An EER Prototyping Environment and its Implementation in a Datalog Language*

Norbert Kehrer, Gustaf Neumann
Vienna University of Economics and Business Administration
Department of Management Information Systems
Augasse 2–6
A–1090 Vienna, Austria

kehrer@wu-wien.ac.at, neumann@wu-wien.ac.at

Abstract

In this paper we present an approach to represent schema information, application data and integrity constraints as a datalog program. The schema information is supplied as an enhanced entity relationship (EER) model which is transformed by a one-to-one mapping into a set of ground facts. The application data corresponding to the schema is represented by ground facts as well by using a single predicate named observation. In order to check whether the application data conforms to the given schema, a set of integrity rules is defined which expresses the dependencies (mostly functional and inclusion dependencies) implied by the EER model. In order to check whether the application EER model is a valid EER model a meta EER model is defined. Any application EER diagram appears as an instance of the meta EER diagram which can be specified using the same application data representation as above. This way a single set of the integrity rules can be used to check the conformance between the application data and the application EER diagram, the meta EER diagram and the application EER diagram. Since the representation of the meta EER diagram is an instance of the meta EER diagram, the validity of the meta EER diagram is checked as well. The resulting logic program is composed of the application data, the application schema, the meta schema, a general set of constraints plus optionally additional application specific constraints and deduction rules.

1 Introduction

The entity relationship (ER) approach is undoubtably a very popular tool for the communication between a database designer and a user of a database system. In our opinion the reason for the popularity is due to the fact that ER models have a graphical representation and that ER models can be mapped in a systematic way to a relational database schema [TYF86]. Extensions of the basic formalism of Chen [Che76] were proposed by various authors to capture concepts like generalization [SS77] or categories [EWH85]. In this paper we will follow the enhanced entity relationship (EER) flavor as presented in [EN89]. Within this paper we will not deal with certain EER constructs such as composite, derived or multi-valued attributes and predicate defined categories or sub-/superclasses.

We will present a prototyping environment based on a deductive database system for EER model designers in the form of an executable EER specification. To accomplish this goal we

*Published in the Proceedings of: "*Entity-Relationship Approach - ER'92, 11th International Conference on the Entity-Relationship Approach*", pp 243-261, Karlsruhe, Germany, October, 1992.

develop a representation of EER models, which is derived from a meta EER model, and a representation of the application data consisting only of a single table, which can be provided at design time. We do not focus in this paper on the generation of efficient information system for production use, but we want to provide instead a tool for a database designer to model EER applications together with its data, to experiment with various design approaches and to refine the EER model if necessary. The provided integrity constraints may be used to check the consistency of both, the application EER model (by checking it against the meta EER model) and the application data. In order to obtain an efficient information system using a relational database system the standard mapping techniques can be applied. This paper does not, however, address application issues like transactions, locking, user interfaces, output, etc.

2 Using a Deductive Database System for EER Modelling

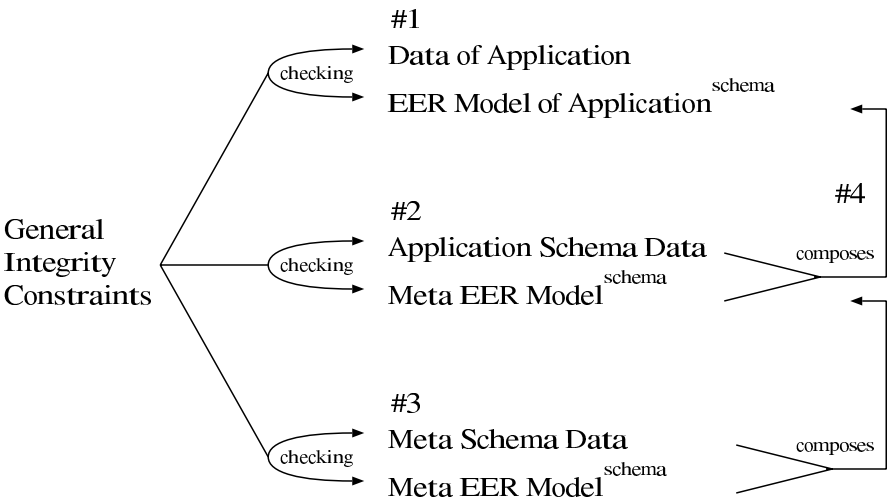

Figure 1: Using a general EER specific set of constraints to check meta and application integrity

In our approach we start from a given EER diagram which is represented as a set of facts. These facts are the result of a pure one-to-one mapping of the EER concepts used in the diagram representing the schema information of the application. In addition to the schema the application data will be given as facts as well. By applying general consistency rules we are able to check the conformance of data and schema (see Figure 1). This consistency checking is implemented using a deductive database system. We have chosen the datalog language SYLLOG [WMSW90] with its near-English representation and its comfortable user interface, where one can easily generate explanations for inconsistent data. SYLLOG is implemented using a backchain iteration procedure which gives clear semantics to stratified logic knowledge bases [ABW87]. In general, our approach does not rely on the SYLLOG system. The same representation and the same rules can be used in a different syntactic form in a Prolog based environment [KN91], where the system implementor has to care about termination, or in deductive database system providing sound and complete computations in a datalog language. For a port, the target system must be able to deal with negation in stratified programs. Function symbols and arithmetic computations are not required. Candidates for such target systems are for example LDL [NT89] or RDL [KMS90].

We have developed a representation schema that allows us to keep the schema and data together in the same database. This prerequisite allows us to specify further application EER schemas as instances of a meta EER diagram. The meta EER diagram specifies how an application EER diagram might be composed. Since the application EER models appear as data of

the meta EER model their well-formedness can be checked using the same general integrity constraints. Finally it can be checked, whether the meta EER diagram is a valid EER diagram.

The resulting checked database has the form of a datalog program which can be extended with deduction rules or additional constraints (which exceed the expressibility of the EER methodology) as needed by the application. In certain cases one might want to add further constraints which are not entailed by the EER diagram (such as domain restrictions). Such constraints will be discussed in the section of the meta EER model.

Since all constraints are formulated in this paper in terms of datalog rules, one might ask, why we used the EER methodology in the first place, and why we have not used a logic specification instead. We think that the main advantages of the EER methodology are its graphical representation, which forces the user to express a relatively wide set of constraints without the need of going into representation details.

Our work was influenced by [DZ88], who developed a first order specification of inference rules together with a set of integrity constraints for a graphical information systems specification language. In contrast to our work, a new formalism called LOCS is proposed. We based our work on the well established and well known EER approach. The paper of [DZ88] does not mention any attempt to check the wellformedness of the application schema using the same integrity constraints.

3 Representing EER-Diagrams and its Data in a Datalog Language

The information contained in an EER diagram can be separated into two components.

1. An extensional part containing the names of the concepts used in the EER model, a certain classification of these concepts (attribute, entity type, relationship type), the links between these basic concepts and the definition of certain properties of the concepts, and

2. an intensional part containing integrity constraints and deduction rules. In this paper we are concerned primarily with integrity constraints that are induced by the EER model.

The extensional part of an application consists of the extensional part of the schema as specified by the EER diagram plus application data. The intensional part of the application is composed of the intensional part of the schema plus optionally additional constraints over the data that cannot be expressed in an EER model. We will show some examples for such constraints in a later section which discusses a meta EER model. The integrity rules will be used to check the conformance between the schema and the data. In order to check the wellformedness of a schema we will introduce a meta EER diagram.

The extensional part of an EER diagram can be obtained by performing a simple one-to-one mapping from the EER diagram to a set of facts. In our representation we represent an EER diagram in terms of the links between the basic EER concepts. These links are either roles, attributes, generalizations, or categories. In addition a predicate is needed to identify weak entities. We are using the following SYLLOG sentences for representing EER diagrams (words starting with the prefix *eg-*, *some-*, *the-*, *a-* and *an-* are logical variables in SYLLOG; the prefixes are used only for readability. Two variables that differ only in their prefix denote the identical variable):

1. One-to-one mapping of roles:

   ```
   Role a-role-name a-rel-name an-ent-name the-cardinality the-participation
   ```

   where *the-cardinality* is either *One* or *Many* and *the-participation* is either *Partial* or *Total*.

   In EER diagrams roles are arcs connecting entity types and relationship types. All roles are labelled with their names and their cardinalities (1 for *One*, n or m for *Many*). Role

arcs drawn as thick lines denote a total participation of the participating entity type in the connected relationship type, thin lines denote a partial participation.

2. One-to-one mapping of attributes:

   ```
   Attribute an-att-name a-mt-name the-type
   Composite a-mt-name an-att-name an-att-name-component
   ```

   where *the-type* is one of *Simple*, *Identifying* or *Multivalued*. *a-mt-name* stands for the name of a modelled type of the EER methodology, i.e. an entity type or a relationship type.

   In the EER diagrams attributes are drawn as ellipses, identifying attributes are underlined. As mentioned above the examples throughout this paper will contain only simple or identifying attributes.

3. One-to-one mapping of generalizations:

   ```
   Generalization a-gen-name an-ent-name the-disjointness the-completeness
   G-sub a-gen-name an-ent-name
   ```

   where *the-disjointness* is either *Overlapping* or *Disjoint* and *the-completeness* is either *Partial* or *Total*.

   Generalizations are denoted in the diagrams as undirected arcs leading from a supertype to a small circle containing either a *d* (for disjoint subclasses) or an *o* (overlapping subclasses) which is connected with arrows pointing to the entity types of the subclasses. A thick line between the supertype and the small circle indicates a total generalization. In cases where a supertype has a single subtype the undirected arc and the small circle can be omitted. In such cases a partial overlapping generalization is assumed. In our representation a unique generalization name (*a-gen-name*) is used to represent the circle symbol. The supertype of a generalization is included in the predicate *Generalization*. Each subtype is stated as a fact for the predicate *G-sub*

4. One-to-one mapping of categories:

   ```
   Category a-cat-name an-ent-name the-completeness
   C-super a-cat-name an-ent-name
   ```

   where *the-completeness* is either *Partial* or *Total*.

   Categories are drawn like generalizations except that the character *u* is used in the circle and there is only one arrow from the circle to the category entity type.

5. Identification of weak entity types:

   ```
   Identifies a-rel-name a-weak-ent-name
   ```

   Weak entity types and identifying relationship types are drawn in a grey box using a thicker line style.

A Schema of a simple Airline Application and its Extensional Representation

Now we will use the relational schema for representing EER diagrams introduced in the last section for a sample application. The EER diagram in Figure 2 is mapped to a set of facts. It should be noted that this transformation is easy enough to be done by a fairly simple transformation program. We have developed such a program that transforms EER diagrams drawn with the publicly available graphical editor TGIF [Che91] into a relational schema conforming

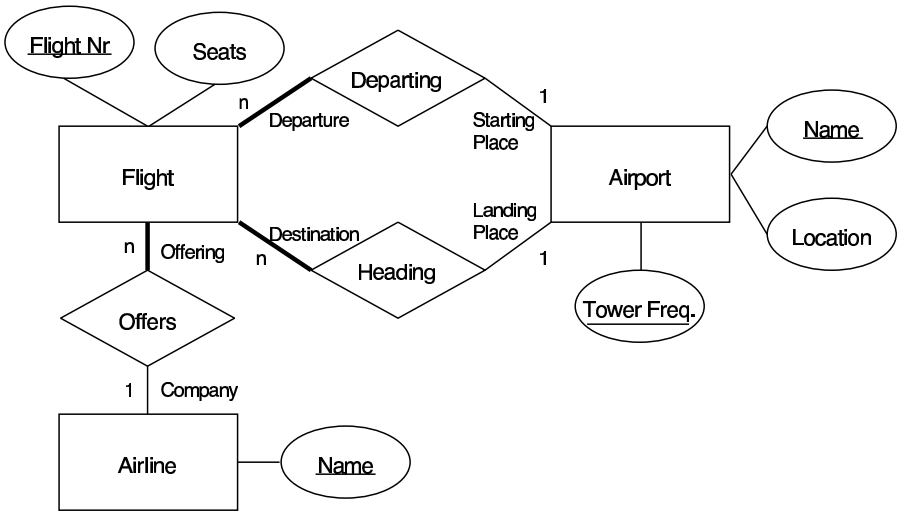

Figure 2: An EER Diagram specifying a simple Airline Information System

to the following SYLLOG tables. In SYLLOG a table is written as SYLLOG sentence followed by a double dashed line followed by the values. All of these facts have to be ground.

Since the EER diagram contains only entity types, relationship types and attributes it suffices to use *Role* and *Attribute* sentences.

```
Role the-name    the-relationship-type the-entity-type the-cardinality the-participation
=================================================================================
     From        Departing             Airport          One             Partial
     Departure   Departing             Flight           Many            Total
     To          Heading               Airport          One             Partial
     Destination Heading               Flight           Many            Total
     Company     Offers                Flight           Many            Total
     Offering    Offers                Flight           One             Partial

Attribute the-attribute the-entity-type the-type-of-attribute
=============================================================
          Country       Airport          Simple
          Name          Airport          Identifying
          Location      Airport          Simple
          Tower-Freq    Airport          Identifying
          Flight-Nr     Flight           Identifying
          Seats         Flight           Simple
          Location      Airline          Simple
          Name          Airline          Identifying
```

A Set of Instances for the Airline Schema

In order to make the EER diagram executable it is necessary to store the data for the diagram in the same SYLLOG knowledge base. To achieve maximal flexibility we decided to use a fairly atomistic representation schema based on so called *Observations*. An observation is a fact determining that some attribute (or role) belonging to a certain entity type (or relationship type) has for a given object a certain value. Thus all instances of the EER schema are defined by a single predicate *Observation* with four arguments. The following patterns are possible:

```
Observation some-attribute some-entity-type some-tuple-identifier some-value or
Observation some-attribute some-relationship-type some-tuple-identifier some-value or
Observation some-role some-relationship-type some-tuple-identifier some-value
```

The first two arguments refer to the schema information, the third argument *tuple-identifier* is used to group the various *Observation*s to a certain tuple (aggregation). The *tuple-identifier* determines uniquely the object in the database. The tuple identifier is a concept comparable to the surrogate in [Cod79].

Note that a representation based on *Observations* allows us to cope with null values (no observation available) or with multi-valued attributes (several observations with identical first three arguments and different fourth arguments) in a simple way.

```
Observation the-attribute the-modelled-type the-tuple-id the-value
=================================================================
            Location      Airline           Airline1      Wien
            Name          Airline           Airline1      Aua
            Location      Airline           Airline2      Budapest
            Name          Airline           Airline2      Malev
            Name          Airport           Airport1      Wien-Schwechat
            Country       Airport           Airport1      Wien
            Location      Airport           Airport1      Schwechat
            Tower-Freq    Airport           Airport1      123
            Name          Airport           Airport2      Jfk
            Country       Airport           Airport2      Usa
            Location      Airport           Airport2      New-York-City
            Tower-Freq    Airport           Airport2      222
            Flight-Nr     Flight            Flight1       123
            Seats         Flight            Flight1       130
            Flight-Nr     Flight            Flight2       234
            Seats         Flight            Flight2       170
            From          Departing         Departing1    Airport2
            Departure     Departing         Departing1    Flight1
            To            Heading           Heading1      Airport1
            Destination   Heading           Heading1      Flight1
```

One might argue that this representation is very atomistic and hard to use. The major advantages are however that it allows a very general way to reference the EER concepts from within the integrity rules and it is fairly easy to perform certain schema modifications (eg. introducing a new attribute).

In order to provide a more more traditional view of the data specified in the *Observation* facts one might provide rules. A SYLLOG rule consists of one or several SYLLOG sentences (premises) followed by a line consisting of dashes followed by one or several SYLLOG sentences (conclusions). The following two SYLLOG rules show how to make the data more accessible and how to specify additional rules which are not expressible in the EER model (recursively defined transitive closure).

```
Observation Flight-Nr   Flight    some-tuple-flight  some-flight
Observation Seats       Flight    some-tuple-flight  some-seats
Observation Offering    Offers    some-tuple-offers  some-tuple-flight
Observation Company     Offers    some-tuple-offers  some-tuple-airline
Observation Name        Airline   some-tuple-airline some-airline
Observation Departure   Departing a-tuple-departing  some-tuple-flight
Observation From        Departing a-tuple-departing  some-airport-a
Observation Destination Heading   a-tuple-heading    some-tuple-flight
Observation To          Heading   a-tuple-heading    some-airport-b
Observation Name        Airport   some-airport-b     the-place-b
Observation Name        Airport   some-airport-a     the-place-a
-------------------------------------------------------------------------------------------
some-flight is a flight of some-airline from the-place-a to the-place-b with some-seats seats
one can fly from some-place-a to some-place-b

one can fly from some-place-a to some-place-b
one can fly from some-place-b to some-place-c
-----------------------------------------
one can fly from some-place-a to some-place-c
```

Rules like the first one can be found by applying a procedure like in [TYF86], [Teo90], or [EN89] to transform an EER model into a relational schema. These procedures are oriented towards generating a small number of relations together with their attributes. Those relations are used as conclusions of SYLLOG rules where the attributes are stated as variables. The premises of the rules are formed by grouping together the *Observations* that are needed to specify the variables in the conclusion. Both the generation of a relational schema and the grouping of the *Observations* can be done automatically, so the advantages of the representation of the data as *Observations* and of the possibility to easily access the data can be utilized without additional effort.

The transformation into relations with a high number of attributes can entail some disadvantages: If a particular *Observation* is missing to form such a wide relation, either the whole relation tuple will be omitted or a special representation for null values is needed. This null value problem occurs if an attribute is missing or when a partial n-to-1 relation is resolved as additional attributes of the table corresponding to an entity type. An easy solution for missing attributes would be to use another integrity constraint that forbids missing values. This constraint would be very similar to the constraint that each role in a relation must be specified, which is discussed in the next section. This consistency rule would be a non standard extension. A solution for the partial relation problem would be to map such relationship types to separate tables.

4 General Integrity Constraints of the EER Model

Now we will present a set of general integrity constraints which can be used to check whether the instances of an EER diagram conform to the restrictions entailed by this EER diagram. We describe the different types of integrity constraints and show how integrity checking can be implemented using stratified datalog knowledge bases in SYLLOG. It is assumed that the EER diagram is represented as set of facts for the predicates resulting from the one-to-one mapping of the meta EER diagram (*Attribute*, *Role*, etc.) and that the instances of the EER diagram are given as ground facts using the predicate *Observation*.

Because we have available both – the information about the EER model and the instances of the model – we are able to check the integrity of a database with one general set of integrity constraints. Unlike other approaches [TYF86, MS89, EN89] which generate for each EER model it's own set of integrity constraints, we only have one set of integrity constraints which can be used for any EER model.

An integrity constraint is formulated as a SYLLOG deduction rule with the conclusion stating that the constraint is violated and specifying the role or attribute and the modelled type which violate the constraint. The premise of the rule states the conditions for the violation and combines predicates referring to EER schema information and the *Observation* predicate containing the instances of the EER schema. To make the rules shorter and more readable we have introduced auxiliary predicates.

4.1 Functional Dependencies

Marking attributes as identifying and the specification of cardinalities of 1 in relationship types in an EER model are ways to express functional dependencies on the modelled data.

A functional dependency (FD) is a constraint on a relation R which states that the values of a tuple on one set of attributes X uniquely determine the values on another set Y of attributes. It is written as $X \Rightarrow Y$ and is formally defined by the following implication [Mai83, GV89]:

$$t_1(X) = t_2(X) \rightarrow t_1(Y) = t_2(Y)$$

t_1 and t_2 are two different tuples of R. If the values on the set of attributes X are the same in t_1 and t_2 then the values on the attribute set Y have to be the same, too.

A FD is violated if there exist two tuples which have the same values in X and different values in Y. In SYLLOG this can be expressed by the following rule:

```
two observations with different values in [eg-att-Rhs eg-mt-Rhs] are eg-t1 eg-t2
not:  two observations with different values in [eg-att-Lhs eg-mt-Lhs] are eg-t1 eg-t2
-------------------------------------------------------------------------------------
value of [eg-att-Lhs eg-mt-Lhs] does not determine value of [eg-att-Rhs eg-mt-Rhs]
```

This rule defines the violation of a functional dependency of the type *value(attribute-LHS) ⇒ value(attribute-RHS)* (where the *attribute*s are atomic attributes). Since in our representation both the values and the tuple identifiers are accessible in the same way, we could express dependencies of the form *value(attribute-LHS) ⇒ tupid(attribute-RHS)* or *tupid(attribute-LHS) ⇒ value(attribute-RHS)* or *tupid(attribute-LHS) ⇒ tupid(attribute-RHS)* with the same ease. We could generalize the rule as follows:

```
two observations with different eg-vtR in [eg-attR eg-mtR] are eg-t1 eg-t2
not:  two observations with different eg-vtL in [eg-attL eg-mtL] are eg-t1 eg-t2
-------------------------------------------------------------------------------
eg-vtL of [eg-attL eg-mtL] does not determine eg-vtR of [eg-attR eg-mtR] for ATOMIC
```

In this syllogism *vt* stands for '*value* or *tuple*'. In cases where the left hand side of a functional dependency is not atomic ($AB \rightarrow C$), it can be stated informally that the dependency is violated if "*...for any different RHS all LHS are equal*", or "*...no elements of the LHS are allowed to be different*". We can use a syllogism to generate the left hand side attributes and proceed as follows:

```
two observations with different eg-vtR in [eg-attR eg-mtR] are eg-t1 eg-t2
not:  any observations of type eg-t with different eg-vtL in eg-mtR using eg-t1 eg-t2 exist
-------------------------------------------------------------------------------
eg-vtL of [eg-attL eg-mtL] does not determine eg-vtR of [eg-attR eg-mtR] for eg-t

Some syllogism that concludes eg-attL for eg-mtL
two observations with different eg-vtL in [eg-attL eg-mtL] are eg-t1 eg-t2
-----------------------------------------------------------------------------------
any observations of type specialized-type with different eg-vtL in mtL using eg-t1 eg-t2 exist
```

Here the checking if two tuples of the modelled type of the left hand side are different is done by a special syllogism which determines the items of the left hand side of the FD. For each type of non-atomic left hand side one wants to use, an own syllogism has to be written, which may be selected by an integrity rule via the variable *specialized-type*. Examples for this procedure will be given later.

Identifying attribute determines tuple identifier

For each identifying attribute *att* of a modeled type of the EER schema there exists a functional dependency between *att* and the tuple identifier of the form:

value(identifying-att,modelled-type) ⇒ tupid(modelled-type)

This corresponds to the definition of an identifying attribute as an attribute whose values can be used to identify an entity uniquely [EN89], because in our approach an entity is represented by its tuple identifier.

In SYLLOG this is formulated by a rule which has the conditions for the violation of the functional dependency as its premises and states the attribute and modelled type which violates the FD as conclusion.

```
Attribute eg-att eg-mt identifying
not:  eg-mt is a weak entity type
value of [eg-att eg-mt] does not determine tupid of [eg-mt eg-att] for ATOMIC
------------------------------------------------------------------------
fd of eg-att -> tuple id.  in eg-mt is violated
```

It has to be noted that the identifying attribute of weak entity types does not determine the weak entity [Che76], but together with the owner entities it does. So we need a separate rule to check this functional dependency violation for weak entity types. It states that the identifying attribute together with the tuple identifier(s) of the owner(s) determine the tuple identifier of the weak entity type:

$$\textit{value(identifying-att+tupids-of-owners,weak-ent)} \Rightarrow \textit{tupid(weak-ent)}$$

Tuple identifier determines single-valued attributes

Chen defined a (single-valued) attribute as a function which maps from an entity set or a relationship set into a value set [Che76]. For our representation this means that the value of each single-valued attribute of the modelled type *mt* is determined by the tuple identifier of *mt*:

$$\textit{tupid(modelled-type)} \Rightarrow \textit{value(any-attribute,modelled-type)}$$

We need not check the constraint that the identifying attribute determines the values of the other attributes [MS89], because it follows from the two previous constraints:

$$\textit{value(identifying-att,modelled-type)} \Rightarrow \textit{tupid(modelled-type)} \wedge$$
$$\textit{tupid(modelled-type)} \Rightarrow \textit{value(any-attribute,modelled-type)} \rightarrow$$
$$\textit{value(identifying-att,modelled-type)} \Rightarrow \textit{value(any-attribute,modelled-type)}$$

Entities participating in a relationship type with cardinality *One*

Each role r of a relationship type rel in which an entity type participates with cardinality *One* is determined by all other roles of rel together [TYF86]:

$$\textit{value(other roles,rel-type)} \Rightarrow \textit{value(one-role,rel-type)}$$

This constraint is independent of the degree of the relationship type.

All roles determine tuple identifier

All roles of a relationship type *r* together determine the tuple identifier of *r*:

$$\textit{value(all roles,rel-type)} \Rightarrow \textit{tupid(rel-type)}$$

Which is expressed in SYLLOG as:

```
Role a-role a-rel an-ent the-cardinality the-participation
value of [a-role a-rel] does not determine tupid of [eg-any a-rel] for ALL-ROLES
------------------------------------------------------------------------
fd of all roles -> tuple-id in a-rel is violated
```

```
Role eg-role eg-rel eg-ent eg-cardinality eg-participation
two observations with different eg-vt in [eg-role eg-rel] are eg-t1 eg-t2
-------------------------------------------------------------------------------
any observations of type ALL-ROLES with different eg-vt in eg-rel using eg-t1 eg-t2 exist
```

This is an example of a FD where the left hand side is not atomic, but consists of all the roles of a relationship type. Therefore we define a syllogism to determine two tuples of a relationship type *rel* where at least one observation of a role of *rel* differs in these tuples. The FD violation is checked using the rule for a non-atomic left hand side which is described at the beginning of this section.

4.2 Inclusion Dependencies

The use of relationship types, generalizations, and special-izations in EER models indicates that entity or relationship sets are subsets of some other entity or relationship set. The property of being a subset of another set is covered by inclusion dependencies.

Inclusion dependencies (ID) specify that each member of some set A must also be a member of a set B. An inclusion dependency $A \subseteq B$ is violated iff there is an occurrence (value or tuple identifier) of A which is not an occurrence of B. In SYLLOG this is expressed by the following rule:

```
eg-x1 is an attribute or a role of eg-mt1
eg-x2 is an attribute or a role of eg-mt2
one observation of eg-vt1 of [eg-ra1 eg-mt1] is eg-x
not:  one observation of eg-vt2 of [eg-ra2 eg-mt2] is eg-x
-----------------------------------------------------------
eg-vt1 of [eg-ra1 eg-mt1] is not a eg-vt2 of [eg-ra2 eg-mt2]
```

Like the rule for FD violations this rule may be used to check inclusion dependencies between tuple identifiers and attribute or role values in any combination by specifying the variables *vt1* and *vt2* as *"value"* or *"tupid"*. The sentence "`one observation of eg-vt1 of [eg-ra1 eg-mt1] is eg-x`" returns a tuple identifier or a value of a role or attribute depending on the value of *vt*.

Participating entities included in entity type

The values of a role of a relationship type must be tuple identifiers of the entity type participating in that role:

$$value(role,rel\text{-}type) \subseteq tupid(entity\text{-}type)$$

In SYLLOG the ID is checked by the integrity rule:

```
Role eg-r eg-rel eg-ent eg-card eg-part
value of [eg-r eg-rel] is not a tupid of [eg-att eg-ent]
--------------------------------------------------------
id role eg-r of eg-rel << entity type eg-ent is violated
```

Totally participating entity types

For entity types which participate totally in a relationship type the previous ID has to hold in the other direction, too. Each tuple identifier of an entity type *e* must be a value of a role in which *e* participates totally:

$$tupid(entity) \subseteq value(role,rel\text{-}type)$$

Generalizations

A generalization may be total or partial. A total generalization specifies the constraint that every entity in the superclass must be a member of some subclass in the specialization [EN89]. For our representation this means that in a total generalization with supertype *super* there must be at least one subclass *sub* for each tuple identifier *T* of *super*, where *T* is included in *sub*:

$$\textit{tupid(supertype)} \subseteq \textit{tupid(at-least-one-subtype)}$$

There is also an inclusion dependency *tupid(subtype)* $\subseteq$ *tupid(supertype)* for both partial and total generalizations. In our representation we guarantee through the use of deduction rules that the tuple identifiers of supertypes are also tuple identifiers of the subtypes. Therefore this ID needs not to be checked.

Categories

Like for generalizations there are deduction rules to assure that tuple identifiers of the superclasses are also tuple identifiers of the subclass in a category and that the attributes are inherited. This mechanism may not be applied to partial categories, because not every entity of a supertype has to be member of the subclass. Instead the members of the subclass in partial categories have to be stated explicitly. Therefore we will have to check if the tuple identifiers and attribute values of a subclass in partial category occur in one of the superclasses specified for the category, which is expressed by the inclusion dependencies:

$$\textit{tupid(subclass)} \subseteq \textit{tupid(some-superclass)}$$
$$\textit{value(att,subclass)} \subseteq \textit{value(att,some-superclass)}$$

The mechanism of attribute inheritance will be described in more detail in a later section.

All roles in a relationship must be specified

In each relationship instance the associated entities have to be specified. This constraint is violated if there are two different roles *role1* and *role2* in a relationship type *rel* and the set of tuple identifiers of *rel* which have a value for *role1* is a proper subset of the tuple identifiers of *rel* having a value for *role2*. A is a proper subset of B if $A \subseteq B$ and not $B \subseteq A$. So the constraint may be written as

$$\textit{tupid(role1,rel-type)} \subseteq \textit{tupid(role2,rel-type)} \wedge \neg\, \textit{tupid(role2,rel-type)} \subseteq \textit{tupid(role1,rel-type)}$$

which is expressed in SYLLOG as:

```
Role eg-r1 eg-rel eg-ent1 eg-card1 eg-part1
Role eg-r2 eg-rel eg-ent2 eg-card2 eg-part2
tupid of [eg-r1 eg-rel] is not a tupid of [eg-r2 eg-rel]
not:  tupid of [eg-r2 eg-rel] is not a tupid of [eg-r1 eg-rel]
------------------------------------------------------------
not all roles specified in relationship type eg-rel
```

4.3 Exclusion Dependencies

An exclusion dependency (ED) is the constraint indicating that no member of a set A is a member of a set B (empty intersection). An ED $A \rightleftharpoons B$ is violated iff a tuple identifier of A is also a tuple identifier of B. We define this constraint violation only for tuple identifiers and not for attribute values or roles because this is not expressible in the EER methodology. The SYLLOG representation of the violation of the constraint is easy:

```
Observation eg-ra1 eg-mt1 eg-t1 eg-v1
Observation eg-ra2 eg-mt2 eg-t1 eg-v2
not:  eg-mt1 equal eg-mt2
-----------------------------------------------------
tuple id eg-t1 of eg-mt1 is also a tuple id of eg-mt2
```

Disjoint subclasses

A disjointness constraint on a generalization specifies that the subclasses in the generalization must be disjoint [EN89]. This constraint can be expressed by mutual exclusion dependencies between all the subclasses. Let *sub1* and *sub2* be two different subclasses of a disjoint generalization. If a tuple identifier of *sub1* is also a tuple identifier of *sub2* the ED of disjoint subclasses is violated:

$$tupid(disjoint\text{-}subclass\text{-}1) \rightleftharpoons tupid(disjoint\text{-}subclass\text{-}2)$$

Unique Tuple Identifiers

Two different modeled types *mt1* and *mt2* may not contain the same tuple identifier unless *mt1* is a subtype of *mt2* or *mt2* is a subtype of *mt*. The sentence `eg-sub is a subtype of eg-super` checks if one entity type is a subtype (via a subclass or category) of another entity type.

```
tuple id eg-t1 of eg-mt1 is also a tuple id of eg-mt2
not:  eg-mt1 is a subtype of eg-mt2
not:  eg-mt2 is a subtype of eg-mt1
-----------------------------------------------------
eg-t1 is not a unique tuple identifier of eg-mt1
```

4.4 Schema Conformity

The conformity of the database with the schema – i.e. the attributes, roles, and modelled types appearing in *Observation* facts must be specified in the schema – cannot be checked by one of the above dependencies, because the rules only refer to data integrity whereas the schema conformity may be viewed as an inclusion dependency between data and schema representation. Therefore we use a separate consistency rule for this dependency.

An observation containing an attribute or role *ra* of a modelled type *mt* does not conform to the schema if *ra* or *mt* have not been specified in the schema correspondingly.

```
Observation eg-ra eg-mt eg-tuple-id eg-value
not:  eg-ra is an attribute or a role of eg-mt
-------------------------------------------------------------
schema conformity violated by tuple eg-tuple-id for eg-ra eg-mt
```

4.5 Type Hierarchy

The concepts of the generalization and category allow the construction of a hierarchy of entity types. In our representation we use a mechanism for the inheritance of attributes in that hierarchy and for the inclusion of tuple identifiers of one entity type in other entity types. It is built upon the following rules:

- In a generalization the tuple identifiers and attributes which were stated in an *Observation* for a subtype become tuple identifiers and attributes of the supertype. The entity – represented by the tuple identifier – belongs to both types. The name of the supertype is an alias for the name of the subtype. Therefore we call this process "aliasing".
- In a total category the attributes specified for a superclass are inherited to the subclass, and the tuple identifiers of the superclass become tuple identifiers of the subclass (aliasing).

Nonetheless, additional *Observations* may be specified for the superclass in a generalization and for a subclass of a category. Therefore the corresponding inclusion dependencies, which we described earlier, have to be checked.

The inheritance of attributes and the aliasing of tuple identifiers is performed by SYLLOG deduction rules for a predicate called *Observation-In-Hierarchy*. This predicate has the same arguments as *Observation* and covers all *Observations* plus the ones that result from the inheritance mechanism. Actually the integrity constraint definitions are based upon this predicate except in the cases where *Observation* is used explicitly.

4.6 Integrity Checking in Syllog

The checking of the integrity constraints is performed by querying the constraint violations which were defined above. If there is a positive answer the database is inconsistent. The integrity rules may be extended to include the tuple identifiers of the violated *Observations*. Then the answers to queries contain the observations which violate the constraint, and a set of observations which represents a consistent database could be generated. Such a consistent subset is comprised of the answers to the predicate *Consistent observation* which may be defined as follows:

```
fd of eg-att -> tuple-id in eg-mt is violated for tuple eg-ti
-------------------------------------------------------------
inconsistent observation eg-att eg-mt eg-ti

id role eg-r of eg-rel << entity type eg-ent is violated for tuple eg-ti
------------------------------------------------------------------------
inconsistent observation eg-r eg-rel eg-ti

...

Observation eg-att eg-mt eg-ti eg-v
not:  inconsistent observation eg-att eg-mt eg-ti
-----------------------------------------------
Consistent observation eg-att eg-mt eg-ti eg-v
```

The integrity of an inconsistent database can be recovered by deducing the consistent observations and replacing the set of *Observation* facts by the set of consistent observations. But this new set of observations is not necessarily a consistent database, because of the deletion of the inconsistent observations other integrity constraints (e.g. inclusion dependencies) might be violated now. So the process of integrity checking and generation of consistent observations has to be repeated until no more inconsistent observations can be detected (fixed point).

This iterative process of recovering the integrity of a database is necessary because the integrity constraints are based upon the predicate *Observation* and therefore use the stored facts regardless of the recognition of some of these facts as inconsistent observations by other integrity constraints. One possible solution would be to use the predicate *Consistent observation* instead of the basic *Observation* predicate in the integrity constraints, but here the problem arises that the knowledge base becomes unstratified and cannot be used with traditional inference mechanisms.

4.7 Generating Meaningful Explanations

If, by the integrity checking it can be detected that the database is inconsistent, it is important to know the reasons for the integrity violation to be able to update the database appropriately. SYLLOG offers the facility to generate explanations of the answers it gives. This facility can be used to detect the reasons for integrity constraints violations. A rule to check the consistency of the whole database can be defined as:

```
not:  some FD is violated
not:  some ID is violated
not:  some ED is violated
not:  there is some tuple identifier that is not unique
...
-------------------------------------------------------------------
general integrity constraints hold

fd of eg-att -> tuple id.  in eg-mt is violated
-----------------------------------------------
some FD is violated

fd of all roles -> tuple-id in a-rel is violated
------------------------------------------------
some FD is violated

...
```

The integrity of a database may be checked by submitting the query "`general integrity constraints hold`" to SYLLOG. If the answer is *"No"* the database is violated and an explanation like the following may be generated.

```
general integrity constraints hold
==========================

  Sorry, no

  Because....

not :  some FD is violated
not :  some ID is violated
not :  some ED is violated
not : there is some tuple identifier that is not unique
...
-------------------------------------------------------------------
general integrity constraints hold

t123 is not a unique tuple identifier of departing
--------------------------------------------------
not : there is some tuple identifier that is not unique

tuple id t123 of departing is also a tuple id of offers
not :  departing is a subtype of offers
not :  offers is a subtype of departing
-----------------------------------------------------
t123 is not a unique tuple identifier of departing

observation from departing t123 Vienna
observation company offers t123 Aua
not :  departing equal offers
-----------------------------------------------------
tuple id t123 of departing is also a tuple id of offers
```

The items in italics could not be proved by SYLLOG. This explanation shows that *t123* was used as a tuple identifier for two different modelled types (*departing* and *offers*).

By using different groupings of the premises of the rule that checks the integrity of the database different aspects of integrity violations may be covered and the different explanations are generated. E.g. if we use the following rules to check the integrity we obtain for each modelled type as an explanation the reason why this modelled type violates the integrity:

```
not:  fd of all roles -> tuple-id in eg-mt is violated
not:  id role eg-r of eg-mt << entity type eg-ent is violated
not:  eg-ti is not a unique tuple identifier of eg-mt
not:  schema conformity violated by tuple eg-tuple-id for eg-ra eg-mt
...
-----------------------------------------------------------------
general integrity constraints hold for modelled type eg-mt

eg-mt is a modelled type
not:  general integrity constraints hold for modelled type eg-mt
-----------------------------------------------------------
general integrity constraints violated
```

Another useful feature of SYLLOG is the possibility to generate the domain of a variable, i.e the values it may have. So when we have the query

```
fd of eg-att -> tuple-id in eg-mt is violated for tuple eg-ti
```

e.g. the domain of *eg-att* can be computed which is the set of all attributes in the current EER model. The domain computations are very useful in the exploration of the EER model and to specialize variables in queries.

5 Using a Meta EER Diagram to Reason about the Consistency of an Application EER Model

So far, we are only able to check whether some given data conforms to the integrity constraints of a given EER diagram. In a next step we will check whether the EER diagram is a valid EER diagram. This task is performed mostly by using the meta EER diagram of Figure 3 and by a few further integrity constraints which are not expressible in the meta EER diagram.

The meta EER diagram in Figure 3 can be read as follows: The central EER concepts are *entity type* and *relationship type*. Both of these concepts are generalized to so called *modelled types*. Since a modelled type is either an entity type or a relationship type a disjoint generalization was used. A modelled type might be described by *attributes*. An attribute is identified by a name (identifying attribute) and characterized by its *type* (simple, identifying or multivalued). Composite attributes are constructed using the composite relationship type). Since the names of attributes are only unique per modelled type, attributes are modelled as weak entities with the modelled type as owner.

Entity types and relationship types can be connected via roles, which are identified using a name, and which have a cardinality and a participation value. The role names are unique per relationship type, each occurrence of a relationship type participates in the *participates* relation (total participation). Each *weak entity type* (a subtype of entity type) is identified by an *identifying relationship type* (subtype of relationship type) and vice versa. The enhanced ER constructs of the *category* and the *generalization* are used to define hierarchies of entity types. A generalization (identified by name, characterized by the attributes *disjointness* and *completeness*) has one entity type a supertype and might have several (one or many) entity types as subtypes. A *category* on the contrary has one subtype and might have several supertypes.

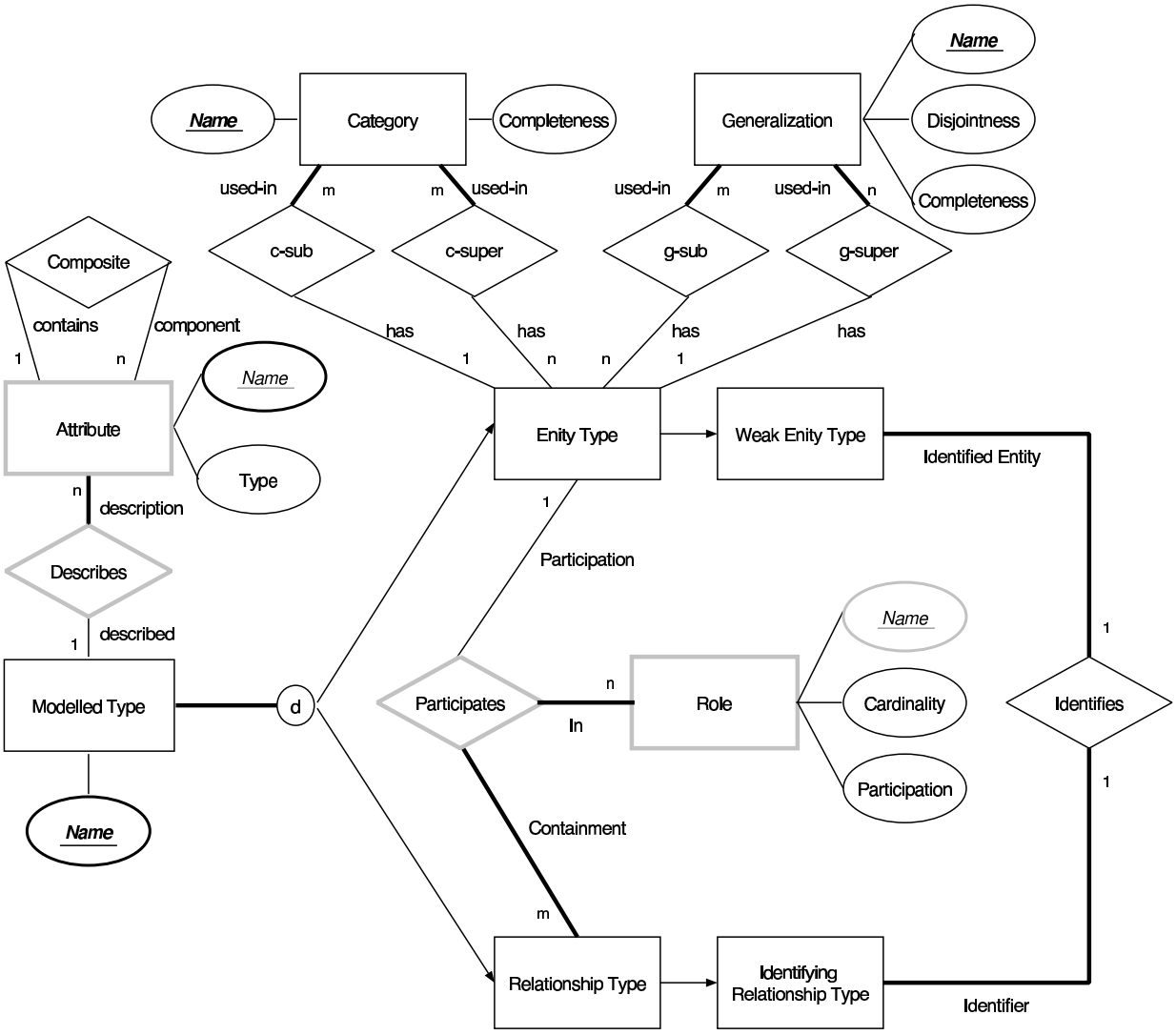

Figure 3: A Meta EER Diagram

The Schema of the Meta EER Model as One-to-One Mapping from the EER Diagram

Since the meta model is an EER model we can transform it into a SYLLOG knowledge base using the same procedure as for the application EER model (airline example). Below you will find the result of the one-to-one mapping.

```
Attribute the-attribute the-entity-type the-type-of-attribute
=========================================================
          Name           Modelled-Type   Identifying
          Name           Attribute       Identifying
          Type           Attribute       Simple
          Name           Role            Identifying
          Cardinality    Role            Simple
          Participation  Role            Simple
          Name           Generalization  Identifying
          Completeness   Generalization  Simple
          Disjointness   Generalization  Simple
          Name           Category        Identifying
          Completeness   Category        Simple

Role the-name          the-rel-type  the-entity-type   the-cardinality the-participation
=============================================================================================
     Description       Describes     Attribute         Many            Total
```

```
     Described          Describes     Modelled-Type     One             Partial
     Contains           Composite     Attribute         One             Partial
     Component          Composite     Attribute         Many            Partial
     Participation      Participates  Entity-Type       One             Partial
     Containment        Participates  Rel-Type          Many            Partial
     In                 Participates  Role              Many            Total
     Identified-Entity  Identifies    Weak-Entity-Type  One             Total
     Identifier         Identifies    Id-Rel-Type       One             Total
     Used-in            G-Super       Generalization    Many            Total
     Has                G-Super       Entity-Type       One             Partial
     Used-in            G-Sub         Generalization    Many            Total
     Has                G-Sub         Entity-Type       Many            Partial
     Used-in            C-Sub         Category          Many            Total
     Has                C-Sub         Entity-Type       One             Partial
     Used-in            C-Super       Category          Many            Total
     Has                C-Super       Entity-Type       Many            Partial


Generalization the-name     the-entity-type the-disjointness the-participation
=============================================================================
               Types        Modelled-Type   Disjoint         Total
               Entity-Types Entity-Type     Disjoint         Partial
               Rel-Types    Rel-Type        Disjoint         Partial

G-sub the-name     the-subtype
==============================
      Types        Entity-Type
      Types        Rel-Type
      Entity-Types Weak-Entity-Type
      Rel-Types    Identifying-Rel-Type

the relationship some-relationship-type identifies the weak entity some-entity-type
===================================================================================
                 Describes                                          Attribute
                 Participates                                       Role
```

All but the names of the generalizations can be inferred automatically from the diagram. Our transformation program returns as "name" of Generalization its coordinate pairs. In the SYLLOG representation above the generalization names *Types*, *Entity-Types* and *Rel-Types* were used for better readability.

5.1 The Schema of the Airline Example as a Set of Instances for the Meta EER Model

The schema of the application EER model was represented as a set of SYLLOG sentences of the form:

```
Attribute Country Airport Simple
Attribute Name    Airport Identifying
...
```

The same information can be expressed as follows as data for the meta EER model:

```
Observation some-attribute-name some-mt-name some-tupid    some-value
=====================================================================
          Name                 Attribute    Attribute1    Country
          Type                 Attribute    Attribute1    Simple
          Description          Describes    Describes1    Attribute1
          Described            Describes    Describes1    Entity-Type-1
          Name                 Entity-Type  Entity-Type-1 Airport
          Name                 Attribute    Attribute2    Name
          Type                 Attribute    Attribute2    Identifying
          Description          Describes    Describes2    Attribute2
          Described            Describes    Describes2    Entity-Type-1
...
```

This rather complicated representation does not turn out to be a problem since the transformation from the diagram is performed by a program. It is interesting to note that the transformation from the one-to-one mapping (SYLLOG sentence starting with *Attribute*) to the notation using *Observation* is nontrivial, since it must be distinguished between a first reference and a later reference to generate the correct tuple identifiers. The transformation in the other direction (from *Observation* to eg. *Attribute*) is much easier:

```
Observation Name        Attribute   some-att some-attribute-name
Observation Type        Attribute   some-att the-type-of-attribute
Observation Description Describes   some-rel some-att
Observation Described   Describes   some-rel some-mt
Observation Name        an-et-or-rt some-mt  some-mt-name
-------------------------------------------------------------------------
Attribute some-attribute-name some-mt-name the-type-of-attribute
```

Using the schema information of the meta EER diagram and the instances above the well-formedness of the application EER diagram (the airline schema) can be checked by applying the general EER integrity constraints.

One can, however, check more than is specified in the EER diagram. There are essentially two types of constraints missing: domain restrictions and exclusion of certain (recursive) definitions. An example of a missing domain restriction would be to specify that the *participation* of a *role* is either *total* or *partial* (or that the *role* in which a weak entity type participates in an *identifying Relationship* must be *total*, the names of roles and attributes of an relation must be disjoint). An example of an EER construction that should be forbidden would be if an entity-type is *owner* of itself (or if a *subtype* is its own *supertype*, two supertypes in a generalization do not have a common root). Another constraint would be that each relationship type must have at least two roles attached. Such constraints (we call them application specific constraints as opposed to the general constraints) can be easily added.

```
...
not:  a relationship has two or more roles
...
-------------------------------------------------
application specific constraints hold for METAEER

Role some-role-name1 the-rel some-entity1 some-c1 some-p1
Role some-role-name2 the-rel some-entity2 some-c2 some-p2
not:  some-role-name-1 equals some-role-name2
-----------------------------------------------------------
a relationship has two or more roles

general integrity constraints hold
application specific constraints hold for some-application
-----------------------------------------------------------
database for some-application consistent
```

6 Using a Meta EER Diagram to Reason about the Consistency of the Meta EER Model

As demonstrated above one can easily derive from the observation information of the meta EER diagram the schema information of the application EER diagram. But since the meta EER diagram itself is an EER diagram it has the same schema as the application EER diagram.

In order to check the consistency between the application schema and its data it is only necessary to store the observation information and to use deduction rules like the rule above.

Thus, none of the instances of the one-to-one mapping introduced in an earlier section must be available in the final system, since all of these sentences might be formulated as deduction rules based on observations. One could even rewrite the integrity constraints introduced above to access the observation information of the schema directly which would make the deduction rules for *Attribute* and the like unnecessary.

If the meta EER diagram is specified in terms of instances (using the *Observation* representation) of the meta EER schema, the same general and application specific integrity rules (as introduced the previous section) can be applied to check the wellformedness of the meta EER model.

The relationships between meta EER model, application EER model, and application data in our approach can be summarized as follows:

A typical application consists of schema information (one-to-one mapping) and corresponding data in form of observations:

(1) $schema_{application\ EER} + observations_{application\ EER}^{describing\ application\ data} = application\ program$

By using the general integrity constraints it is possible to check the data against the schema. In order to check the wellformedness of the application EER diagram the meta EER diagram can be used where the application EER diagram is given in form of observations for the meta EER diagram:

(2) $schema_{meta\ EER} + observations_{meta\ EER}^{describing\ application\ EER} = schema_{application\ EER}$

(3) $schema_{meta\ EER} + observations_{meta\ EER}^{describing\ application\ EER} + observations_{application\ EER}^{describing\ application\ data} = application\ program$

In order to check the wellformedness of the meta EER diagram the meta EER diagram itself can be expressed in terms of observations:

(4) $schema_{meta\ EER} + observations_{meta\ EER}^{describing\ meta\ EER} = schema_{meta\ EER}$

(5) $observations_{meta\ EER}^{describing\ meta\ EER} + observations_{meta\ EER}^{describing\ application\ EER} + observations_{application\ EER}^{describing\ application\ data} = application\ program$

Item (5) shows that in principle the whole application could be specified only in terms of observations plus a single set of EER specific integrity rules. However, when the system is maintained "*manually*", it appears to be very hard to distinguish the various abstraction layers and to comprehend the observations. To reduce this disadvantage a simple set of rules can be given which deduces the representation of the one-to-one mapping from a set of observations.

7 Conclusion and Future Work

We presented in this paper a set of general integrity constraints for the EER model which are implemented using stratified datalog programs. Integrity checking is however the most naive approach to exploit the integrity information, since it might be too costly for reasonably sized databases. A large improvement in performance could be achieved, when only the relevant integrity constraints are tested on each update.

As pointed out in an earlier section it would be desirable not only to check the whole database, but instead to compute the set of consistent tuples or observations, ignoring the

invalid (or incomplete) information during the computations of an application. As a consequence either a layering of the integrity constraints must be introduced (eg. compute first the set of consistent observations using only integrity-constraint-1, apply integrity-constraints-2 on its results, and so on), or to specify the integrity constraints recursively, which leads to non-stratified knowledge bases. The disadvantage of the first approach is that the layering of the integrity constraints might be very hard (for n integrity rules exist $n!$ different layerings), the disadvantage of the second approach is that most implemented deduction methods rely on stratified programs.

A problem of a complete other nature is the missing modularity of our approach. When several EER models are kept in a single knowledge base (eg. a meta EER model and several application EER models), the user has to care that the names of the modelled types do not interfere. Our integrity checking rules can detect many clashes, but the maybe cleaner approach is to introduce an EER diagram name, which can be specified as an additional argument in the *Observation* facts.

Although our approach has several shortcomings, we think it might lead to better understanding of EER modeling and of integrity checking in general, and that our system is a very powerful prototyping system and case designer, where traditional relational databases could be easily integrated.

References

[ABW87] C. Apt, H. Blair, A. Walker: "*Towards a Theory of Declarative Knowledge*", in Minker (ed.): "*Foundations of Deductive Databases and Logic Programming*", Morgan Kaufmann, Los Altos 1987.

[Che76] P. Chen: "*The Entity Relationship Model – Toward a Unified View of Data*", ACM Transactions on Database Systems, 1:1, March 1976.

[Che91] W.C. Cheng: "*Tgif 2.6 - A Xlib based drawing facility under X11*", available via anonymous ftp from export.lcs.mit.edu, May 1991.

[Cod79] E. Codd: "*Extending the Database Relational Model to Capture More Meaning*", Transactions on Database Systems, 4:4, December 1979.

[DZ88] P.W. Dart, J. Zobel: "*Conceptual Schemas Applied to Deductive Database Systems*", Information Systems, Vol. 13, 1988.

[EN89] R. Elmasri, S.B. Navathe: "*Fundamentals of Database Systems*", Benjamin/Cummings, Redwood City 1989.

[EWH85] R. Elmasri, J. Weeldreyer, A. Hevner: "*The Category Concept: An Extension to the Entity-Relationship Model*", International Journal on Data and Knowledge Engineering, 1:1, May 1985.

[GMN84] H. Gallaire, J. Minker, J. Nicolas: "*Logic and Databases: A Deductive Approach*", ACM Computing Surveys 16, 2, June 1984.

[GV89] G. Gardarin, P. Valduriez: "*Relational Databases and Knowledge Bases*", Addison-Wesley, Reading 1989.

[KN91] N. Kehrer, G. Neumann: "*Treating Enhanced Entity Relationship Models in a Declarative Style*", in: Proceedings of the "*2nd Russian Conference on Logic Programming*", September 11-16, 1991, Leningrad (Proceedings will be published in LNCS 1992).

[KMS90] G. Kiernan, C. de Maindreville, E. Simon: "*Making Deduktive Databases a Practical Technology: A Step Forward*", in: Proceedings of the ACM SIGMOD, Atlantic City, USA, May 1990.

[Mai83] D. Maier: "*The Theory of Relational Databases*", Computer Science Press, Rockville 1983.

[MS89] V. M. Markowitz, A. Shoshani: "*On the Correctness of Representing Extended Entity-Relationship Structures in the Relational Model*", in: J. Clifford, B. Lindsay, D. Maier (eds.): "*Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*", ACM, New York 1989.

[NT89] S. Naqvi, S. Tsur: "*A Logic Language for Data and Knowledge Bases*", Computer Science Press, New York 1989.

[TYF86] T.J. Teorey, D. Yang, J.P. Fry: "*A logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model*", ACM Computing Surveys 18, 2, June 1986.

[Teo90] T.J. Teorey: "*Database Modeling and Design: The Entity-Relationship Approach*", Morgan Kaufmann, San Mateo 1990.

[SS77] J. Smith, D. Smith: "*Database Abstractions: Aggregation and Generalization*", Transactions on Database Systems, 2:2, June 1977.

[WMSW90] A. Walker (ed.), M. McCord, J.F. Sowa, W.G. Wilson: "*Knowledge Systems and Prolog*", 2nd Edition, Addison-Wesley, Reading 1990.