Proc. 14th Int. Conference on Object-Oriented and Entity Relationship Modelling (OO-ER'95), M. Papazoglou (Ed.), Springer, Berlin, LNCS (1995)

A Declarative Query Approach to Object Identification

Martin Gogolla*
Bremen University, FB 3, Informatics
Postfach 330440, D-28334 Bremen
Germany
e-mail: gogolla@informatik.uni-bremen.de

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 in order to observe a unique, identifying property of objects of the corresponding type.

1 Introduction

One of the central issues in object-orientation [ABD+90] is object identity. In object models with identity, objects can exist independent of the values of their attributes. Advantages of such identities are object sharing and object updates. The importance of the identity concept with different perspectives from programming and database languages [KC86] and philosophy [WJ91] has also been emphasized.

Object identity has been studied intensively in recent literature. 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. 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]. An approach to object identity from the functional programming perspective by the use of categorical monad structures is given in [Oho90]. 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 [Mas90]. 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. [SSW92] treats

* Work reported here has been partially supported by the CEC under Grant No. 6112 (COMPASS).

object identity from the data type point of view by providing a special identifier data type for this purpose. A similar framework like ours but with emphasis on integrating different databases and with attention to the instance level has been put forward in [LSPR93]. 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 paper is organized as follows. In Sect. 2 we introduce an example schema which will be used throughout the paper, and in Sect. 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. 4. The advantages of our new proposal are discussed in Sect. 5. The paper ends with some concluding remarks in Sect. 6.

2 An Example Schema: Town-Country-River

Let us start with a simple example application where real world objects like towns, countries, and rivers are of interest. Towns and countries both possess the data-valued properties 'Name' and 'Pop' (for population), rivers also have a 'Name' and a 'Length' of flow. Towns possess an object-valued property 'LiesIn' which gives the country object the town object lies in; towns also have a set- and object-valued property 'LiesAt' yielding the set of rivers the town lies at. Finally, rivers are assumed to 'FlowThrough' a set of country objects. The schema for this application can be represented graphically as given in Fig. 1 (following the notation of [GH91]).

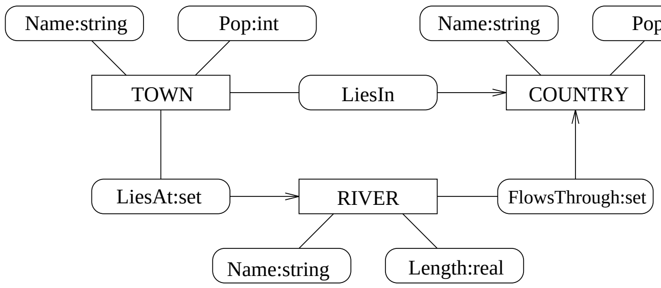

Fig. 1. Town-Country-River OO-like

In order to round off the discussion of the representation of the considered schema we explain how the above would have been modelled in the classical Entity-Relationship [Che76] approach. Here, the object-valued attributes would have been expressed as general relationships as denoted in Fig. 2. In order to simplify matters we refrain from adorning the arcs of the relationships with cardinality specifications typically found in many Entity-Relationship approaches.

It is amazing to see how similar these schemas are. In fact, in our comprehension it is only a matter of taste – as far as structural modelling is concerned – to use the term "Entity-Relationship" or "Object-Oriented" database schema. Indeed,

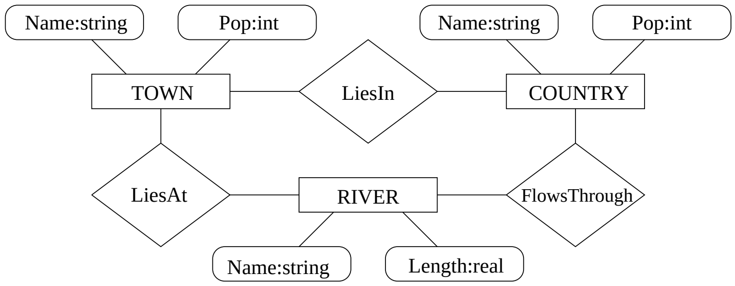

Fig. 2. Town-Country-River ER-like

the relation between Entity-Relationship and Object-Oriented modelling has always been in the center of interest on the Entity-Relationship conferences (see for example [Dit87, NP88, KS88, LBSL90, RS91, ADD+92, PKB94, LT94], among many other interesting papers).

3 Interpretation of Database Schemas

We now turn to the compelling question of how single elements of the given object domains are identified. But before going into details concerning this point, we have to clarify how we formally interpret schemas like the above ones. 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 $I(string)$.[1]

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

Object sorts: Each object sort is connected with a infinite universe of object identities[2] for this object sort like $U(RIVER) = \{river_n \mid n \in \mathbf{N}\}$ for the object sort `RIVER`. In each database state there is an "active" finite subset of object identities like $I(RIVER) = \{river_2, river_4, river_8\}$ 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 $I(Name) : I(TOWN) \rightarrow I(string)$ for the attribute `Name : TOWN -> string`.

1 Here, we give formal semantics to syntactic domains by applying the function I which returns for the parameter the corresponding interpretation.

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

Type constructors: Each type constructor builds over its argument a new collection of items corresponding to its task. For example, for `set(RIVER)` we have $I(set)(I(RIVER)) = \{\ \emptyset,\ \{river_2\},\ \{river_4\},\ ...,\ \{river_2, river_4\},\ ...,\ \{river_2, river_4, river_8\}\ \}$.

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 `LiesIn : TOWN -> COUNTRY` we have $I(LiesIn) : I(TOWN) \to I(COUNTRY)$, and for the attribute `LiesAt : TOWN -> set(RIVER)` we have $I(LiesAt) : I(TOWN) \to I(set)(I(RIVER))$. Thus, in a concrete state we could have for instance $I(LiesAt) : town_{42} \mapsto \{river_2, river_8\}$.

4 Approaches to Object Identification

After we have given an impression how to interpret schemas like the above ones, 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 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 $country_2$ and $country_4$, but for which the interpretation of both attributes $I(Name)$ and $I(Pop)$ coincides. Thus we cannot distinguish between the objects $country_2$ and $country_4$ wrt. their characteristics, but assume we would like to do so. The object identities are of no help in this case because both identities tell us nothing 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 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 would 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:

$(A)\ \forall(c_1, c_2 : COUNTRY)$
$$(Name(c_1) = Name(c_2)) \Rightarrow (Pop(c_1) = Pop(c_2))$$
$(B)\ \forall(c_1, c_2 : COUNTRY)$
$$(Name(c_1) = Name(c_2)) \Rightarrow (c_1 = c_2)$$

The first formula (A) 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 (B) states that under the same precondition as in (A) the two *objects* must be identical not only their 'Pop' *data values*.[3] The difference can be explained best by the sample state mentioned above where two different COUNTRY objects exist which have identical values for their 'Name' and 'Pop' attributes: This state would be accepted as valid by formula (A) but not by formula (B). Following the arguments from above stating that different objects with indistinguishable properties should be avoided we cannot accept (A) as a proper solution but must prefer the stronger[4] formula (B) as the right formulation for the problem.

Perhaps it is instructive to explain how the above formulas contribute to object identification. Let us first re-formulate formula (B) equivalently by employing the simple logical law $(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\beta \Rightarrow \neg\alpha)$.

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

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 could require that 'Name' and 'LiesIn' are the key for TOWN allowing that different towns in different countries can have the same name. The restricting formula then reads as follows:

$(C)\, \forall(t_1, t_2 : TOWN)$

$$(Name(t_1) = Name(t_2) \wedge LiesIn(t_1) = LiesIn(t_2)) \Rightarrow (t_1 = t_2)$$

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 presented in Fig. 3.

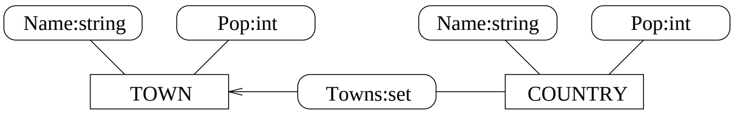

Fig. 3. Town-Country-River OO-like (modified)

We have replaced the single-valued attribute 'LiesIn : TOWN $\rightarrow$ COUNTRY' by the inverse, set-valued attribute 'Towns : COUNTRY $\rightarrow$ set(TOWN)'. Now consider the same requirement for towns as above, namely that different towns in different countries with the same name are allowed to exist. Expressing this requires to refer to an attribute not belonging to the object type 'TOWN'. The respective formula reads as follows:[5]

[3] The distinction between values and objects is discussed, for instance, in [Bee90].
[4] 'Stronger' in the sense that (B) implies (A).
[5] Requirements (B) and (C) look like requirements for well-known functional dependencies, however, this requirement (D) and the ones to follow cannot be expressed in this way.

$(D)\ \forall(t_1, t_2 : TOWN)$
$(Name(t_1) = Name(t_2) \wedge$
$(\forall(c : COUNTRY)(t_1\ in\ Towns(c) \Leftrightarrow t_2\ in\ Towns(c))))$
$\Rightarrow (t_1 = t_2)$

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

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

For instance in the case of formula (C) we had $\varphi(t_1, t_2) \equiv (Name(t_1) = Name(t_2) \wedge LiesIn(t_1) = LiesIn(t_2))$ 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 could be for instance $\varphi(t_1, t_2) \equiv (Name(t_1) = Name(t_2) \vee LiesIn(t_1) = LiesIn(t_2))$ which would require that two different towns must have different names and must lie in different countries.

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 following schema in Fig. 4 in which each town (apart from having a 'Name') has a direct 'Link' to its nearest neighbour town.

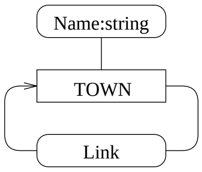

Fig. 4. Schema for towns linked to nearest neighbours

Furthermore assume that different towns must have different names or must be linked to different towns, which can be formulated as follows:

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

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

$(F')\ \forall(t_1, t_2 : TOWN)$
$(Name(t_1) = Name(t_2) \wedge Link(t_1) = Link(t_2)) \Rightarrow (t_1 = t_2)$

Although this seems to be some kind of observable inequality there is a problem with this formula. Consider the following state with three different TOWN objects $town_1$, $town_2$ and $town_3$ which have identical 'Name' attribute values and are linked as depicted in Fig. 5.

The formula (F) accepts the state in Fig. 5 as a valid state but due to the cyclic 'Link' structure there is no possibility to distinguish between, say, the town objects $town_1$ and $town_2$, by taking data observations only. Therefore all towns

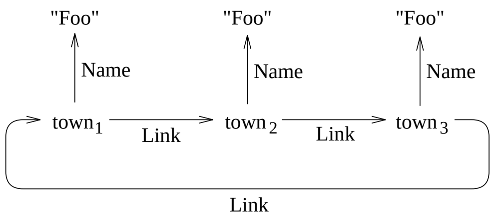

Fig. 5. Database state with cyclic links

in the state cannot be observed to be different although the formula is valid.

Identity observing terms and queries: The above obstacle 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.

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

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.

(B)**:** The formula (B) can be achieved if we actualize the above axiom schema with the term $\tau(c) \equiv Name(c)$ which has the free variable $(c : COUNTRY)$ and is of sort $string$. Of course, we obtain a formula equivalent to formula (B') from above.

(C)**:** Requirement (C) from above is gained by substituting the term $\tau(t) \equiv tuple(Name(t), Name(LiesIn(t)))$ 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]

(D)**:** Constraint (D) may be seen as an instantiation of the above schema with the following observation term:

$$(H)\, \tau(t) \equiv tuple(Name(t), (select\ Name(c)\ from\ (c : COUNTRY)\ where\ t\ in\ Towns(c)))$$

The 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, HG94]). The 'where' part is a formula which returns true if the

[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 $mk_{tuple} : string \times int \rightarrow tuple(string, int)$). However, we use the notation $tuple$ in both positions.

parameter town t is an element of the set $Towns(c)$. The 'select' terms are only evaluated and returned for those countries for which the 'where' part yields true.

(F): The most complex situation was given in formula (F) 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. 5.
$(I)\ \tau(t) \equiv tuple(Name(t), Name(Link(t)), Name(Link(Link(t))))$
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 named 'Foo') would not be 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 (data) sort s_τ and that this attachment induces the constraint given hereafter.
$(J)\ (\forall o_1, o_2 : os)(o_1 \neq o_2) \Rightarrow (\tau_{os}(o_1) \neq \tau_{os}(o_2))$
We assume further that this constraint is enforced during the whole 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) and 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.
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]

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 shortly explain how to define the approach formally, 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, and how to handle updates on the attributes involved in observation terms.

Formal definition: Our approach is defined on a formal mathematical basis. Thus, comprehension and implementation is supported without any ambiguities. Due

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

to space limitations we have to refer to [Gog94, HG94] for a more elaborate treatment.

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 ($c : COUNTRY$): $\tau(c) \equiv Geo(c)$. 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 applicatibility 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.

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 should 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.

6 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 its formal definition, the employment of multiple observation terms, dynamic observation terms, and updates on observation contributing attributes.

We have not taken into account implementation issues, for example index and clustering techniques which could be of use here. 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, *Proc. Int. Conf. on Deductive and Object-Oriented Databases (DOOD'89)*, pages 223–240. North-Holland, Amsterdam, 1990.

[Abi88] S. Abiteboul. Updates, a New Frontier. In M. Gyssens, J. Paredaens, and D. van Gucht, editors, *Int. Conf. on Data Base Theory (ICDT'88)*, 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 (SIGMOD'89)*.

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

[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.

[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.

[Gog94] M. Gogolla. *An Extended Entity-Relationship Model – Fundamentals and Pragmatics*. Springer, Berlin, LNCS 767, 1994.

[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.

[HY90] R. Hull and M. Yoshikawa. ILOG: Declarative Creation and Manipulation of Object Identifiers. In D. McLeod, R. Sacks-Davis, 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. ACM Int. Conf. on Object Oriented Programming Systems, Languages and Applications (OOPSLA'86).*

[KLS92] M. Kramer, G. Lausen, and G. Saake. Updates in a Rule-Based Language for Objects. In L.-Y. Yuan, editor, *Proc. 18th Int. Conf. on Very Large Databases (VLDB'92)*, pages 251–262. Morgan Kaufmann, 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. 3rd ACM Symp. Principles of Database Systems (PODS'84)*, pages 86–96. ACM, New York, 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.

[LSPR93] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity Identification in Database Integration. In *Proc. 9th Int. Conf. Data Engineering (ICDE'93)*, pages 294–301. IEEE, 1993.

[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 W. Kim, J.-M. Nicolas, 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.

[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.

[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.

[SSW92] K.-D. Schewe, J.W. Schmidt, 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.

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

[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.