Chapter 1

Martin Gogolla: Identifying Objects by Declarative Queries

Abstract

Object identification in data models, especially in semantic, Entity-Relationship, and Object-Oriented data models is studied. Various known approaches to object identification are shown, and an alternative proposal to the topic is put forward. The main new idea is to attach to each object type an arbitrary query, a so-called observation term, in order to observe a unique, identifying property of objects of the corresponding type.

1.1 Introduction

Conceptual modeling of information systems means to fix the static and dynamic characteristics of the system to be developed before going into implementation details. One of the main tasks in conceptual modeling is to identify the type of objects to be handled with the system under development: The objects [ABD$^+$90] are of central interest. Our approach concentrates on the question how to refer to single objects and identify them in a global system context.

One very often used approach for object identification is to refer to objects by a single, unique number like a document number or a personal identification number. The drawback of this approach is however that such a number is frequently not a natural property of the object is question. Such an identification number is something which comes from outside the object and which is only used to distinguish one object from another object, often in an unnatural way. Therefore one may argue that such artificial identification mechanisms do not

have to do with conceptual modeling because they already bring efficiency and implementation issues into consideration.
In contrast, our proposal here is to refer to objects by some of their natural, inherent properties. These properties are explicitly declared as contributing to the object identification (or reference) mechanism. The main new idea of our proposal is to attach to each object type an arbitrary query, a so-called observation term, in order to observe a unique, identifying property of objects of the corresponding type. The advantages of our so-called observation term approach are that

- it is possible to retrieve abstract objects by concrete data values,
- the approach allows not only a single identification mechanism but multiple observation terms are allowed,
- observations terms can be used in a variety of data models like Object-Oriented or Entity-Relationship data models,
- observation terms can be introduced dynamically during the lifetime of an information system as they are needed,
- it is possible to make even updates on the attributes contributing to the observation terms, and
- the approach is completely formally defined.

Our paper is organized as follows. In Sect. 1.2 we explain the basic idea of our approach and introduce an example schema which will be used throughout the paper. In Sect. 1.3 the interpretation of object schemas is sketched in an informal way. Several known and our new proposal (the so-called observation term approach) to object identification are studied in Sect. 1.4. The advantages of our new proposal are discussed in Sect. 1.5, and its formal definition is presented in Sect. 1.6. Our approach is put into the context of papers from the literature in Sect. 1.7. Section 1.8 concludes the paper.

1.2 The Basic Idea

As the starting point we consider in Fig. 1.1 a simple fraction of a car rentals case study [Wil94]. We take into account car rental company branches providing services like rental bookings and car deliveries. Branches are characterized in a simplified way by their street address and their location in a town. Towns, in which also the customers live, lie in countries. Towns and countries are recorded with their names and population. Branches own a set of cars characterized by a chasis number and an equipment category. Customers, which are described by their names, ask for bookings of cars in a certain category and for a certain time period in form of a start and end day. Such bookings are recorded together with the current date as the order date and a running booking number. After the start day bookings are associated with the car which has been delivered to the customer. A graphical description for a corresponding object schema following the UML notation [BJR97] but neglecting the data types of attributes is given in Fig. 1.1.

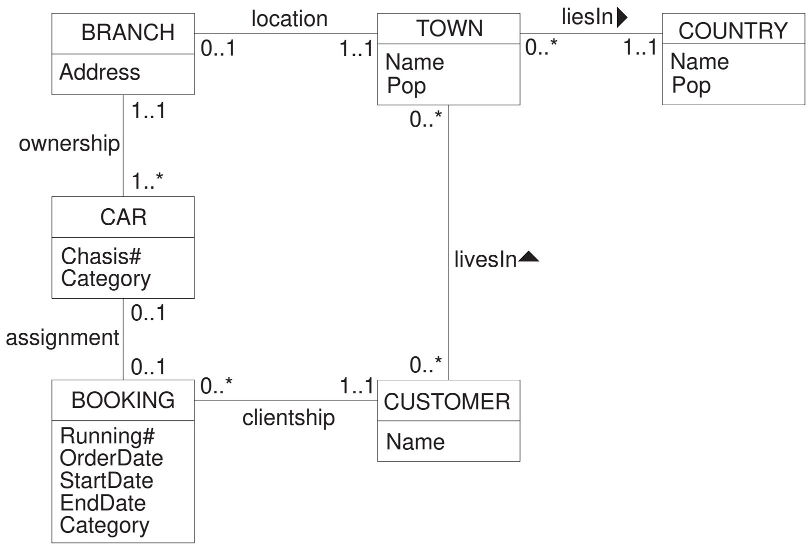

Figure 1.1: Graphical UML description of the car rentals case study schema

We shortly mention the UML class diagram features employed in our example: Object types (classes) are pictured as rectangles containing the attributes. Relationships (in UML terminology associations) are shown as lines together with the relationship name and an optional indication for the read-direction of the name: The black triangle indicates that, for example, towns lie in countries. Cardinalities are shown as integer intervals, in which a star $*$ may indicate that no upper bound is specified. The intervals express the minimal and maximal number of associated objects on the other side of the relationship: For example, each booking is related via the association 'clientship' with exactly one customer, but a customer may have zero or more bookings.
In Fig. 1.2 we make the diagram more precise by stating which data types the attributes have. Relationships are expressed as object-valued attributes, which are either single- or set-valued depending on the upper bound of the cardinality specification: An upper bound 1 indicates a single-valued, an upper bound $*$ a multi-valued attribute (interval 0..1 for optional, 1..1 for mandatory, 0..$*$ for empty set allowed, 1..$*$ for empty set not allowed). Inverse object-valued attributes are indicated in comments. The textual formulation of the diagram is given in an O_2-like style [Deu91] employing type constructors like `tuple` and `set`. We use the UML convention that role names for opposite object types in relationships are given by the name of the object type written with only

```
define type BRANCH:
  tuple( Address:   string,
         car:       set(CAR),      -- inverse of CAR.branch
         town:      TOWN );        -- inverse of TOWN.branch
define type CAR:
  tuple( Chasis#:   string,
         Category:  string,
         branch:    BRANCH,        -- inverse of BRANCH.car
         booking:   BOOKING );     -- inverse of BOOKING.car
define type BOOKING:
  tuple( Running#:  nat,
         OrderDate: string.
         StartDate: string,
         EndDate:   string,
         Category:  string,
         car:       CAR,           -- inverse of CAR.booking
         customer:  CUSTOMER );    -- inverse of CUSTOMER.booking
define type CUSTOMER:
  tuple( Name:      string,
         booking:   set(BOOKING),  -- inverse of BOOKING.customer
         town:      set(TOWN) );   -- inverse of TOWN.customer
define type TOWN:
  tuple( Name:      string,
         Pop:       int,
         branch:    BRANCH,        -- inverse of BRANCH.town
         customer:  set(CUSTOMER), -- inverse of CUSTOMER.town
         country:   COUNTRY );     -- inverse of COUNTRY.town
define type COUNTRY:
  tuple( Name:      string,
         Pop:       int,
         town:      set(TOWN) );   -- inverse of TOWN.country
```

Figure 1.2: Textual O_2-like description of the car rentals case study schema

lower case letters. Thus, for example, the relationship 'assignment' between 'BOOKING' and 'CAR' induces (through the role names) the object-valued attributes 'booking: CAR $\rightarrow$ BOOKING' and 'car: BOOKING $\rightarrow$ CAR'.
After having clarified the syntactic situation we can proceed to explain the basic idea of our proposal. As said before, the plan is to attach to each object type a query term which observes the identity of the corresponding objects. We show such query terms for some of the object types of the schema.

Object type 'CAR': We have a simple situation for cars because they possess a natural identification number. By natural we mean that such an identification number already is present in the problem domain and must not be introduced in an artificial way by the information system to be developed. Cars are identified by their chasis number which is a simple data-valued attribute. Thus the following integrity constraint has to be respected in all information system states.

$$\forall(c_1, c_2 : CAR)(Chasis\#(c_1) = Chasis\#(c_2)) \Rightarrow (c_1 = c_2)$$

We can also formulate this constraint equivalently by taking $(c_1 \neq c_2)$ as the premise.

$$\forall(c_1, c_2 : CAR)(c_1 \neq c_2) \Rightarrow (Chasis\#(c_1) \neq Chasis\#(c_2))$$

This formulation clearly points out that different cars can be observed by their different chasis numbers. Thus the string-valued chasis number constitutes the observation term τ for cars.

$$\tau(c : CAR) \equiv Chasis\#(c)$$

Object type 'BRANCH': The situation is more involved for branches because we cannot avoid that two branches have the same street address, for example, both are located on `"700 Main Street"`. Our solution is to take into account that branches lie in towns which again lie in countries.

$$\tau(b : BRANCH) \equiv tuple(Name(town(b)), Name(country(town(b))))$$

This observation term induces a constraint like the above one for cars which will be respected in all information system states. Again, this observation term is data-valued. The term has the sort $tuple(string, string)$.

Object type 'CUSTOMER': As the last example at this point we take into account customers. They may be identified, for demonstration purposes, by their own names together with the set of names of towns they live in.

$$\tau(c : CUSTOMER) \equiv tuple(Name(c), (\begin{array}{ll} select & Name(t) \\ from & (t : TOWN) \\ where & t \ in \ town(c))) \end{array}$$

As this last example shows, observation terms can have a complex structure because they may even include queries. They are however in any case data-valued. The sort of the last term is $tuple(string, bag(string))$.

The ideas underlying these examples will be explained and justified in more detail in the sections to follow.

1.3 Interpretation of Database Schemas

Before going into the details how exactly single elements of the given object domains are identified, we have to clarify how we formally interpret schemas like the above ones. Looking at the textual version of the schema in Fig. 1.2 we recognize that the following items must have an interpretation: Data sorts and data operations, object sorts, data-valued attributes, type constructors, and object-valued attributes. We explain the interpretation of these notions by examples from the above schema.

Data sorts: Each data sort like `string` is associated with a corresponding set of values denoted by $I(\texttt{string})$.[1]

Data operations: Each data operation like the string concatenation operation `concat : string string -> string` is associated with a function with respective source and target like $I(concat) : I(string) \times I(string) \to I(string)$.

Object sorts: Each object sort is connected with an infinite universe of object identities[2] for this object sort, like $U(\texttt{TOWN}) = \{\texttt{twn}_n \mid n \in \mathsf{N}\}$ for the object sort `TOWN`. In each database state there is an "active" finite subset of object identities like $I(\texttt{TOWN}) = \{\texttt{twn}_2, \texttt{twn}_4, \texttt{twn}_8\}$ representing the objects in the current state.

Data-valued attributes: In each database state, a data-valued attribute is seen as a function from the appropriate active object domain into the associated set of values like $I(\texttt{Name}) : I(\texttt{TOWN}) \to I(\texttt{string})$ for the attribute `Name : TOWN -> string`.

Type constructors: Each type constructor takes its argument and builds a new collection of items corresponding to its task while assuming that the active domain is fixed. For example, for `set(TOWN)` with $I(\texttt{TOWN})$ as above we have $I(\texttt{set})(I(\texttt{TOWN})) = \{\ \emptyset, \{\texttt{twn}_2\}, \{\texttt{twn}_4\}, ..., \{\texttt{twn}_2, \texttt{twn}_4\}, ..., \{\texttt{twn}_2, \texttt{twn}_4, \texttt{twn}_8\}\ \}$. Thus `set(TOWN)` is consisting of all sets which have elements of the active domain for `TOWN`.

Object-valued attributes: In each database state an object-valued attribute is connected with a function from the respective active object domain into the appropriate target. For example, for the attribute `country : TOWN -> COUNTRY` we have $I(\texttt{country}) : I(\texttt{TOWN}) \to I(\texttt{COUNTRY})$, and for the attribute `town : COUNTRY -> set(TOWN)` we have $I(\texttt{town}) : I(\texttt{COUNTRY}) \to I(\texttt{set})(I(\texttt{TOWN}))$. Thus, in a concrete state we could have for instance $I(\texttt{country}) : \texttt{twn}_2 \mapsto \{\texttt{cty}_{42}, \texttt{twn}_8\}$ and $I(\texttt{town}) : \texttt{cty}_{42} \mapsto \{\texttt{twn}_2, \texttt{twn}_8\}$.

A more general description for the presented items will be given later in Sect. 1.6.

1.4 Approaches to Object Identification

After we saw how to interpret schemas like the above one, in particular how to interpret object sorts by object identities, we can now proceed and discuss the topic of object identification. In principle, all occurring questions originate from

[1] Here, we give formal semantics to syntactic items by applying the function I which returns for the parameter the corresponding interpretation. The convention throughout the paper is that I denotes the interpretation of a syntactic item.

[2] Object identities have appeared under the name surrogates in [Cod79], l-values in [KV84], and object identifiers in [AH87].

the freedom and flexibility offered by the above interpretation for object sorts and, in consequence of this, from the freedom for the interpretation of attributes. The following situation can occur: There may be objects which have different identities, say country objects $\texttt{cty}_2$ and $\texttt{cty}_4$, but for which the interpretation of all attributes $I(\texttt{Name})$ and $I(\texttt{Pop})$ coincides. Thus we cannot distinguish between the objects $\texttt{cty}_2$ and $\texttt{cty}_4$ with respect to their characteristics, but assume we would like to do so, because we argue that formally different objects must somehow be formally different to the outside. The internal object identities are of no help in this case because both identities $\texttt{cty}_2$ and $\texttt{cty}_4$ do not tell us anything about the properties of the objects. Therefore, there must be some other means to distinguish them. In the following we discuss several approaches to achieve this distinction, namely

- data-valued key attributes,
- data- and object-valued key attributes,
- data- and object-valued key attributes with additional contributions from other attributes having the respective object sort as target sort,
- identity observing formulas, and
- identity observing terms and queries.

Data-valued keys: In the relational data model it is usual to mark some of the attributes of a relation as keys. This means that tuples in the relation are already uniquely identified by the values of the key attributes. This is also possible in our object model. For example, we could require that the attribute 'Name' is the key for the object type 'COUNTRY'. By doing this, we state a restriction on the interpretation of 'Name' and 'Pop' saying that the value of 'Name' already determines the rest of the object. However, we have to be careful with the formalization of this requirement. At least two different possibilities exist:

(1) $\forall(c_1, c_2 : COUNTRY)$
$\quad (Name(c_1) = Name(c_2)) \Rightarrow (Pop(c_1) = Pop(c_2))$
(2) $\forall(c_1, c_2 : COUNTRY)$
$\quad (Name(c_1) = Name(c_2)) \Rightarrow (c_1 = c_2)$

The first formula (1) requires that whenever two COUNTRY objects coincide in their 'Name' attribute, then they must also have equal values for the 'Pop' attribute. The second formula (2) states that under the same precondition as in (1) the two *objects* must be identical not only their 'Pop' *data values*. Data values are printable and meaningful to users while pure objects (not their attributes) are not meaningful to users and therefore not printable.[3] The difference between formulas (1) and (2) can be explained best by the sample state mentioned above where two different

[3]The distinction between values and objects is discussed, for instance, in [Bee90].

COUNTRY objects exist which have identical values for their 'Name' and 'Pop' attributes: This state is accepted as valid by formula (1) but not by formula (2). Following the arguments from above stating that different objects with indistinguishable properties have to be avoided we cannot accept (1) as a proper solution but must prefer the stronger[4] formula (2) as the right formulation for the problem. Let us explain how the above formulas contribute to object identification. We first re-formulate formula (2) equivalently by employing the simple logical law $(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\beta \Rightarrow \neg\alpha)$.

$$(2') \; \forall (c_1, c_2 : COUNTRY)(c_1 \neq c_2) \Rightarrow (Name(c_1) \neq Name(c_2))$$

This formulation now clearly states that two different countries must have different names.

Data- and object-valued keys: What we have done for data-valued attributes above can also be done for object-valued attributes. For example, we can require that 'Name' and 'country' constitute the key for TOWN allowing that different towns in different countries can have the same name. The restricting formula then reads as follows:

$$(3) \; \forall (t_1, t_2 : TOWN)$$
$$(Name(t_1) = Name(t_2) \wedge country(t_1) = country(t_2)) \Rightarrow (t_1 = t_2)$$
$$(3') \; \forall (t_1, t_2 : TOWN)$$
$$(t_1 \neq t_2) \Rightarrow (Name(t_1) \neq Name(t_2) \vee country(t_1) \neq country(t_2))$$

Data- and object-valued keys plus other attributes: The above achievement can even be extended by allowing not only attributes of the considered object type to contribute to object identity but also attributes of other object types which have the considered object sort as their target sort. Consider the slight modification of Fig. 1.1 presented in Fig. 1.3.

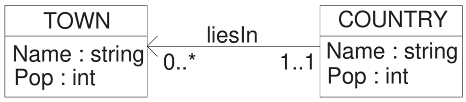

Figure 1.3: Part of modified car rental schema

As indicated by the arrow pointing to 'TOWN' we have replaced the bi-directional association 'liesIn' between 'TOWN' and 'COUNTRY' by a uni-directional association traversable only from 'COUNTRY' to 'TOWN'. Uni-directional associations are another language feature in UML and are different from the read-direction indication. This change means that we do not have the function 'country : TOWN $\rightarrow$ COUNTRY' available anymore. Now consider the same requirement for towns as above, namely that different towns in different countries with the same name are allowed

[4]'Stronger' in the sense that (2) implies (1).

to exist. Expressing this requires to refer to an attribute not belonging to the object type 'TOWN'. The respective formula reads as follows:[5]

$$\begin{array}{l}(4)\ \forall(t_1, t_2 : TOWN)\\ \quad (Name(t_1) = Name(t_2) \land\\ \quad \forall(c : COUNTRY)(t_1\ in\ town(c) \Leftrightarrow t_2\ in\ town(c)))\\ \quad \Rightarrow (t_1 = t_2)\end{array}$$

Again, it will be instructive to re-formulate the formula equivalently by employing the logical law $(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\beta \Rightarrow \neg\alpha)$. Then the requirement, in which $\oplus$ denotes the exclusive or, looks as follows:

$$\begin{array}{l}(4')\ \forall(t_1, t_2 : TOWN)\\ \quad (t_1 \neq t_2) \Rightarrow\\ \quad (Name(t_1) \neq Name(t_2) \lor\\ \quad \exists(c : COUNTRY)(t_1\ in\ town(c) \oplus t_2\ in\ town(c)))\end{array}$$

The formula states that, if two towns are different, then their names are different or there is a country such that one town lies in the country and the other not. Thus inequality of towns can be observed by 'Name' or 'town' (or both).

Identity observing formulas: All above axioms had a very particular form. Speaking in general terms, they were instances of the following axiom schema where the item *os* refers to an object sort and φ is a formula with two free variables o_1 and o_2.

$$(5)\ \forall(o_1, o_2 : os)\ \varphi(o_1, o_2) \Rightarrow (o_1 = o_2)$$

For instance in the case of formula (3) we had $\varphi(t_1, t_2) \equiv (Name(t_1) = Name(t_2) \land country(t_1) = country(t_2))$ which is a precondition analogously to the precondition of key constraints in the relational model. But if we take a broader view and allow not only these special key constraints, the formula $\varphi(o_1, o_2)$ can be any formula (with free variables o_1 and o_2) which then serves as an object identification mechanism. One such alternative choice is for instance $\varphi(t_1, t_2) \equiv (Name(t_1) = Name(t_2) \lor country(t_1) = country(t_2))$ which requires that two different towns must have different names *and* must lie in different countries.

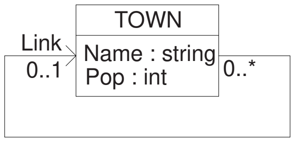

Figure 1.4: Schema for towns linked to nearest neighbours

[5]Requirements (2) and (3) look like requirements for well-known functional dependencies from relational database theory, however, this requirement (4) and the ones to follow cannot be expressed in this way.

Although such general preconditions offer a great amount of flexibility and look attractive, there are some difficulties which can arise when we employ such formulas for observing the identity of objects. Consider the schema in Fig. 1.4 in which each town (apart from having a 'Name') has a direct 'Link' to its nearest neighbour town. Here, the association in the UML schema does not possess a name but only a role name is specified. Assume that different towns must have different names or must be linked to different towns, which could be tried to be formulated as follows:

(6) $\forall (t_1, t_2 : TOWN)$
$$(t_1 \neq t_2) \Rightarrow (Name(t_1) \neq Name(t_2) \vee Link(t_1) \neq Link(t_2))$$

We re-formulate the formal requirement equivalently and obtain the equation $t_1 = t_2$ as the conclusion.

(6′) $\forall (t_1, t_2 : TOWN)$
$$(Name(t_1) = Name(t_2) \wedge Link(t_1) = Link(t_2)) \Rightarrow (t_1 = t_2)$$

Although this seems to be some kind of observable inequality there is a problem with this formula. Consider the state in Fig. 1.5 with three different TOWN objects twn_1, twn_2 and twn_3 which have identical 'Name' attribute values and are linked as depicted.

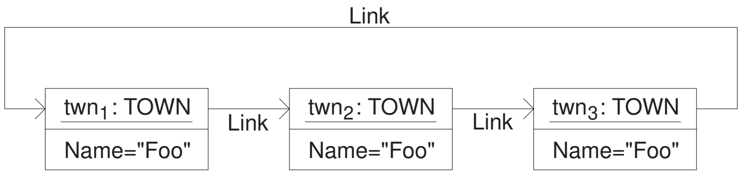

Figure 1.5: Database state with cyclic links

The formulas (6) and (6′) accept the state in Fig. 1.5 as a valid state but due to the cyclic 'Link' structure there is no possibility to distinguish between, say, the TOWN objects twn_1 and twn_2, by taking data observations only. Therefore the towns in the state cannot be observed to be different although the formula is valid. Thus we need something stronger and more informative than just truth values of formulas: We take data-valued queries.

Identity observing terms and queries: The above deficiency will be removed when one employs not only formulas for identification purposes but data terms which can be looked at and which can really generate object identities. Therefore the general idea for identity observing terms and queries is to have axiom schemas of the following form where τ is a term with a sort built over data sorts and type constructors and with one free variable of object sort os.

(7) $\forall (o_1, o_2 : os)(\tau(o_1) = \tau(o_2)) \Rightarrow (o_1 = o_2)$

Let us examine the above examples again in the light of this new perspective. We specify the observation terms τ which have to be substituted in the above axiom schema in order to yield the desired constraints.

- The formula (2) can be achieved if we actualize the above axiom schema with the term $\tau(c : COUNTRY) \equiv Name(c)$ which has the free variable $(c : COUNTRY)$ and is of sort *string*. Of course, we obtain a formula equivalent to formula (2′) from above.
- Requirement (3) from above is gained by substituting the term $\tau(t : TOWN) \equiv tuple(Name(t), Name(country(t)))$ with free variable $(t : TOWN)$. The term has the sort $tuple(string, string)$. *tuple* is assumed to be a type constructor which takes a collection of single values and returns the tuple value which consists of the given values.[6]
- Constraint (4) may be seen as an instantiation of the above schema with the following observation term:

  $$(8)\ \tau(t : TOWN) \equiv tuple(Name(t), (select\ Name(c)\ \ from\ \ (c : COUNTRY)\ \ where\ t\ in\ town(c)))$$

  This query term has exactly one free variable $(t : TOWN)$ and is of sort $tuple(string, bag(string))$. It employs a query term which is evaluated for the free variable $(t : TOWN)$ as a kind of parameter. The query follows the well-known select-from-where structure proposed in SQL. In our approach, the result of a query is a bag of values retaining duplicates; sets can also be obtained by applying an operator eliminating the duplicates (for details, see [Gog94]). The 'where' part is a formula which returns true if the parameter town t is an element of the set $town(c)$. The 'select' terms are only evaluated and returned for those countries for which the 'where' part yields true. Although we expect the query to return exactly one string value, we must formulate it as a multi-valued term because in the syntactic situation described in Fig. 1.3 the association is traversable in only one direction and the term 'Name(country(t))' is syntactically not allowed.
- The most complex situation was given in formula (6) where a cyclic 'Link' structure was specified in the schema, and also a cyclic state structure was given in the counter example state in Fig. 1.5.

  $$(9)\ \tau(t : TOWN) \equiv tuple(Name(t), Name(Link(t)), Name(Link(Link(t))))$$

  This term is again a tuple term and has one free variable $(t : TOWN)$. It is of sort $tuple(string, string, string)$ and selects the name of the 'parameter' town and the names of the first and second towns linked to the parameter town. Thus the above state (with three towns

[6] If want we to very precise we have to distinguish syntactically between the type constructor *tuple* (for example to construct the sort $tuple(string, int)$) and an operation mk_{tuple} constructing an appropriate value (for example $mk_{tuple} : string \times int \to tuple(string, int)$). However, we use the notation *tuple* in both positions.

$$\tau_{os} \text{ injective}$$

$$\begin{array}{rcccl} \tau_{os}: & os & \rightarrow & s_\tau & \\ & os & \leftarrow & s_\tau & : f_{\tau_{os}} \end{array}$$

$$f_{\tau_{os}} \text{ bijective on } \tau_{os}(os)$$

Figure 1.6: Overview on observation term τ_{os} and induced function $f_{\tau_{os}}$

named 'Foo') is not accepted by the constraint generated by the axiom schema if we actualize it by this observation term.

Let now us focus on the most important point of identity observing terms and queries. We assume that each object sort os in the database schema is attached to such an observation term τ_{os} of sort s_τ and that this attachment induces the constraint given hereafter. The sort s_τ may be a simple data sort or a complex sort expression, but a sort expression built only over data sorts with no object sorts in it.

(10) $(\forall o_1, o_2 : os)(o_1 \neq o_2) \Rightarrow (\tau_{os}(o_1) \neq \tau_{os}(o_2))$

We assume further that this constraint is enforced during the lifetime of the database. This means that two different os objects with the same τ_{os} value of sort s_τ cannot exist. This value can be computed for an os object by applying the term τ_{os} to the object and the other way round: An os object can be computed from an s_τ value. Or, to say it in other words, we have described the concept of a partial function $f_{\tau_{os}} : s_\tau \rightarrow os$ which will be *bijective* in all database states (on the part where it yields defined values). The function can therefore serve for object identification purposes. The function will be given an s_τ value as a parameter. It will respond by either returning a uniquely determined object with that s_τ value, or it will say that there is no object in the current database state with the respective property. Because of the possibility of this negative answer, the function will be partial. An overview of the situation is given in Fig. 1.6. In the total, we achieve with observation terms an approach which seems to be more general than and thus can be refined to the well-known "deep equality".[7]

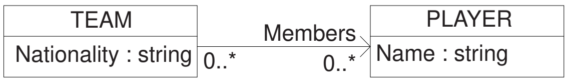

Figure 1.7: Teams and players

[7] "Deep equality" again refines to "shallow equality" and the "identity predicate" as pointed out in [KC86].

Let us finally give an example for an observation term where queries are really necessary. As pictured in Fig. 1.7 we consider teams of sportswomen and sportsmen who play, for instance, the Daviscup. Because there may be many teams for one nationality, one has to take into account the names of the team members for identifying single teams.

$$(11)\ \tau(t:TEAM) \equiv tuple(Nationality(t), (\begin{array}{ll} select & Name(p) \\ from & (p:PLAYER) \\ where & p\ in\ Members(t))) \end{array}$$

The observation term above has one free variable $(t:TEAM)$ and has the sort $tuple(string, bag(string))$. In general, the 'select' term will return a proper multi-valued result because teams usually consist of more than one player.

1.5 Advantages of the Observation Term Approach

Our observation term approach to object identification embodies a number of features that distinguishes it from other ways to handle the problem and offers a greater amount of flexibility. We explain how to retrieve objects, how to treat multiple observation terms, how to apply the approach to other data models, how to dynamically use observation terms, how to handle updates on the attributes involved in observation terms, and how to define the approach formally.

Retrieving abstract objects by concrete values: Within the observation term approach it is possible to retrieve abstract objects, which are internally identified by surrogates, with the help of concrete data values. After retrieving an object in this way one can access other properties which are different from the identifying ones.

Multiple observation terms: It is also possible to have more than one observation term for a single object sort. Consider the case that we have for COUNTRY objects in addition to a 'Name' observation (i.e., 'Name' is a key for COUNTRY) also an attribute for the country's geometry. In order to keep things simple we assume this geometry is given by the country's border represented by a polygon: 'Geo : COUNTRY $\rightarrow$ polygon'. Now clearly two different countries will have different geometries, i.e., different polygons, and therefore the attribute 'Geo' can serve as a second observation structure for a COUNTRY object $(c:COUNTRY)$: $\tau(c:COUNTRY) \equiv Geo(c)$. Such multiple observation terms and their implementation by some sort of index are helpful from the conceptual point of view, because different identification mechanisms for one object sort are supported. Such multiple observation terms are also useful from the implementation point of view in terms of query optimization, because these 'query' indexes can be employed for faster query evaluation and constraint enforcement.

Observation terms for other data models: Clearly, the applicability of the observation term approach is not restricted to the object-oriented world. The underlying idea originated from keys in the relational model, and therefore, this approach is applicable to the relational model per se. Another candidate for application are Entity-Relationship schemas.

Dynamic observation terms: Yet another distinguishing feature of the observation term approach is that it can be employed dynamically during the lifetime of a database. It is not necessary to specify all observation terms at the time when the database schema is fixed. Initially, only one such term may be given. Afterwards, other such observations may be added and dropped as it is needed. This is analogously to the relational model where indexes on different attributes for a relation may be created and deleted dynamically.

Updates on observation contributing attributes: The observation term approach generalizes key attributes of the relational model, and there has been a debate in the database field whether to allow for updates on key attributes or not. Formally, we do not have key attributes but we can coin the notion of an *observation contributing attribute* for an attribute which is needed for calculating the τ_{os} value for an object o of sort os. And then we can ask the same question as in the relational model, namely, whether to allow updates on these attributes or not. Our answer to this is very simple: Yes, we allow for such updates as long as the integrity of the database is respected. Take for example the observation term (4) from above and the situation depicted in Fig. 1.8.

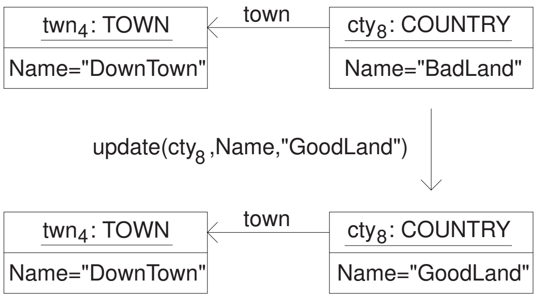

Figure 1.8: Update on an observation contributing attribute

The attribute 'Name' of 'COUNTRY' is an observation contributing attribute for 'TOWN' in the above sense of the word, and it is updated in the example in Fig. 1.8. This update presents no difficulty as long as the follow-up state obeys the specified constraints. In the new state we use

$tuple("DownTown", \{"GoodLand"\})$ to identify the object twn_4 instead of $tuple("DownTown", \{"BadLand"\})$ in the old state. If one does not want the effect that an object is identified by data term t_1 in one state and data term t_2 in the other one, then one can exclude this behavior by additional transitional (or even temporal) constraints. For the general case, we prefer to have the freedom to change observation contributing attributes, because additional requirements in this respect are really a restriction which can be formulated explicitly.

Formal definition: As will be pointed out in the following Sect. 1.6, our approach is defined on a formal mathematical basis. Thus, comprehension and implementation is supported without any ambiguities.

1.6 Formal Definition

This section gives (in a very condensed way) the formal definitions neccessary for the well-definedness of our approach. A more elaborate treatment is given for example in [Gog94,HG94]. Readers with no or little mathematical background are invited to inspect those papers first. We start with some preliminary notations.

Notation: Let sets $S, S_1, \ldots, S_n$ be given. Then $\boldsymbol{\mathcal{F}(S)}$ denotes the restriction of the powerset of S to finite sets, $\boldsymbol{\mathcal{B}(S)}$ the set of finite multisets (bags) over S, and $\boldsymbol{S_1 \times \ldots \times S_n}$ the Cartesian product of the sets $S_1, \ldots, S_n$. Finite sets are written as $\boldsymbol{\{c_1, \ldots, c_n\}}$, bags as $\boldsymbol{\{\!\{ c_1, \ldots, c_n \}\!\}}$, lists as $< \boldsymbol{c_1, \ldots, c_n} >$, and elements of the Cartesian product as $\boldsymbol{(c_1, \ldots, c_n)}$.

The basis of our framework assumes a data signature and a corresponding interpretation to be given. Such an interpretation is mainly an algebra in the sense of abstract data type theory.

Definition 1: Data signature

A data signature is given by a triple $\Sigma = (S, \Omega, \Pi)$ in which S denotes a set of sorts, $\Omega = \{\Omega_s\}_{s \in S^* \times S}$ an $S^* \times S$-indexed family of operation symbols and $\Pi = \{\Pi_s\}_{s \in S^*}$ an S^*-indexed family of predicate symbols.

$\omega_{s_1 \ldots s_n, s} \in \Omega$ is also written as $\omega : s_1 \times \ldots \times s_n \to s$ and $\pi_{s_1 \ldots s_n} \in \Pi$ as $\pi : s_1 \times \ldots \times s_n$.

For a given data signature Σ an interpretation structure is defined as a triple $I(\Sigma) = (I(S), I(\Omega), I(\Pi))$.

- $I(S)$ associates each sort $s \in S$ with a set $I(s)$ such that $\perp_s \in I(s)$.
- $I(\Omega)$ associates each operation symbol $\omega : s_1 \times \ldots \times s_n \to s \in \Omega$ with a total function $I(\omega) : I(s_1) \times \ldots \times I(s_n) \to I(s)$.
- $I(\Pi)$ associates each predicate symbol $\pi : s_1 \times \ldots \times s_n \in \Pi$ with a relation $I(\pi) \subseteq I(s_1) \times \ldots \times I(s_n)$.

The interpretation of some standard data sorts may be fixed as follows.

$I(\text{nat}) = \mathsf{N}_0 \cup \{\perp_{\text{nat}}\}$, $I(\text{int}) = \mathsf{Z} \cup \{\perp_{\text{int}}\}$, $I(\text{real}) = \mathsf{Q} \cup \{\perp_{\text{real}}\}$,
$I(\text{bool}) = \{\mathit{true},\ \mathit{false}\} \cup \{\perp_{\text{bool}}\}$,
$I(\text{char}) = A \cup \{\perp_{\text{char}}\}$, $I(\text{string}) = A^* \cup \{\perp_{\text{string}}\}$,

Here, A denotes a finite set of characters. Standard operations and predicates have the usual meaning. Formally, we have a bottom value $\perp_s$ representing undefined for every sort s. When no confusion can occur we drop the additional index.

Examples:

i. string $\in S$

ii. concat : string string $\rightarrow$ string $\in \Omega$

iii. $=$: string string $\in \Pi$

Definition 2: Sort expressions
Let a set S with $S = S_D \cup S_O$, S_D a set of data sorts according to Def. 1 and S_O a set of object sorts be given. Then the set $\textit{S-Expr}(S)$ of sort expressions (or type constructor applications) over S is defined as follows.

i. If $s \in S$, then $\boldsymbol{s} \in \textit{S-Expr}(S)$.

ii. If $s \in \textit{S-Expr}(S)$, then $\mathbf{set}(\boldsymbol{s})$, $\mathbf{bag}(\boldsymbol{s})$, $\mathbf{list}(\boldsymbol{s}) \in \textit{S-Expr}(S)$.

iii. If $s_1, \ldots, s_n \in \textit{S-Expr}(S)$, then $\mathbf{tuple}(\boldsymbol{s_1, \ldots, s_n}) \in \textit{S-Expr}(S)$.

Let us assume that there is a fixed interpretation I of S_D according to Def. 1. Then the interpretation of sort expressions is defined as follows.

i. $I(ds)$ is given as in Def. 1 for data sorts $ds \in S_D$. For each object sort $os \in S_O$ we have an infinite universe of object identifiers $U(os) = \{os_n \mid n \in \mathsf{N}\} \cup \{\perp\}$. In each database state we have an "active" finite subset $I(os) \subseteq_{finite} U(os)$ of the universe representing the objects in the current state with $\perp \in I(os)$.

ii. $I(\text{set}(s)) := \mathcal{F}(I(s)) \cup \{\perp\}$. $I(\text{bag}(s)) := \mathcal{B}(I(s)) \cup \{\perp\}$.
$I(\text{list}(s)) := I(s)^* \cup \{\perp\}$.

iii. $I(\text{tuple}(s_1, \ldots, s_n)) := (I(s_1) \times \ldots \times I(s_n)) \cup \{\perp\}$.

Sort expressions are usually associated with a large number of *generic* (or overloaded) functions. There are operations and predicates that

- *compose* structured values from simpler ones. For instance, { ... } is used as a constructor for set values and *tuple*(...) is used as a constructor for tuple values.
- *decompose* structured values into simpler ones. For instance, the operation $PRJ_i : \mathit{tuple}(s_1, \ldots, s_n) \rightarrow s_i$ selects the ith component of a tuple.
- *convert* structured values. For instance, $\texttt{BTS}_{\text{bag}(s)}$: bag$(s) \rightarrow$ set(s) converts a bag into a set by duplicate elimination.

- *exploit* the distinct properties of certain kinds of structured values. For instance, $\mathtt{CNT}_{\mathrm{set}(s)} : \mathrm{set}(s) \to \mathrm{nat}$ counts the elements in a finite set (also defined for bags and lists), $\mathtt{OCC}_{\mathrm{bag}(s)} : \mathrm{bag}(s) \times s \to \mathrm{nat}$ counts the occurrences of a given item in a bag, $\mathtt{IN}_{\mathrm{set}(s)} : \mathrm{set(s)} \times s$ denotes the membership predicate (also defined for bags and lists).

There are many other generic operations (for details see [GH91,Gog94]). All operations induced by sort expressions and the operations induced by the database schema are summarized in $\Omega_E(\textit{S-Expr}(S))$. Analogously all predicates are combined in $\Pi_E(\textit{S-Expr}(S))$.

Examples:

i. TOWN $\in S_O$

ii. bag(tuple(CUSTOMER,tuple(int,set(string)))) $\in \textit{S-Expr}(S_D \cup S_O)$

iii. Name : CUSTOMER $\to$ string $\in \Omega$

iv. town : COUNTRY $\to$ set(TOWN) $\in \Omega$

From now on let $\Sigma_D = (S_D, \Omega_D, \Pi_D)$ denote a data signature according to Def. 1 so that $\Sigma = (S, \Omega, \Pi)$ denotes an extended signature given by $S = \textit{S-Expr}(S_D \cup S_O)$, $\Omega = \Omega_D \cup \Omega_E(\textit{S-Expr}(S_D \cup S_O))$, and $\Pi = \Pi_D \cup \Pi_E(\textit{S-Expr}(S_D \cup S_O))$. Furthermore, all following syntactic items (variables, terms, formulas, declarations, queries, constraints) are interpreted in a given database state. Such a database state allows only a finite number of current objects (according to the definition of sort expression).

Definition 3: Variables and variable assignments

Let an S-indexed family $\mathit{Var} = \{\mathit{Var}_s\}_{s \in S}$ of sets of variables be given. The set of variable assignments A is defined by $A := \{\alpha \mid \alpha : \mathit{Var}_s \to I(s)$ for all $s \in S\}$. The special assignment $\epsilon : \mathit{Var} \to \{\perp\}$ is called the *empty assignment*. The index of a set of variables denotes the sort of the respective variables.

Examples:

i. s $\in V_{string}$

ii. t $\in V_{TOWN}$

Definition 4: Terms

The syntax of terms is given by an S-indexed family $\mathit{Term} = \{\mathit{Term}_s\}_{s \in S}$ and a function free : $\mathit{Term} \to \mathcal{F}(\mathit{Var})$ defined by the following rules.

i. If $v \in \mathit{Var}_s$, then $\boldsymbol{v} \in \mathit{Term}_s$ with $\mathrm{free}(v) := \{v\}$.

ii. If $\omega : s_1 \times \ldots \times s_n \to s \in \Omega$ and $\tau_i \in \mathit{Term}_{s_i}$ $(i = 1 \ldots n)$, then $\boldsymbol{\omega(\tau_1, \ldots, \tau_n)} \in \mathit{Term}_s$ with $\mathrm{free}(\omega(\tau_1, \ldots, \tau_n)) := \mathrm{free}(\tau_1) \cup \ldots \cup \mathrm{free}(\tau_n)$.

iii. If $\tau \in \mathit{Term}_s$, $\delta \in \mathit{Decl}$ and $\varphi \in \mathit{Form}$, then $\boldsymbol{\{\!\!\{\, \tau \mid \delta,\ \varphi \,\}\!\!\}} \in \mathit{Term}_{\mathrm{bag}(s)}$ with $\mathrm{free}(\{\!\!\{\, \tau \mid \delta,\ \varphi \,\}\!\!\}) := (\mathrm{free}(\tau) \cup \mathrm{free}(\delta) \cup \mathrm{free}(\varphi)) \setminus \mathrm{decl}(\delta)$.

For a fixed interpretation I and a variable assignment $\alpha \in A$ the evaluation of terms is defined as follows.

i. $(I,\alpha)[\![\, v \,]\!] = \alpha(v)$.

ii. $(I,\alpha)[\![\, \omega(\tau_1,\ldots,\tau_n) \,]\!] = I(\omega)((I,\alpha)[\![\, \tau_1 \,]\!],\ldots,(I,\alpha)[\![\, \tau_n \,]\!])$.

iii. $(I,\alpha)[\![\, \{\!\mid \tau \mid \delta,\ \varphi \mid\!\} \,]\!] = \{\!\{\, (I,\alpha')[\![\, \tau \,]\!] \mid\ \alpha' \in A$ with $\alpha'(v) = \alpha(v)$ for all $v \in \mathit{Var} \setminus \mathrm{decl}(\delta)$ and $(I,\alpha') \models \delta$ and $(I,\alpha') \models \varphi \,\}\!\}$.

Examples:

i. c $\in \mathit{TERM}_{\mathit{COUNTRY}}$

ii. town(c) $\in \mathit{TERM}_{\mathit{set(TOWN)}}$

iii. $\{\!\mid$ Name(c), Name(t) | (c:COUNTRY; (t:TOWN)), t in town(c) $\mid\!\} \in$ $\mathit{TERM}_{\mathit{bag(tuple(string,string))}}$

Definition 5: Formulas

The syntax of formulas is defined by a set *Form* and a function free : $\mathit{Form} \rightarrow \mathcal{F}(\mathit{Var})$ defined by the following rules.

i. If $\pi : s_1 \times \ldots \times s_n \in \Pi$ and $\tau_i \in \mathit{Term}_{s_i}$ $(i = 1\ldots n)$, then $\boldsymbol{\pi(\tau_1,\ldots,\tau_n)}$ $\in \mathit{Form}$ with $\mathrm{free}(\pi(\tau_1,\ldots,\tau_n)) := \mathrm{free}(\tau_1) \cup \ldots \cup \mathrm{free}(\tau_n)$.

ii. If $\tau_1, \tau_2 \in \mathit{Term}_s$, then $\boldsymbol{\tau_1 = \tau_2} \in \mathit{Form}$ with $\mathrm{free}(\tau_1 = \tau_2) := \mathrm{free}(\tau_1) \cup \mathrm{free}(\tau_2)$.

iii. If $\varphi \in \mathit{Form}$, then $\boldsymbol{\neg(\varphi)} \in \mathit{Form}$ with $\mathrm{free}(\neg(\varphi)) := \mathrm{free}(\varphi)$.

iv. If $\varphi_1, \varphi_2 \in \mathit{Form}$, then $\boldsymbol{(\varphi_1 \vee \varphi_2)} \in \mathit{Form}$ with $\mathrm{free}((\varphi_1 \vee \varphi_2)) := \mathrm{free}(\varphi_1) \cup \mathrm{free}(\varphi_2)$.

v. If $\delta \in \mathit{Decl}$ and $\varphi \in \mathit{Form}$, then $\boldsymbol{\exists\delta(\varphi)} \in \mathit{Form}$ with $\mathrm{free}(\exists\delta(\varphi)) := (\mathrm{free}(\delta) \cup \mathrm{free}(\varphi)) \setminus \mathrm{decl}(\delta)$.

For a fixed interpretation I and a variable assignment $\alpha \in A$ the validity of formulas is defined as follows.

i. $(I,\alpha) \models \pi(\tau_1,\ldots,\tau_n)$ iff $((I,\alpha)[\![\, \tau_1 \,]\!],\ldots,(I,\alpha)[\![\, \tau_n \,]\!]) \in I(\pi)$.

ii. $(I,\alpha) \models \tau_1 = \tau_2$ iff $(I,\alpha)[\![\, \tau_1 \,]\!] = (I,\alpha)[\![\, \tau_2 \,]\!]$.

iii. $(I,\alpha) \models \neg(\varphi)$ iff not $(I,\alpha) \models \varphi$.

iv. $(I,\alpha) \models (\varphi_1 \vee \varphi_2)$ iff $(I,\alpha) \models \varphi_1$ or $(I,\alpha) \models \varphi_2$.

v. $(I,\alpha) \models (\exists\delta(\varphi))$ iff there is a variable assignment $\alpha' \in A$ with $\alpha'(v) = \alpha(v)$ for all $v \in \mathit{Var} \setminus \mathrm{decl}(\delta)$ and $(I,\alpha') \models \varphi$ and $(I,\alpha') \models \delta$.

The other logical connectives $\wedge$, $\Rightarrow$, $\Leftrightarrow$, and the quantifier $\forall$ can be defined by the given material.

Examples:

i. Name(c)="Greece" $\vee$ t in town(c) $\in \mathit{Form}$

ii. clientship(b,c) $\Rightarrow$ StartDate(b) $\leq$ EndDate(b) $\in$ *Form*

Definition 6: Declarations
The syntax of declarations is given by a set *Decl* and the functions free, decl : $Decl \rightarrow \mathcal{F}(Var)$ defined by the following rules.

i. If $v \in Var_s$, $\tau \in Term_{\mathrm{set}(s)}$ and $v \notin \mathrm{free}(\tau)$, then $\boldsymbol{(v : \tau)} \in Decl$, $\mathrm{free}((v : \tau)) := \mathrm{free}(\tau)$ and $\mathrm{decl}((v : \tau)) := \{v\}$.

ii. If $v \in Var_s$, $\tau \in Term_{\mathrm{set}(s)}$, $\delta \in Decl$ and $v \notin \mathrm{free}(\tau) \cup \mathrm{free}(\delta) \cup \mathrm{decl}(\delta)$, then $\boldsymbol{(v : \tau;\ \delta)} \in Decl$, $\mathrm{free}((v : \tau;\ \delta)) := (\mathrm{free}(\tau) \setminus \mathrm{decl}(\delta)) \cup \mathrm{free}(\delta)$ and $\mathrm{decl}((v : \tau;\ \delta)) := \{v\} \cup \mathrm{decl}(\delta)$.

For a fixed interpretation I and a variable assignment $\alpha \in A$ the validity of declarations is defined as follows.

i. $(I, \alpha) \models (v : \tau)$ iff $\alpha(v) \in (I, \alpha)[\![\, \tau \,]\!]$.

ii. $(I, \alpha) \models (v : \tau;\ \delta)$ iff $\alpha(v) \in (I, \alpha)[\![\, \tau \,]\!]$ and $(I, \alpha) \models \delta$.

For a declaration δ the function free returns the free variables of δ and the function decl gives the set of variables declared within δ. For a declaration sequence $(x_1{:}\tau_1;\ (x_2{:}\tau_2;\ \ldots;\ (x_n{:}\tau_n)))$ this is $\{x_1, \ldots, x_n\}$. In such a declaration sequence a variable x_i must not be declared twice and is not allowed to be free to the right, i.e., to be free in τ_j with $j \geq i$.

Examples:

i. (c:COUNTRY; (t:TOWN)) $\in$ *Decl*

ii. (c:COUNTRY; (t:town(c))) $\in$ *Decl*

The concrete syntax of terms, formulas, and declarations may vary from the concise notation above. For example, instead of $\{\!|\, \tau \mid \delta, \varphi \,|\!\}$ we have used the more familiar (*select* τ *from* δ *where* φ). Furthermore, because object sorts are interpreted by finite sets (of object identities), we allow them as set-valued terms in declarations. For example, $(t : TOWN)$ is a valid declaration.

Definition 7: Queries and constraints

1. Every term $\tau \in Term$ with $\mathrm{free}(\tau) = \emptyset$ defines a query. The evaluation of queries is determined by $(I, \epsilon)[\![\, \tau \,]\!]$.

2. Every formula $\varphi \in Form$ with $\mathrm{free}(\varphi) = \emptyset$ defines a constraint. The validity of constraints is determined by $(I, \epsilon) \models \varphi$.

Examples:

i. Query: (select Name(c), (select Name(t) from (t:town(c))) from (c:COUNTRY))

ii. Constraint: $\forall$ (b:BOOKING) StartDate(b) $\leq$ EndDate(b)

1.7 Comparision with Other Approaches

Our approach is unique in comparison to related work because the general concept of observation term has not been introduced before. Observation terms are not even found in the literature under a different label. The degree of formality of our approach also seems to distinguish it from others. Nevertheless object identity has been studied intensively in recent literature. The approaches can be categorized into programming and specification language-oriented and database-oriented ones.

Database-oriented approaches: In [AK89] the significance of object identity w.r.t. data base query languages is pointed out. The proposed query language IQL employs object identities to represent data structures with sharing and cycles, to manipulate sets, and to express any computable database query. [Osb88] discusses object identity with special emphasis on query optimization, and [PG88] extends the relational key concept. Object identities from the deductive databases point of view are treated in [HY90]. Logic databases are also considered in [KLS92] with special emphasis on updates, a topic which seems to be one of the mayor problems in this field [Abi88]. For the special purpose of databases, a functional language incorporating parametrized set types is introduced. Different kinds of (not necessarily formal) object equality definitions are studied in [Mas89]. The implementation of the proposed concepts in the database management system OMEGA is also discussed there. In [VD91], algebraic query processing was the main motivation for given a set-theoretic semantics for object identities in the presence of multiple-inheritance. The paper also discusses implementation issues in the context of the EXTRA/EXCESS system. A similar framework like ours but with emphasis on integrating different databases and with attention to the instance level has been put forward in [EPLR93]. [SS95] studies object identity in connection with federated databases.

Programming or specification language-oriented approaches: A proposal to object identity from the functional programming perspective by the use of categorical monad structures is given in [Oho90]. [KDSW92] treats object identity from the data type point of view by providing a special identifier data type for this purpose. The approach in [LFBB94] is motivated from the declarative programming language implementation point of view. Our approach differs from the cited ones because we establish a connection between abstract object identities and concrete data values in order to identify objects. Our proposal to object identification can be implemented our the object specification language TROLL *light* [GCH93,RG97] which has been used recently in connection with the Unified Modeling Language [GPP98,GR98a,GR98b].

1.8 Conclusion

We have studied object identification in the context of Entity-Relationship and Object-Oriented data models. The new idea of our proposal, the so-called observation term approach, was to associate with each object type a query in order to observe an identifying property of objects. This proposal seems to be more general than the well-known “deep equality” approach [KC86]. We have argued that the advantages of our approach are the possibility of multiple observation terms, dynamic observation terms, updates on observation contributing attributes, and its formal definition.

We have not taken into account implementation issues, for example index and clustering techniques. Such techniques are advantageous in terms of query optimization, because ‘query’ indexes induced by observation terms can be employed for faster query evaluation and constraint enforcement.

References

[ABD$^+$90] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The Object-Oriented Database System Manifesto. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 223–240. Elsevier (North-Holland), 1990.

[Abi88] S. Abiteboul. Updates, a New Frontier. In M. Gyssens, J. Paredaens, and D. Van Gucht, editors, *Proc. 2nd Int. Conf. on Data Theory (ICDT)*, pages 1–18. Springer, Berlin, LNCS 326, 1988.

[ADD$^+$92] A. Auddino, Y. Dennebouy, Y. Dupont, E. Fontana, S. Spaccapietra, and Z. Tari. SUPER-Visual Interaction with an Object-Based ER Model. In G. Pernul and A.M. Tjoa, editors, *Proc. 11th Int. Conf. on Entity-Relationship Approach (ER'92)*, pages 340–356. Springer, Berlin, LNCS 645, 1992.

[AH87] S. Abiteboul and R. Hull. IFO — A Formal Semantic Database Model. *ACM Trans. on Database Systems*, 12(4):525–565, 1987.

[AK89] S. Abiteboul and P. Kanellakis. Object Identity as a Query Language Primitive. *ACM SIGMOD Record*, 18(2):159–173, 1989. Proc. ACM SIGMOD Conf. on Management of Data.

[Bee90] C. Beeri. A Formal Approach to Object-Oriented Databases. *Data & Knowledge Engineering*, 5(4):353–382, 1990.

[BJR97] G. Booch, I. Jacobson, and J. Rumbaugh, editors. *UML Summary (Version 1.1)*. Rational Corporation, Santa Clara, 1997. `http://www.rational.com`.

[Che76] P.P. Chen. The Entity-Relationship Model – Towards a Unified View of Data. *ACM Trans. on Database Systems*, 1(1):9–36, 1976.

[Cod79] E.F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Trans. on Database Systems*, 4(4):397–434, 1979.

[Deu91] O. Deux et. al. The O2 System. *Communications of the ACM*, 34(10):34–48, 1991.

[Dit87] K.R. Dittrich. Object-Oriented Database Systems: A Workshop Report. In S. Spaccapietra, editor, *Proc. 5th Int. Conf. on Entity-Relationship Approach (ER'86)*, pages 51–66. North Holland, Amsterdam, 1987.

[EGH$^+$92] 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, North-Holland*, 9(2):157–204, 1992.

[**EPLR93**] S. Prabhakar E.-P. Lim, J. Srivastava and J. Richardson. Entity Identification in Database Integration. In IEEE, editor, *Proc. 9th Int. Conf. Data Engineering (ICDE'93)*, pages 294–301, 1993.

[**GCH93**] M. Gogolla, S. Conrad, and R. Herzig. Sketching Concepts and Computational Model of TROLL light. In A. Miola, editor, *Proc. 3rd Int. Symposium Design and Implementation of Symbolic Computation Systems (DISCO'93)*, pages 17–32. Springer, Berlin, LNCS 722, 1993.

[**GH91**] M. Gogolla and U. Hohenstein. Towards a Semantic View of an Extended Entity-Relationship Model. *ACM Trans. on Database Systems*, 16(3):369–416, 1991.

[**GHC+93**] M. Gogolla, R. Herzig, S. Conrad, G. Denker, and N. Vlachantonis. Integrating the ER Approach in an OO Environment. In R. Elmasi and V. Kouramajian, editors, *Proc. 12th Int. Conf. on Entity-Relationship Approach (ER'93)*, pages 373–384. ER Institute, Pittsburgh (CA), Participants' Proceedings, 1993.

[**Gog94**] M. Gogolla. *An Extended Entity-Relationship Model – Fundamentals and Pragmatics.* Springer, Berlin, LNCS 767, 1994 (Also `http://www.db.informatik. uni-bremen.de`).

[**GPP98**] M. Gogolla and F. Parisi-Presicce. State Diagrams in UML - A Formal Semantics using Graph Transformation. In M. Broy, D. Coleman, T. Maibaum, and B. Rumpe, editors, *Proc. ICSE'98 Workshop on Precise Semantics of Modeling Techniques (PSMT'98)*, pages 55–72. Technical University of Munich, Technical Report TUM-I9803, 1998.

[**GR97**] M. Gogolla and M. Richters. On Constraints and Queries in UML. In M. Schader and A. Korthaus, editors, *Proc. UML'97 Workshop 'The Unified Modeling Language - Technical Aspects and Applications'*, pages 109–121. Physica-Verlag, Heidelberg, 1997.

[**GR98a**] M. Gogolla and M. Richters. Equivalence Rules for UML Class Diagrams. In P.-A. Muller and J. Bezivin, editors, *Proc. UML'98 Workshop 'Beyond the Notation'*, pages 86–97. Universite de Haute-Alsace, Mulhouse, 1998.

[**GR98b**] M. Gogolla and M. Richters. On Combing Semi-Formal and Formal Object Specification Techniques. In F. Parisi-Presicce, editor, *Proc. 12th Int. Workshop Abstract Data Types (WADT'97)*. Springer, LNCS, 1998.

[**HG94**] R. Herzig and M. Gogolla. A SQL-like Query Calculus for Object-Oriented Database Systems. In E. Bertino and S. Urban, editors, *Proc. Int. Symp. on Object-Oriented Methodologies and Systems (ISOOMS'94)*, pages 20–39. Springer, Berlin, LNCS 858, 1994 (Also `http://www.db.informatik.uni-bremen.de`).

[HY90] R. Hull and M. Yoshikawa. ILOG: Declarative Creation and Manipulation of Object Identifiers. In R. Sacks-Davis D. McLeod and H.-J. Schek, editors, *Proc. Int. Conf. Very Large Data Bases (VLDB'90)*, pages 455–468. Morgan Kaufmann, 1990.

[KC86] S.N. Khoshafian and G.P. Copeland. Object Identity. *ACM SIGPLAN Notices*, 21(11):406–416, 1986. Proc. OOPSLA.

[KDSW92] J.W. Schmidt K.-D. Schewe and I. Wetzel. Identification, Genericity and Consistency in Object-Oriented Databases. In J. Biskup and R. Hull, editors, *Proc. 4th Int. Conf. on Database Theory (ICDT'92)*, pages 341–356. Springer, Berlin, LNCS 646, 1992.

[KLS92] M. Kramer, G. Lausen, and G. Saake. Updates in a Rule-Based Language for Objects. In Li-Yan Yuan, editor, *Proc. 18th Int. Conf. on Very Large Databases, Vancouver*, pages 251—262, 1992.

[KS88] G. Kappel and M. Schrefl. A Behaviour Integrated Entity-Relationship Approach for the Design of Object-Oriented Databases. In C. Batini, editor, *Proc. 7th Int. Conf. on Entity-Relationship Approach (ER'88)*, pages 175–192. ER Institute, Pittsburgh (CA), Participants' Proceedings, 1988.

[KV84] G.M. Kuper and M.Y. Vardi. A New Approach to Database Logic. In *Proc. 3th ACM Symp. Principles of Database Systems (PODS)*, pages 86–96, 1984.

[LBSL90] K.J. Lieberherr, P. Bergstein, and I. Silva-Lepe. Abstraction of Object-Oriented Data Models. In H. Kangassalo, editor, *Proc. 9th Int. Conf. on Entity-Relationship Approach (ER'90)*, pages 81–94. ER Institute, Pittsburgh (CA), Participants' Proceedings, 1990.

[LFBB94] G. Lopez, B.N. Freeman-Benson, and A. Borning. Constraints and Object Identity. In M. Tokoro and R. Pareschi, editors, *Proc. 8th European Conf. Object-Oriented Programming (ECOOP'94)*, pages 260–279. Springer, Berlin, LNCS 821, 1994.

[LT94] T.W. Ling and P.K. Teo. A Normal Form Object-Oriented Entity Relationship Diagram. In P. Loucopoulos, editor, *Proc. Int. Conf. on Entity-Relationship Approach (ER'94)*, pages 241–258. Springer, Berlin, LNCS 881, 1994.

[Mas90] Y. Masunaga. Object Identity, Equality and Relational Concept. In J.-M. Nicolas W. Kim and S. Nishio, editors, *Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases (DOOD'89)*, pages 185–202. North-Holland, Amsterdam, 1990.

[NP88] S.B. Navathe and M.K. Pillalamarri. OOER: Toward Making the E-R Approach Object-Oriented. In C. Batini, editor, *Proc. 7th Int. Conf.*

on Entity-Relationship Approach (ER'88), pages 55–76. ER Institute, Pittsburgh (CA), Participants' Proceedings, 1988.

[Oho90] A. Ohori. Representing Object Identity in a Pure Functional Language. In S. Abiteboul and P.C. Kanellakis, editors, *Proc. 3rd Int. Conf. on Database Theory (ICDT'90)*, pages 41–55. Springer, Berlin, LNCS 470, 1990.

[Osb88] S.L. Osborn. Identity, Equality, and Query Optimization. In K. Dittrich, editor, *Proc. 2nd Int. Workshop on Object-Oriented Database Systems*, pages 346–351. Springer, Berlin, LNCS 334, 1988.

[PG88] N.W. Paton and P.W.D. Gray. Identification of Database Objects by Key. In K. Dittrich, editor, *Proc. 2nd Int. Workshop on Object-Oriented Database Systems*, pages 280–285. Springer, Berlin, LNCS 334, 1988.

[PKB94] M.P. Papazoglou, B.J. Kraemer, and A. Bouguettaya. On the Representation of Objects with Polymorphic Shape and Behaviour. In P. Loucopoulos, editor, *Proc. 13th Int. Conf. on Entity-Relationship Approach (ER'94)*, pages 223–240. Springer, Berlin, LNCS 881, 1994.

[RG97] M. Richters and M. Gogolla. A Web-based Animator for Validating Object Specifications. In B.C. Desai and B. Eaglestone, editors, *Proc. Int. Database Engineering and Applications Symposium (IDEAS'97)*, pages 211–219. IEEE, Los Alamitos, 1997.

[RS91] E. Rose and A. Segev. TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints. In T.J. Teorey, editor, *Proc. 10th Int. Conf. on Entity-Relationship Approach (ER'91)*, pages 205–230. ER Institute, Pittsburgh (CA), Participants' Proceedings, 1991.

[SS95] I. Schmitt and G. Saake. Managing Object Identity in Federated Database Systems. In M.P. Papazoglou, editor, *Proc. Int. Conf. Object-Oriented and Entity-Relationship Modelling (ER'95)*, pages 400–411. Springer, Berlin, LNCS 1021, 1995.

[VD91] S.L. Vandenberg and D.J. DeWitt. Algebraic Support for Complex Objects with Arrays, Identity, and Inheritance. In *Proc. ACM SIGMOD Conf. on Management of Data (SIGMOD'91), SIGMOD Record, 20(2)*, pages 158–167, , 1991.

[Wil94] B. Wilson. EU-Rent Car Rentals Case Study. Technical report, Model Systems & Brian Wilson Associates, 1994.

[WJ91] R. Wieringa and W. de Jonge. The Identification of Objects and Roles — Object Identifiers Revisited. Technical Report IR-267, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1991.