properties of the database instance can be abstractly described at meta model level and be specialized to executable queries.

- The meta model is executable when loading it together with the data model description and the database schema discription into ConceptBase. Computation follows the declarative semantics of Datalog with negation. This property was exploited in a prototype.

- An implementation within a commercial CASE toolkit and the application to two real cases show the practicability of the approach.

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 toolkit already supplied such a facility. 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.

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

- 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 O-Telos is the possibility to reason about both. The data models can be queried hypothetically for possible mappings, i.e., without taking an example schema into account.

Acknowledgements. We would like to thank Matthias Hallmann and Wolf Fischer from debis Systemhaus for helping us to evaluate our prototype within a commercial system. Many thanks to Matthias Jarke for providing hints on the symmetry of the mapping.

References

[1] Andersson M., "Extracting an entity relationship schema from a relational database through reverse engineering" In Loucopoulos P. (ed.): *Entity-Relationship Approach (ER'94)*, Springer-Verlag, pp. 403-419, 1994.

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

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

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

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

[6] 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.

[7] Ceri S., Gottlob G., Tanca L., *Logic programming and databases*, Springer-Verlag, 1990.

[8] Chikofsky E.J., Cross J.H., "Reverse engineering and design recovery: a taxonomy", In *IEEE Software*, pp. 13-17, January 1990.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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, 1995.

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

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

[17] 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 Interoperability in Multidatabase System (RIDE-IMS)*, Vienna, Austria, 1993.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Petit J.-M., Kouloumdjian J., Boulicaut J.-F., Toumani F., "Using queries to improve database reverse engineering", In Loucopoulos P. (ed.): *Entity-Relationship Approach (ER'94)*, Springer-Verlag, pp. 369-386, 1994.

[23] Premerlani W.J., Blaha M.R., "An approach for reverse engineering of relational databases", *Communications of the ACM*, 37, 5, pp. 42-49, May 1994.

[24] 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.