Deriving Identity from Extensionality

A.H.M. ter Hofstede and Th.P. van der Weide

Published as: A.H.M. ter Hofstede and Th.P. van der Weide. Deriving Identity from Extensionality. Technical Report CSI-R9416, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, December 1994.

***Abstract*— In recent years, a number of proposals have been made to extend conventional conceptual data modeling techniques with concepts for modeling complex object structures. Among the most prominent proposed concepts is the concept of collection type. A collection type is an object type of which the instances are sets of instances of an other object type. A drawback of the introduction of such a new concept is that the formal definition of the technique involved becomes considerably more complex. This is a result of the fact that collection types are populatable types and such types tend to complicate updates. In this paper it is shown how a new kind of constraint, the extensional uniqueness constraint, allows for an alternative treatment of collection types avoiding update problems. The formal definition of this constraint type is presented, other advantages of its introduction are discussed, and its consequences for, among others, identification schemes are elaborated.**

***Keywords*— Conceptual Data Modeling, Uniqueness constraint, Extensional uniqueness constraint, Identification, Collection type, ER, NIAM.**

Classification: AMS 68P15, CR H.2.1

I. Introduction

Nowadays it is commonly accepted that information systems are best specified at the conceptual level first. Usually, two perspectives are distinguished at this level, the data and the process perspective. It is generally recognized that the data perspective tends to be more stable than the process perspective, and that a precise model of this perspective is of vital importance for successful implementation of the system under consideration.

Many techniques for conceptual data modeling exist ([12; 16]). Among the most well-known is the Entity Relationship approach, see e.g. [3], and its many variants. Other approaches are functional data modeling techniques, such as FDM [20], and object role modeling techniques, such as NIAM [15].

Dept. of Information Systems, University of Nijmegen, Toernooiveld 1, NL-6525 ED Nijmegen, The Netherlands, {arthur,tvdw}@cs.kun.nl

Complex application domains, such as hypermedia, CAD/-CAM, meta-modeling, and office automation, have led to the introduction of advanced modeling concepts, such as present in the various forms of Extended ER (see e.g. [21; 5]), IFO [1], and object-role modeling extensions such as FORM [6] and PSM [11; 9].

One of the most fundamental concepts in many of these extensions seems to be the *collection type*. A *collection type* is an object type of which the instances correspond to (nonempty) sets of an other object type, referred to as its *element type*. Collection types correspond to *grouping* in IFO, *association* in ECR [5], *grouping classes* in SDM [7], and *power types* in PSM. In [7] the *Convoy Problem* is introduced as an illustration of the application of a collection type. Convoys are simply sets of ships and do not have some external form of identification, e.g. a convoy code. This can be modeled with a collection type *convoy* having element type *ship*.

Unfortunately, the introduction of collection types tends to complicate the formal definition of a technique considerably. Next to relationship types, collection types also are populatable object types. This may lead to update problems. As an example consider the PSM schema of figure 1, where A is a collection type with element type B. Collection type A has an instance $\{b_1, b_2, b_3\}$, which participates in three relationship types (f, g, and h). If one would want to add an instance, e.g. b_4, to this set, the instantiations (populations) of three relationship types are affected.

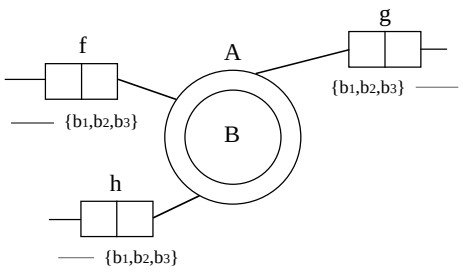

Fig. 1. The update problem for collection types

A solution to this update problem may be found in the separation of properties and representations. Consider the schema of figure 2. In this schema the element-of relation is explicitly represented. The associated relationship type $\in$ captures the fact that a_1 corresponds to the set $\{b_1, b_2, b_3\}$. Addition of an instance to this set only leads to a change

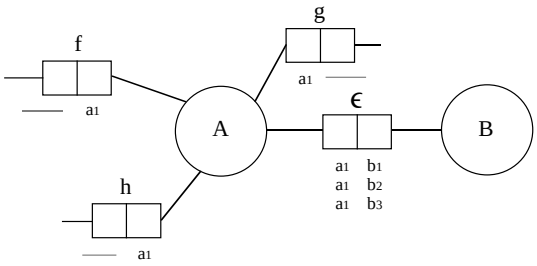

Fig. 2. A solution for the update problem for collection types

in the population of this relationship type, the populations of the other relationship types remain unaffected. In order to fully capture the fact that A corresponds to a collection type, it is necessary to impose a constraint on relationship type $\in$ to avoid that another instance a_2 exists that represents the same set as a_1. Such a constraint is called an *extensional uniqueness constraint*.

The focus of this paper is on the formalization and applications of extensional uniqueness constraints. Firstly, they allow for an alternative treatment of collection types, avoiding update problems. Further, they facilitate transformations of conceptual data models to internal models that do not support collection types directly (e.g. the relational model). In addition to that, they narrow the gap between the world of conceptual data models and the world of OO. This is a direct result of the fact that the extensional uniqueness constraint allows a focus on properties instead of on representations guaranteeing those properties. Consider the schema of figure 2 again. Collection type A does not contain sets anymore, but elements with set behavior. As such, the sets have received their own identity (as in OO) and structure conflicts can be avoided.

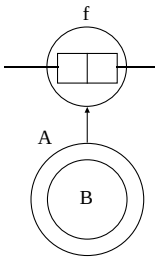

Fig. 3. Collection type which is subtype of a relationship type

To illustrate this latter point consider the schema of figure 3, where collection type A is a subtype of relationship type f. As the instances of f are tuples and the instances of A sets, this is not ordinarily possible. In the schema of figure 4, A is a subtype of f again, but here there are no structure conflicts. Instances of A may be viewed as sets of instances of B as well as special instances of f. The extensionality constraint needed on the role attached to object type B guarantees that no two instances of object type A have the same members and hence represent identical sets. As such, this constraint guarantees a unique identification for each instance of A.

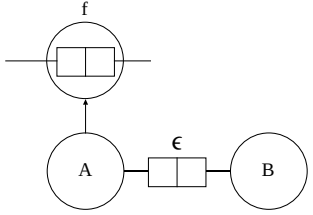

Fig. 4. Subtype with collection type behavior

The paper is organized as follows. In section II the nesting join operator is introduced which plays a central role in the definition of the extensional uniqueness constraint in section III. In section IV the implications of the alternative treatment of collection types by the use of extensional uniqueness constraints for identification are discussed. It also turns out that this new type of constraint allows for more powerful identification schemes, useful in complex modeling domains. In sections V and VI the formal consequences for type relatedness and populations are treated. Section VII contains conclusions and suggestions for further research.

Finally, it should be remarked that in this paper the graphical conventions and the formal foundation of PSM will be employed. The approach is however sufficiently general to be applicable to other conceptual data modeling techniques as well.

II. The nesting join operator

In data modeling techniques (such as ER, NIAM and PSM) many types of constraints can be specified graphically. Sometimes these constraints go beyond the boundaries of a single relationship type. In these cases, the semantics of the constraint is an expression over a special combination of the relationship types involved. The nesting join operator ξ has been introduced for the specification of this combination. The rest of this section illustrates the working of this operator, and its application in the context of uniqueness constraints. For a more detailed formal description of this operator refer to [22] and [8].

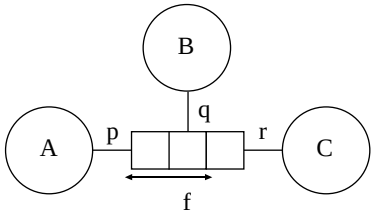

Fig. 5. Uniqueness constraint over a single relationship type

Figure 5 contains a single (ternary) relationship type f. Its roles are p, q, and r, which are played by A, B, and C respectively. The roles p and q are subject to a uniqueness constraint. This constraint states that instances of relationship type f are uniquely determined by the combination of their p and q components. In other words, r is

functionally dependent on $\{p, q\}$. As a result, $\{p, q\}$ forms a candidate key of relationship type f. This is formally expressed as the following functional dependency (identifying f with the set of its roles):

$$\{p, q\} \xrightarrow{f} f$$

Generally, a uniqueness constraint τ is a nonempty set of roles. A population (instantiation) Pop of a data model satisfies a uniqueness constraint over τ, $\mathsf{Pop} \models \mathsf{unique}(\tau)$, if and only if $\mathsf{Pop} \models \mathsf{identifier}(\xi(\tau), \tau)$, meaning that τ is a key for the derived relationship type $\xi(\tau)$ in population Pop.

As stated, $\xi(\tau)$ represents a derived relationship type. Derived relationship types are specified by means of relational algebra operators (see for example [19; 2; 8]). On each relational expression two functions can be applied, Val and Schema. The expression $\mathsf{Val}[\![r]\!](\mathsf{Pop})$ yields the values that occur in relational expression r in population Pop. The expression $\mathsf{Schema}(r)$ yields the schema of relational expression r, a set of roles. An instance in $\mathsf{Val}[\![r]\!](\mathsf{Pop})$ is a function from $\mathsf{Schema}(r)$ to Ω, i.e. the set of instances of all domains. This corresponds to the mapping oriented approach.

In determining $\xi(\tau)$, three cases can be distinguished. These cases are treated subsequently by means of examples.

Single relationship type

If a uniqueness constraint τ does not exceed the boundaries of a single relationship type f (i.e. $\tau \subseteq f$), then $\xi(\tau) = f$. In this situation, τ is called a *key* of relationship type f. A relationship type can have several keys.

In figure 5, the uniqueness constraint $\tau = \{p, q\}$ is restricted to relationship type f. This uniqueness constraint excludes the following population of relationship type f:

$\mathsf{Pop}(f)$

p	q	r
a	b	c_1
a	b	c_2

Joinable via related object types

If more than one relationship type is involved in a uniqueness constraint, these relationship types should be *joinable via related object types*. If these relationship types can be joined via related object types, they are joined by the join operator $\bowtie$ and the selection operator σ.

Object types are called (type) related, if they can share instances in a population. Type relatedness can be statically determined (see section V).

As an example of this situation consider figure 6. In this figure, object type B is a generalization of object types C and D, which means that the population of object type B is the union of the populations of object types C and D (generalization is discussed in more detail in section IV).

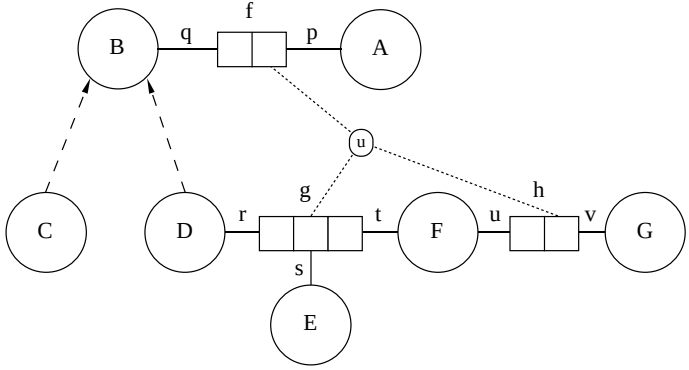

Fig. 6. An example uniqueness constraint

Consequently, D is type related with B, as these object types may share instances.

This schema contains uniqueness constraint $\tau = \{p, s, v\}$. The condition $\mathsf{unique}(\tau)$ requires that τ is a key of

$$\xi(\tau) = \sigma_{q=r \wedge t=u}(f \bowtie g \bowtie h)$$

An example of a population of the relationship types f, g, and h that is excluded by this uniqueness constraint is:

$\mathsf{Pop}(f)$

p	q
a	d_1
a	d_2

$\mathsf{Pop}(g)$

r	s	t
d_1	e	f_1
d_2	e	f_2

$\mathsf{Pop}(h)$

u	v
f_1	g
f_2	g

Uniqueness and objectification

The most complex type of uniqueness constraint involves objectification, i.e. the treatment of a relationship type as an object type. Objectification therefore allows relationship types to participate in other relationship types. This corresponds to the concept of nested relations in the NF^2 data model ([19]).

In figure 7 an example of this type of uniqueness constraint is depicted. In this example, g and h are nested relationship types. The relational operator η is the *strong unnest operator*. The expression $\eta^r(g)$ describes a derived relationship type that will contain instances from g where the r-component is replaced by the respective p- and q-components.

As $\xi(\{p, s, u\}) = \eta^r(g) \bowtie \eta^t(h)$, the semantics of the uniqueness constraint is:

$$\mathsf{identifier}\left(\eta^r(g) \bowtie \eta^t(h), \{p, s, u\}\right)$$

A population of the relationship types f, g, and h, that is excluded by this uniqueness constraint is:

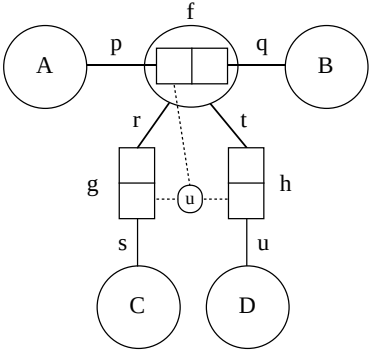

Fig. 7. A uniqueness constraint over objectification

$\mathsf{Pop}(f)$

p	q
a_1	b_1
a_1	b_2

$\mathsf{Pop}(g)$

r		s
p	q	
a_1	b_1	c_1
a_1	b_2	c_1

$\mathsf{Pop}(h)$

t		u
p	q	
a_1	b_1	d_1
a_1	b_2	d_1

As an example of the occurrence of this uniqueness constraint in a concrete example, consider figure 8. The schema in this figure models that *Parts* are used in *Projects* and that *Parts* can be supplied for particular *Projects* by a *Supplier* with a certain *Priority*. Informally, the uniqueness constraint states that no two different *Projects* use the same *Part* supplied by the same *Supplier*, with the same *Priority*.

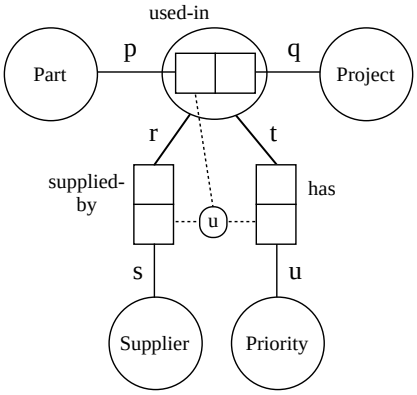

Fig. 8. A schema dealing with parts and suppliers

III. The extensional uniqueness constraint

In this section the various forms of the extensional uniqueness constraint and their semantics are discussed.

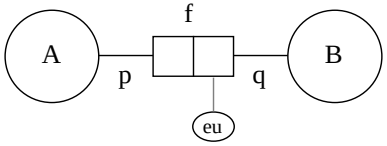

Fig. 9. The simplest case

In figure 9, the simplest form of an extensional uniqueness constraint is depicted. In this schema no two instances of object type A can be associated via f to the same set of instances of object type B. Formally, this constraint $\mathsf{extuniq}(\{q\})$ is satisfied by a population Pop of this information structure, denoted as $\mathsf{Pop} \models \mathsf{extuniq}(\{q\})$ iff:

$$\forall_b \left[\{p:x, q:b\} \in \mathsf{Pop}(f) \Leftrightarrow \{p:y, q:b\} \in \mathsf{Pop}(f)\right] \Rightarrow x = y$$

Note the similarity with the Axiom of Extensionality of classic set theory (see e.g. [4]):

$$\forall_z \left[z \in x \Leftrightarrow z \in y\right] \Rightarrow x = y$$

A population of relationship type f in figure 9 that is excluded by the extensional uniqueness constraint is e.g.:

$\mathsf{Pop}(f)$

p	q
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_2

In this population both a_1 and a_2 would correspond to the set $\{b_1, b_2\}$.

A collection type A over element type B (i.e. instances of A are collections of instances of B) can now be modeled as in the schema of figure 9. Figure 10 shows how the Convoy Problem can be modeled using the extensional uniqueness constraint. The dot on role *contains* is an example of a *total role constraint* and expresses that participation in this role is mandatory. Figure 11 captures the case when no two convoys may share ships. One should realize that these schemata only capture the Convoy Problem completely if the identification of *Convoy* can be realized through *Ship*. This would avoid the introduction of an artificial convoy code to identify convoys. This subject is addressed in section IV.

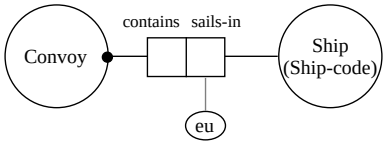

Fig. 10. Possible translation of the Convoy Problem

A more complex situation is the situation in which several roles of the same relationship type are involved in an extensional uniqueness constraint. As a concrete example consider figure 12. Multisets (see e.g. [13]), sometimes

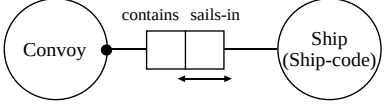

Fig. 11. Another possible translation of the Convoy Problem

referred to as bags, are completely characterized by their elements and their respective occurrence frequencies. This is expressed by the extensional uniqueness constraint.

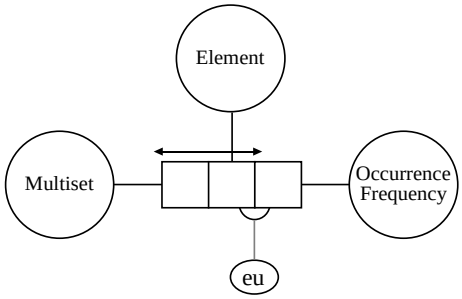

Fig. 12. Conceptual schema of multisets

Formally, if $\tau \subseteq f$, then the constraint $\mathsf{extuniq}(\tau)$ is satisfied for relationship type f in a population Pop iff for all tuples $x, y : (f - \tau) \to \Omega$:

$$\forall_{t:\tau\to\Omega}\,[x \cup t \in \mathsf{Pop}(f) \Leftrightarrow y \cup t \in \mathsf{Pop}(f)] \Rightarrow x = y$$

In general, the extensional uniqueness constraint can be specified for role sets that would constitute a syntactically valid uniqueness constraint. Consider for example figure 13. In this figure, part of the meta-model of elementary Petri nets (see e.g. [18; 17]) is depicted. Petri nets can be considered bipartite directed multigraphs. Places can be connected via multiple edges to transitions. A *marked* net is a Petri net where tokens are assigned to places. A transition can *fire* iff for each arrow to the transition a token is available in the corresponding place. In the resulting marking from each place with an input arrow to the transition tokens are removed (as many as there are input arrows from that place to the transition) while to each place with an output arrow from the transition tokens are added (as many as there are output arrows from the transition to that place). Obviously, two transitions having for each place the same number of input and output arrows, can only cause the same state changes. Hence the extensional uniqueness constraint in figure 13, excluding this situation.

For the formal semantics of the extensional uniqueness constraint in general, the nesting join operator ξ can be employed. A population Pop satisfies a uniqueness constraint $\mathsf{extuniq}(\tau)$, $\mathsf{Pop} \models \mathsf{extuniq}(\tau)$, iff for each $x, y : (\mathsf{Schema}(\xi(\tau)) - \tau) \to \Omega$:

$$\forall_{t:\tau\to\Omega}\,[x \cup t \in \mathsf{Val}[\![\xi(\tau)]\!]\,(\mathsf{Pop}) \Leftrightarrow y \cup t \in \mathsf{Val}[\![\xi(\tau)]\!]\,(\mathsf{Pop})] \Rightarrow x = y$$

As the extensional uniqueness constraint can have the same syntactic appearance as the uniqueness constraint (i.e. be

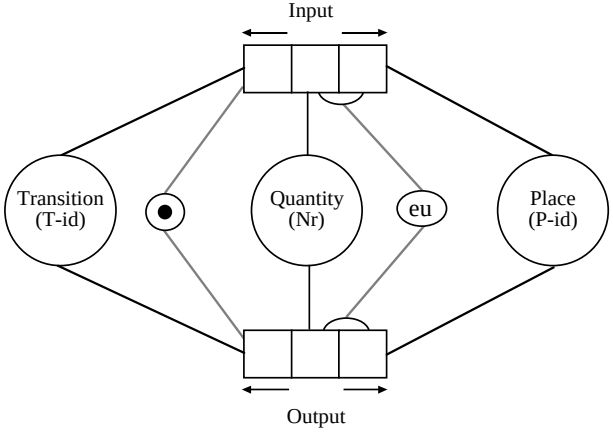

Fig. 13. Metamodel of elementary Petri nets

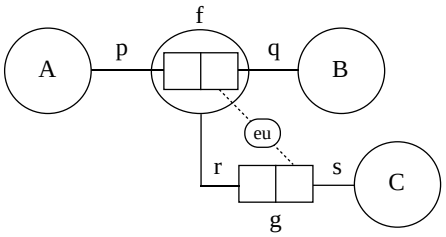

Fig. 14. Abstract example of complex extensional uniqueness constraint

specified for the same sets of roles), an extensional uniqueness constraint may also be specified for objectified structures. An abstract example is shown in figure 14. A population Pop of relationship type g that would be excluded by this constraint is:

$\mathsf{Pop}(g)$

r		s
p	q	
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_1	c_1
a_2	b_2	c_2

The following trivial lemma states that if a population satisfies a uniqueness constraint over a set of roles τ it also satisfies an extensional uniqueness constraint over τ in that population.

Lemma III.1 $\mathsf{Pop} \models \mathsf{unique}(\tau) \Rightarrow \mathsf{Pop} \models \mathsf{extuniq}(\tau)$

IV. Identification

In the previous section the connection between collection types and extensional uniqueness constraints has been discussed. In order to be able to make the concept of collection type expressible in terms of the extensional uniqueness constraint, a more powerful identification scheme than usually present in conceptual data modeling techniques is needed. In addition to that, the definition of type relatedness has to be modified. This section deals with identification, the next section will handle type relatedness.

A. Weak Identification

In many conceptual data modeling techniques, a distinction exists between objects that can be represented directly and objects that cannot be represented directly. In ER, this distinction is reflected by the difference between entities and attributes, while in NIAM and PSM this distinction corresponds with the difference between entities and labels. Labels can be represented directly on a communication medium, while other kinds of objects depend for their representation on labels.

In general, identification deals with representability. Instances are representable if they can be denoted uniquely in terms of labels. If two different objects have identical properties, and are therefore connected (directly or indirectly) to the same labels, then they cannot be distinguished. A population in which objects with the same properties are equal is called *weakly* identified.

Weak identification is a property of populations. Structural identification is a property of schemata and guarantees that *each* population is weakly identified. A schema Σ is called structurally identifiable $\mathsf{StructId}(\Sigma)$, if all object types can be identified:

$$\forall_{x \in \mathcal{O}} \, [\mathsf{Idf}(x)]$$

The predicate Idf is defined using derivation rules. An object type is identifiable if and only if this can be proved from these derivation rules.

B. Rules for Structural Identification

The derivation rules for structural identification distinguish between the various object types in a conceptual data modeling technique. The system of rules guarantees the existence of so-called *identificators* for each object type. An identificator for a certain object type is a set of roles with distinguishing properties for the instances of that type. Identificators play a crucial role in the denotation of instances of object types, two different objects must have different denotations. An identificator can be used for a denotation scheme. An object type may have several identificators and for denotation purposes any choice is acceptable.

Label types

Label types are object types of which the instances originate from concrete domains. Consequently, instances of label types are directly representable and no explicit identificators are needed (the identificator of a label type may be considered to be the empty set). Therefore, label types are structurally identifiable.

[IDT1] $\quad x \in \mathcal{L} \vdash \mathsf{Idf}(x)$

where $\mathcal{L}$ is the set of label types.

Inheriting types

Many conceptual data modeling techniques offer concepts for expressing subtype relations. Subtype relations are used to capture inheritance of properties. In the literature many types of inheritance relations exist and the terminology is far from standard. In this section two important types of inheritance relations are considered: specialization and generalization. Many conceptual data modeling techniques contain at least one of these relations, although probably under a different name. The concepts of specialization and generalization in this paper correspond to a large extent to specialization and generalization as defined in IFO [1].

Specialization is used when specific facts are to be recorded for specific instances of an object type only. For example, in a data model with persons, one may wish to express that only for adults, i.e. persons at least 18 years old, the cars they own are to be recorded. In that case *Adult* becomes a subtype of object type *Person*, and only instances of *Adult* can participate in the relationship registrating car ownership. A specialized object type inherits the properties of its supertype(s), but may have additional properties. *Subtype defining rules* serve as decision criteria for determining which instances of supertypes participate in subtypes. Therefore, they are to be considered population derivation rules. The availability of such rules allows subtypes to be viewed as derivable object types.

A subtype is structurally identifiable if and only if all the object types needed for the evaluation of its subtype defining rule as well as all its supertypes are structurally identifiable:

[IDT2] $\quad \mathsf{spec}(x) \wedge \forall_{y \in \mathcal{O}} \, [x \, \mathsf{Spec} \, y \vee x \, \mathsf{Dep} \, y \Rightarrow \mathsf{Idf}(y)] \vdash \mathsf{Idf}(x)$

where $x \, \mathsf{Dep} \, y$ expresses that y is an object type needed for the evaluation of the subtype defining rule of x (see [8]), $x \, \mathsf{Spec} \, y$ is to be interpreted as "x is a specialization of y", and $\mathsf{spec}(x)$ indicates whether x is a subtype.

The identification derivation rule for subtypes captures the fact that a subtype inherits its identification from its supertypes. However, once a subtype is proved identifiable one may wish to use a different denotation scheme for its instances than the one used for its supertypes. The approach described in [10] allows alternative denotation schemes for subtypes.

Generalization is a mechanism that allows for the creation of new object types by uniting existing object types. Contrary to what its name suggests, generalization is *not* the inverse of specialization. Specialization and generalization originate from different axioms in set theory [11].

The population of a generalized object type is the union of the populations of the participating object types, referred to as the *specifiers*. Typically, properties are propagated "upward" in a generalization hierarchy instead of "downward" (see also [1]). This also implies that the identifica-

tion of a generalized object type depends on the identification of its specifiers.

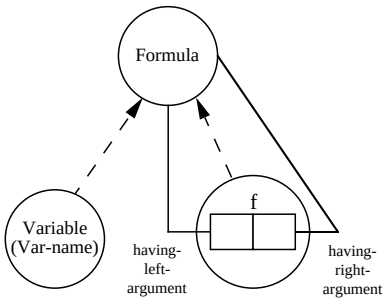

Fig. 15. An example of generalization in PSM

As an example of generalization consider the schema of figure 15. This PSM schema models the construction of simple formulas: a *Formula* may be either a *Variable* or constructed by some function f from simpler formulas. This example demonstrates that generalization can be used for the specification of recursive types. Generalization is also useful when identical properties are relevant for different existing types: these properties can then be related to the generalization of these types.

A generalized object type inherits its identification from *some* of its specifiers. The rationale is that in some cases the identification of a specifier depends on the identification of the generalized type (consider for example relationship type f in the schema of figure 15). Therefore, it is not required that a generalized type is identifiable if and only if *all* its specifiers are identifiable. Formally:

[IDT3] $\mathsf{gen}(x) \wedge \exists_{y \in \mathcal{O}} [x \mathsf{\ Gen\ } y \wedge \mathsf{Idf}(y)] \vdash \mathsf{Idf}(x)$

where $\mathsf{gen}(x)$ expresses that x is a generalized type and $x \mathsf{\ Gen\ } y$ is to be interpreted as "x is a generalization of y".

Entity types

An entity type is structurally identifiable if and only if a set of roles can be found that relate instances of that entity type to unique combinations of other instances.

[IDT4] $e \in \mathcal{E} \wedge \exists_{\tau \subseteq \mathcal{R}} [\mathsf{Identification}(e, \tau)] \vdash \mathsf{Idf}(e)$

where $\mathcal{E}$ is the set of entity types and $\mathcal{R}$ the set of roles. A set τ of roles is an identificator for entity type e, denoted as $\mathsf{Identification}(e, \tau)$, iff:

1. $\vdash \mathsf{extuniq}(\tau)$

   The set of roles τ is to serve as an identification for entity type e. Instances of e can then be denoted by means of their specific combination of τ values. To this end, the combination of τ values should be (at least) extensionally unique for every instance of e.

2. $\forall_{p \in \tau} \exists!_{q \in \mathsf{Rel}(p)} [q \notin \tau]$

   Object types which are not needed for the identification of e should not participate in the relationship types of τ ($\mathsf{Rel}(p)$ yields the relationship type in which role p occurs). This means that each relationship type from τ contains exactly one role outside τ. This unique role is denoted as $\mathsf{co}(\tau, p)$. The object type playing this role, referred to as the base of the role, should be entity type e: $\mathsf{Base}(\mathsf{co}(\tau, p)) = e$.

3. $\vdash \mathsf{total}(\{\mathsf{co}(\tau, p) \mid p \in \tau\})$

   Each instance of e should participate in at least one of the coroles of τ, which can be enforced if a total role constraint is provable on this set of coroles. If this were not the case, then instances of e, not participating in any of the relationship types of τ, might be indistinguishable.

4. $\forall_{p \in \tau} [\mathsf{Idf}(\mathsf{Base}(p))]$

   The bases of the roles in τ have to be identifiable.

As examples consider figures 10 and 11. In both cases, entity type *Convoy* can be identified via entity type *Ship*. In the schema of figure 12, instances of entity type *Multiset* could be identified by their elements and corresponding occurrence frequencies if participation in the ternary relationship type would be mandatory. Now, several instances representing the empty multiset may exist.

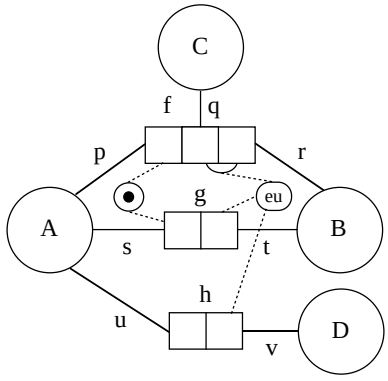

Fig. 16. Abstract sample schema with complex identification

Example IV.1

As a more elaborate example consider figure 16. Assuming that the object types B, C, and D are identifiable, A is also identifiable as the set $\{q, r, t, v\}$ then is a valid identificator for this entity type. Note that $\mathsf{total}(\{p, s\}) \vdash \mathsf{total}(\{p, s, u\})$. □

Example IV.2

The schema of figure 17 contains a concrete example of complex identification. An assembly is uniquely determined by the combination of participating students, staff members, and the chairperson. The chairperson has to be a participating staff member, a condition

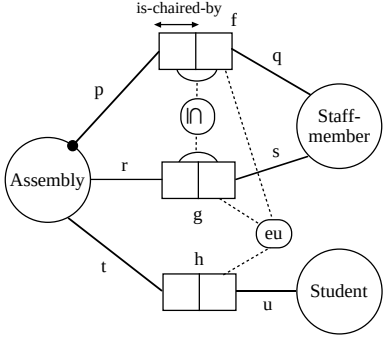

Fig. 17. Concrete sample schema with complex identification

which is captured by the subset constraint in figure 17. The notion of assembly might be of interest in order to express that certain topics should not be discussed in certain companies. The introduction of a special code for assemblies then may be artificial. □

Conceivably, a particular entity type e could have several identificators. As the denotation of entities depends on the identificator chosen, a standard identificator τ_e has to be chosen for every entity type e. In addition to that, a denotation requires an ordering of the roles of the standard identificator. The function $\mathsf{Idn} : \mathcal{E} \rightarrow \mathcal{R}^+$ records this ordering. Consequently, $\tau_e = \mathsf{set}(\mathsf{Idn}(e))$. For entity type e, $\mathsf{Idn}(e)$ will be referred to as its *abstract denotator*. If the associated identificator is extensionally unique, the abstract denotator is referred to as *compound*, otherwise, it is referred to as *simple*. Only in denotations of instances of entity types with compound abstract denotators sets can occur.

Example IV.3

Consider again the schema of figure 17. Let the abstract denotator for entity type Assembly *be* $\langle q, s, u \rangle$. *An assembly consisting of the students* s_1, s_2, *and* s_3, *the staff members* m_1 *and* m_2, *chaired by* m_2, *can then be denoted as:*

$$\langle m_2, \{m_1, m_2\}, \{s_1, s_2, s_3\} \rangle$$

The abstract denotator of Assembly *is a compound denotator as the associated identificator* $\{q, s, u\}$ *is extensionally unique.* □

Relationship types

Relationship types, which formally correspond to sets of roles, depend for their identification on the bases of these constituent roles:

[IDT5] $f \in \mathcal{F} \wedge \forall_{p \in f} [\mathsf{Idf}(\mathsf{Base}(p))] \vdash \mathsf{Idf}(f)$

where $\mathcal{F}$ is the set of relationship types. As such, a relationship type can be considered to be its own identificator.

V. Type relatedness

Intuitively, object types can, for several reasons, have values in common in some instantiation. For example, each value of object type x is, in any instantiation, also a value of each of its supertypes. If x Gen y, then any value of y in any population is also a value of x. A third example, where object types may share values is when two collection types have element types that may share values. In this section, this is formalized in the concept of *type relatedness*. As a consequence, only type related object types may have values in common. Type relatedness plays an important role in query optimization [9].

Formally, type relatedness is captured by a binary relation $\sim$ on $\mathcal{O}$. Two object types are type related if and only if this can be proved from a number of derivation rules. As a result of this, type relatedness can be statically determined, i.e. at "compile time". The following derivation rules, taken from [11], are valid for PSM schemata:

[T1] $\vdash x \sim x$

[T2] $x \sim y \vdash y \sim x$

[T3] $x \,\mathsf{Spec}\, y \wedge y \sim z \vdash x \sim z$

[T4] $y \,\mathsf{Spec}\, x \wedge y \sim z \vdash x \sim z$

[T5] $x \,\mathsf{Gen}\, y \wedge y \sim z \vdash x \sim z$

The rule for collection types in [11], stating that two collection types are type related iff their element types are type related, is omitted. As in our approach collection types are modeled as entity types, we need an extra rule for type relatedness between entity types.

Two entity types x and y are type related iff they have (component-wise) type related abstract denotators $\mathsf{Idn}(x)$ and $\mathsf{Idn}(y)$, which are either both simple or both compound. The rule for type related entity types can therefore be formulated as:

[T6] $\mathsf{Idn}(x) \sim \mathsf{Idn}(y) \wedge (\mathsf{unique}(\tau_x) \Leftrightarrow \mathsf{unique}(\tau_y)) \vdash x \sim y$

As an example of two type related entity types consider figure 18. In this schema, both entity type A and entity type B are identified via a role having as base D and for which an extensional uniqueness constraint is specified. Their abstract denotators are:

$$\begin{aligned} \mathsf{Idn}(A) &= \langle q \rangle \\ \mathsf{Idn}(B) &= \langle s \rangle \end{aligned}$$

Note that both are compound abstract denotators. Instances of both A and B can therefore be seen as sets of instances of object type D. Consequently, A and B are type related. Entity type C has a simple abstract denotator:

$$\mathsf{Idn}(C) = \langle u \rangle$$

Instances of object type C correspond to a single instance of object type D. As a consequence, object types A and D are not type related.

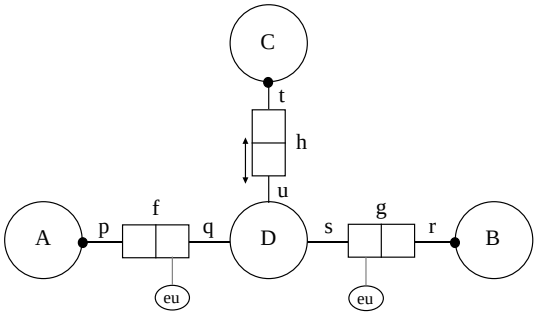

Fig. 18. Example of type related entity types

VI. The population identification rule

As a result of the inheritance of identity, objects can be instance of more than one object type. For example, the population rules for subtyping require subtypes to be a subset of their supertypes. Note that this relation between objects and object types is dynamic. For example, by changing the properties of an object, this object may become a member of a subtype. As a consequence of this approach, no rules are needed for expressing equality between objects.

The new rule for type relatedness between entity types requires an equality expressing rule. Instances of type related entity types are considered to be equal if they have the same abstract denotation. This is expressed by the *population identification rule*.

Let x and y be type related entity types, with identificators τ_x and τ_y. Let furthermore relational expression $\xi_x = \xi(\tau_x)$ have schema S_x. First it should be noted that all tuples t in ξ_x are homogeneous outside τ_x:

$$\forall_{p,q \in S_x - \tau_x} [t(p) = t(q)]$$

This unique value is an instance of object type x, its identifying properties are recorded in the τ_x-part of the corresponding tuples. Let p_x be any role from $S_x - \tau_x$. The identifying properties of instance $i_x \in \mathsf{Pop}(x)$ then are described by

$$\mathsf{Den}_x(i_x) = \pi_{\tau_x} \sigma_{p_x = i_x} \xi_x$$

Let ξ_y, S_y and p_y be analogously defined for object type y. Consequently, the abstract denotations of object types x and y are found in the derived relations ξ_x and ξ_y respectively. These relational expressions may however have a different underlying schema. Let $\phi : \tau_y \to \tau_x$ be the corresponding *denotator matcher*, i.e. the bijection that relates the identifying parts τ_x and τ_y by pairwise matching $\mathsf{Idn}(y)$ and $\mathsf{Idn}(x)$. The population identification rule enforces (in each population) the equality of objects with the same abstract denotation:

$$\mathsf{Den}_x(i_x) = \pi_\phi \, \mathsf{Den}_y(i_y) \Rightarrow i_x = i_y$$

The operator π is the usual projection operator, allowing roles to be renamed [14].

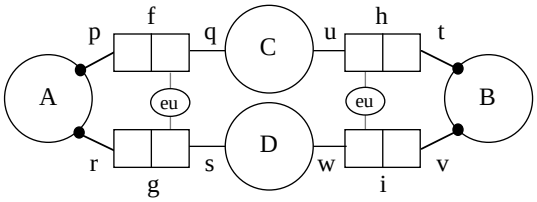

Fig. 19. Example identification

As an example of the application of this rule consider figure 19. In this schema entity type A is identified via the sequence of roles $\mathsf{Idn}(A) = \langle q, s \rangle$, while entity type B is identified by $\mathsf{Idn}(B) = \langle u, w \rangle$. The corresponding denotator matcher maps role u onto q and w onto s. Note that $\xi(\{q, s\}) = \sigma_{p=r}(f \bowtie g)$ and $\xi(\{u, w\}) = \sigma_{t=v}(h \bowtie i)$. The population identification rule in this case can be formulated as:

$$(\pi_{q,s} \sigma_{p=r=i_A} f \bowtie g = \pi_{q:u,s:w} \sigma_{t=v=i_B} h \bowtie i) \Rightarrow i_A = i_B$$

The following population Pop of the relationship types of this schema would not satisfy the population identification rule as $a_1 \neq b_1$ and a_1 and b_1 are connected to precisely the same instances for their identification:

$\mathsf{Pop}(f)$

p	q
a_1	c_1
a_1	c_2

$\mathsf{Pop}(g)$

r	s
a_1	d_1

$\mathsf{Pop}(h)$

t	u
b_1	c_1
b_1	c_2

$\mathsf{Pop}(i)$

v	w
b_1	d_1

VII. Conclusions

In this paper the extensional uniqueness constraint was introduced. It was shown how this type of constraint makes an alternative treatment of collection types avoiding update problems possible. The constraint allows for a separation between structure and properties and as such, also narrows the gap between conceptual data modeling and the OO approach (clearly, however, much more research in this direction is needed). In addition to that, it allows for less rigid identification schemes and facilitates translations to internal models not explicitly supporting collection types.

References

[1] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.

[2] P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, October 1991.

[3] P.P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[4] J.N. Crossley, C.J. Ash, C.J. Brickhill, J.C. Stillwell, and N.H. Williams. *What is mathematical logic?* Oxford University Press, Oxford, United Kingdom, 1972.

[5] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9(4):157–204, 1992.

[6] T.A. Halpin and M.E. Orlowska. Fact-oriented modelling for data analysis. *Journal of Information Systems*, 2(2):97–119, April 1992.

[7] M. Hammer and D. McLeod. Database Description with SDM: A Semantic Database Model. *ACM Transactions on Database Systems*, 6(3):351–386, September 1981.

[8] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[9] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, 1993.

[10] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Grammar Based Information Modelling. Technical Report CSI-R9414, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, October 1994.

[11] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.

[12] R. Hull and R. King. Semantic Database Modelling: Survey, Applications and Research Issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[13] A. Lew. *Computer Science: A Mathematical Introduction*. Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[14] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, Maryland, 1988.

[15] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia, 1989.

[16] J. Peckham and F. Maryanski. Semantic Data Models. *ACM Computing Surveys*, 20(3):153–189, September 1988.

[17] J.L. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice-Hall, Englewoods Cliffs, New Jersey, 1981.

[18] W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1985.

[19] H.J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[20] D.W. Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, March 1981.

[21] T.J. Teorey, D. Yang, and J.P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222, 1986.

[22] Th.P. van der Weide, A.H.M. ter Hofstede, and P. van Bommel. Uniquest: Determining the Semantics of Complex Uniqueness Constraints. *The Computer Journal*, 35(2):148–156, April 1992.