

The identification of objects and roles

- Object identifiers revisited -

Roel Wieringa

Wiebren de Jonge

Faculty of Mathematics and Computer Science, Vrije Universiteit
De Boelelaan 1081a, 1081 HV, Amsterdam
Email: roelw@cs.vu.nl, wiebren@cs.vu.nl

Abstract

In this paper we investigate several concepts that are known in database research for some time but which are still surrounded by some confusion. We start with the concepts of object identifier, surrogate and key and list the differences between these concepts, which in practice are often ignored. Sharpening these differences allows us to analyze the distinction between objects and roles that recently surfaced in the literature. Distinguishing objects from roles helps to avoid migration of objects between classes and thus the problems associated with it. We show that this distinction requires the use of role identifiers that, just as object identifiers, should be globally unique and unchangeable. We next show that using role identifiers requires a distinction between two essentially different kinds of inheritance relationships, *is-a* inheritance and *played-by* inheritance. These are often both treated as if they were one and the same kind of *is-a* relationship. The result of the discussion is a set of clear modeling concepts, which are independent of a particular language or implementation.

1 Introduction

There is a problem related to object identification and class migration that is important for object-oriented modeling and needs to be resolved. To take a standard example, assume that *STUDENT* is a subclass of *PERSON*. Counting the number of students at a university over a period of time, we may get a different answer from when we count the number of persons who studied at that university in the same period of time, because a person is at some universities counted as two students if he or she stops his or her study and starts afresh a second time. To count objects, we must identify them as distinct from others or reidentify them as the same object in a different state, and this is apparently done in a way that depends on the class of the object. Movement of an object to a subclass then creates another way to count objects and hence another criterion for object identity. But if students are counted differently from persons, then *STUDENT* cannot be a subclass of *PERSON*. In this paper, we sort out the problems connected to object identification and class migration and propose a solution to them.

The importance of the concept of an object identifier for object-orientation was pointed out by Khoshafian and Copeland [13] and recently by Kent [12]. The concept of object identifier has its ancestry in that of a surrogate used by Hall et al. [10] and Codd [7], among others, which itself arose out of dissatisfaction with the concept of a key. Along with object classification and the encapsulation of state and behavior, object identity is listed as an essential feature of object-oriented databases (OODB's) in several attempts to define what OODB's are [1, 28].

The relationship between object classification and identification has not been studied in-depth in object-oriented database modeling. The concept of a role, on the other hand, which we use to represent class migration, was already defined in 1977 by Bachman and Daya [2] in the context of the network data modeling approach. Recently, the concept surfaced again in the object-oriented literature [17, 19, 22, 24, 25, 26], although different and sometimes

incompatible definitions and formalizations have been given. We sort out these different approaches in section 8.

In section 2, we give a precise definition of object identifiers and end up with a concept of oid that does *not* coincide with other uses of it. We also show that there are important differences with two related concepts, keys and surrogates. Section 3 then introduces the difference between object classes and role classes and shows why we need both to adequately represent what is usually called class migration. The relation between classification, class migration and identification is explored in section 4. In section 5, the distinction between objects and roles is translated into an existence constraint. We show in section 6 that object classes and role classes each have their own taxonomic hierarchy, in each of which the inheritance mechanism at the instance level is based on identity (*is-a* inheritance). These two distinct hierarchies are related by another inheritance mechanism from objects to roles that are played by them (*played-by* inheritance), which is reminiscent of delegation [16]. Finally, we argue in section 7 that oid's cannot, strictly speaking, be implemented in practice, although it is useful as an ideal and we give ways to approximate it. Section 8 compares our approach with other approaches and section 9 concludes the paper. In order to increase the applicability of the ideas in this paper to different languages, we concentrate on conceptual issues in this paper and ignore language issues. Language issues for specifying roles and identifiers are treated elsewhere [25, 26].

2 Object identifiers, keys, and surrogates

2.1 Reasons for introducing object identifiers

In order to implement an information system (IS) that represents real-world objects, we need a representation of real-world objects that can be stored in an IS. We call this representation a **proper name** of the real-world object. We want this proper name to satisfy the following two requirements.

- The proper name should allow us to pick out an object among indistinguishable objects (i.e. among objects that are in the same state).
- The proper name should allow us to recognize an object as the *same object*, after it has changed state.

The reason for the first requirement is that in any conceptual model of reality, we represent reality at a certain level of abstraction, and this may cause different objects to be represented as if they were in the same state. The only thing that can then distinguish them in the model is their proper name, which must therefore at least be unique.

The reason for the second requirement is that without it, historical information about identities would be lost. Consider what would happen if one would allow changes in proper names. First, *equality* of proper names would then not necessarily imply that the named objects are identical. For example, even if uniqueness of proper names at every single moment in time is respected, two different persons could have the same proper name at different points in time. Second, *difference* of proper names would not signify object difference. For example, even if uniqueness of proper names through time is respected, a single person could have two different proper names at different moments in time.

If we want to represent objects in an information system, then we need to simulate object identities. The only way we can simulate the potentially infinite set of identities is by using

- I The **oid uniqueness principle**: in any possible state of the world, each relevant object has one and only one oid, which differs from the oid of any other relevant object.

- II The **oid persistence principle**: each relevant object has the same oid across all relevant states of the world. That is, the oid of an object remains invariant under any change of state of the object.

Figure 1: The two parts of the identity principle for oid’s.

globally unique¹ proper names, which we henceforth call **object identifiers** (oid’s). When an object becomes relevant for the first time, a fresh oid is created and assigned to the object (this is usually called “object creation”). We now get that an oid is a proper name that satisfies the two principles shown in figure 1, which correspond to the two requirements on proper names listed above.

2.2 Oid’s and keys

The concept of an oid is so general that it does not refer to databases at all, even though oid’s are necessary to make a good database (DB) implementation possible. Thus, oid’s are a modeling concept and not a DB concept. Other concepts that seem to be close to that of an oid are keys and surrogates. In this section we look at differences between oid’s and keys.

A **key** of a database tuple is a set of one or more attributes in the tuple with values that are, and must be, a unique combination in a relevant set of tuples. For example, a *database key* must be unique in each possible state of a database, and a *relation key* must be unique in each relation instance. We call a key **user-assigned** if there is at least one DB user (not being the DB administrator) that may assign key values to tuples. We have been able to uncover the following differences between keys and oid’s (see also figure 2 given at the end of section 2.3).

First, the concept of a key is a DB concept whereas the concept of an oid is more general. Oid’s can be defined for real-world objects as well as for objects residing in a computer.

Second, because keys consist of attributes, key values are information-carrying. For example, a key consisting of name and birth date not only contains the information that the identified tuple differs from any other in the relevant set of tuples, it also contains the name and birth date of the represented entity. By contrast, oid’s contain no other information than difference from other objects and persistence of the same object over time.

Third, keys are updatable whereas oid’s are not. Note, however, that in practice updates of keys are often prohibited and that the requirement of non-updatability could easily be added to the definition of a key. (In particular to the definition of a primary key.)

Fourth, a key is required to be unique in each *single* state of the *database* (or relation), whereas an oid must be unique across *all* possible states of the *world*. We regard this as a fundamental difference that sets oid’s apart from keys.

A difference between keys and oid’s often mentioned is that oid’s avoid a serious problem with the *exchange of identity information* between different DB’s. This is the problem to find out, when two DB’s are combined, which items of information represent the same object

¹By **globally unique** we mean unique across all possible states of the world. By the requirement of unchangeable identity, our oid’s are really what are called *rigid designators* in modal logic [9].

	Keys	Oid's	Surrogates
1.	Database concept	Modeling concept	Implementation concept
2.	Information-carrying	No information beyond object identity	No information beyond object identity
3.	Updatable	Non-updatable	Non-updatable
4.	Unique in each single state of a DB	Unique across all possible states of the world	Unique across all possible states of one DB
5.	Merge problem frequent	No merge problem	DB merge problem
6.	Often assigned by DB user	Assigned by the oid distributor (see section 7)	Assigned by DB system
7.	Visible to the user	Visible to the user	Invisible to the user

Figure 2: Differences between keys, oid's, and surrogates.

and which represent different objects. The problem arises in heterogeneous DB's [11], in DB merges, and in EDI networks. For example, when two vehicle databases are combined that use different keys for vehicles, it is impossible to determine on the basis of keys alone whether two tuples represent the same vehicle. This problem is indeed solved by the concept of oid as defined in figure 1, because all possible vehicles have globally unique and persistent oid's. Any other concept of oid does *not* solve the problem, as discussed in section 7.

2.3 Oid's and surrogates

Besides the concepts of oid and key, the concept of a surrogate has been proposed by others [7, 10]. A **surrogate** is an identifier assigned to the representation of an object by the database *system* itself, when that representation of the object is entered in the database. If we assume that each object is represented by one tuple in the database, then, when a tuple is created, the DB system creates a fresh surrogate that stays invariant even when values in the tuple are changed. That surrogate can be viewed as the internal DB representative of the real-world object.

Surrogates are implementation concepts and are kept invisible for the user. They have no significance in the conceptual model of reality.

When data is exchanged between the DB and some other party (a user or some other DB), identity information can be exchanged only by using an identifier known and used as such by the communicating parties. Thus, there is a need for visible identity information. Surrogates cannot be used for this purpose, since they are invisible outside the DB system. Since oid's are designed to represent identity information, it is natural to let oid's be visible to the user as one of the possible communication partners of the DB. Figure 2 lists the differences between keys, oid's, and surrogates.

3 Object classification and counting

We define an **object class** as the largest set of possible objects that share a set of properties, such as having an attribute (e.g. having the *age* attribute) or having a possible behavior (e.g. having an *inc_age* event in one's life cycle). In what follows, by the **extension** of a class we mean the set of *all its possible* instances, even if most of these do not exist currently and

- An instance of an *object class* cannot exist without being an instance of that class.
- For an instance of a *role class* there are possible states of the world in which it exists, as an object, without being an instance of that role class.

Figure 3: Objects and roles.

even if there are possible instances that may never exist. We call the set of instances existing in a state the **existence set** of a class in that state.

If a class is the largest set of possible objects that share a set of properties, then it seems logical to say that one class is a *subclass* of another if it has more properties than the other. The class of passengers would therefore seem to be a subclass of the class of persons, for a passenger has all the properties of a person, and some more. However, it is not correct to see *PASSENGER* as a subclass of *PERSON*. Compare the number of passengers a bus carried in a week with the number of persons that bus carried in the same week. It is possible that the count of the number of passengers is 4000, but a count of the number of persons is 1000. Apparently, counting the number of instances of class *PASSENGER* gives a different result from counting the number of instances of the class *PERSON*. If *PASSENGER* would be a subclass of *PERSON*, then counting passengers should give the same result as counting persons.

The conclusion of this observation can be stated in terms of identifiers as follows. If *PASSENGER* would be a subclass of *PERSON*, then each passenger identifier would also be a person identifier. Since this is not the case, persons and passengers apparently have different identifiers. This can be understood if we realize that a passenger is not *identical* to a person, but that it is a *state* of a person. When we count passengers, we really count how often persons have been in the state of being a passenger. Contrast this with the relationship between *CAR* and *VEHICLE*; each car is (identical to) a vehicle, so *CAR* is a subclass of *VEHICLE*.

We will say that *PASSENGER* is a **role class** and *PERSON* an **object class**. Instances of role classes are **roles** and instances of object classes are **objects**. A role is (part of) a state in which an object can be and if the object is in that state, we say that it *plays* that role. The difference between a role class and an object class is characterized in figure 3 [25].

For example, a car cannot exist without being an instance of *CAR*, but a student can exist as a person without being a *STUDENT*. This is because a student is a person in a certain state, and that person can exist without being in that state. A car, on the other hand, is not another object in the state of being a car; it is just a car. So *CAR* is an object class, but *STUDENT* is a role class. Most examples of classes in data modeling are actually role classes. Employees, secretaries, drivers, passengers, bosses and property are all roles played by instances of certain classes of objects.

4 The representation of roles

In what follows, we will represent objects by their oid's and object states by tuples containing a mapping from attributes to values. So a person object in a particular state can, in an ad

hoc notation, be represented as

$$(p, (name : n, address : a)),$$

where p is a *PERSON* oid, n is a string, and a an address identifier. We assume that the set of attributes of persons, and the set of allowable values of each attribute, has been declared somewhere.

If a role is (part of) a state, and a state is represented by a set of labeled attribute values, then apparently a role is represented by a set of labeled attribute values. However, this is not quite true. For example,

$$(p, (name : n, address : a, salary : s, company : c))$$

is a person playing the role of employee for a company with oid c and with salary s . After a salary raise, the person is in state

$$(p, (name : n, address : a, salary : s', company : c)).$$

This is a different state of p but we regard it as the *same* role, because the person has not ceased to be that employee, even though the person is in a different state. Thus, just like object states, we want to be able to talk about role states. Only if p resigns and later is rehired, then we regard p as playing a *different* employee role.

A second requirement on the representation of roles is that we should allow objects to play two roles simultaneously. Take a person who is two employees at the same time:

$$(p, (name : n, address : a, salary : s, company : c, salary : s', company : c')).$$

The information which salary belongs to which company is not represented (because the order of the attribute/value pairs in the tuple is not significant). Structuring the tuple does help, but not enough. In

$$(p, (name : n, address : a, (salary : s, company : c), (salary : s', company : c')))$$

it is not clear how we should address the salary p earns with one of the companies.

These requirements lead us to the introduction of **role identifiers** (rid's) for the same reasons as we need object identifiers. First, for each role of an object, we want to be able to distinguish it from other, possibly indistinguishable roles played by the same object and second, for each role played by an object, we want to be able to recognize it as the *same role* even if the object changes the state of that role. So we get as our representation

$$(p, (name : n, address : a), (e, (salary : s, company : c)))$$

for a person p in the state of playing role e with salary s . In

$$(p, (name : n, address : a), (e, (salary : s', company : c))),$$

p is playing the same role with a different salary, and in

$$(p, (name : n, address : a), (e', (salary : s, company : c))),$$

he is playing a different role with the same company and salary.

The phenomenon that a person can become a student is often called “class migration.” The object then gets student properties in addition to its person properties. In some implementations, this involves changing the identifier of the person object into that of a student object [14]. But changing identifiers runs counter to the essence of the idea of an oid and creates practical implementation problems, such as how to avoid dangling references and how to represent historical information. In our approach, class migration is modeled by adding or deleting roles to an object and we need not change any object- or role identifier to do this. An object or role always remains an instance of its classes and when an object starts playing a role, a fresh rid is created for that role.

A second advantage of our representation is that it provides more flexibility in information-sharing. For example, we may have a state of the world with different persons p_1 and p_2 , playing the roles of employee e_1 and e_2 , respectively, that share information as follows:

$$(p_1, (name : \dots), (e_1, (\dots))), \\ (p_2, (name : \dots), (e_2, (manager : e_1, \dots))).$$

Introducing rid’s introduces the problem that a person playing the role of employee is not identical to that employee. This means that properties defined for that person are not directly applicable to that employee. We look at a way to represent this taxonomic relationship in section 6 below. There, we also look at the question whether roles can play other roles.

5 Objects, roles and existence

We have now reached a point where the difference between roles and objects almost seems to vanish. The relation between a role and the object that plays it can be represented in several ways. A person in a state where (s)he is two employees, could be represented as the three tuples

$$(p, (name : n, address : a)) \\ (e, (PERSON : p, salary : s, company : c)) \\ (e', (PERSON : p, salary : s', company : c')).$$

We used *PERSON* as the name of the attribute referring to the object playing the role of employee. (There are other ways to represent this situation, but this does not affect the following argument.) Whatever the representation, this seems indistinguishable from a model in which there are three objects, one person and two employees, which are related to each other in some way. Thus, it looks as if we reduced our model to one in which there are only objects, each with a unique identifier and with attribute values that may themselves be identifiers (or sets of identifiers).

However, the difference between roles and objects stated in figure 3 still holds. One essential difference between objects and roles is that a role cannot exist without being played by an object, but objects can exist without playing roles; another essential difference is that the taxonomic structure of object- and role classes imposes constraints on the way roles can be played (see section 6). Because of the 1-1 correspondence between (existing) objects and roles on the one hand and their identifiers on the other, we can state the first essential difference in terms of identifiers as follows.

1. An existing rid is existence-dependent on an existing oid, directly or indirectly via one or more other existing roles (see section 6).
2. An existing oid need not play any roles.

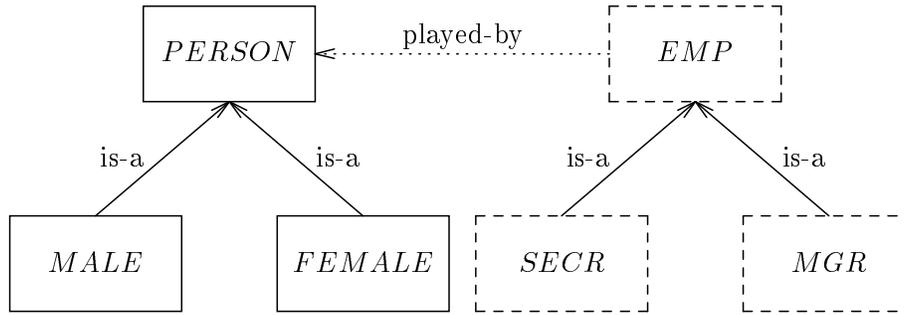


Figure 4: *is-a* inheritance and *played-by* inheritance.

6 Taxonomy

The distinction between objects and roles implies that we have two distinct taxonomic hierarchies, as illustrated in figure 4. The dashed boxes in the figure represent role classes and the other boxes object classes. There are two kinds of inheritance possible, which we call **is-a inheritance**, represented by an unbroken arrow from subclass to superclass, and **played-by inheritance**, represented by a dotted arrow from a role class to the class whose instances can play the role.

By *is-a* inheritance we mean simply that each object or role has precisely one identity, to which all properties defined for its class(es) are applicable. For example, viewing *SECR* as a subclass of *EMP*, a secretary role is an instance of the *SECR* and *EMP* role classes, and therefore just has all attributes, events, constraints etc. defined for instances of these two classes.

Played-by inheritance works as follows. If *name* is a *PERSON* attribute but not an *EMP* attribute, there is an unambiguous answer to the question what the name of an employee is, because for each role there is only one object playing that role. If an employee is asked for its name, it can delegate the answering of this question to the unique and existing object that plays it. The mechanism of *played-by* inheritance is a particular application of inheritance by delegation, introduced by Lieberman [16].

We write $C_1 \leq C_2$ if C_1 is a subclass of C_2 and $R_1 \leq R_2$ if R_1 is a subclass of R_2 . If all instances of C can play role R we write $C \rightsquigarrow R$. An important part of an OODB schema will be the specification of constraints describing which classes of objects can play which classes of roles.

Two interesting interactions between \leq and \rightsquigarrow are (figure 4):

1. If $C_2 \rightsquigarrow R$ and $C_1 \leq C_2$, then $C_1 \rightsquigarrow R$. For example, if all persons can play the role of employee, then all females can play the role of employee.
2. If $C \rightsquigarrow R_1$ and $R_1 \leq R_2$, then $C \rightsquigarrow R_2$. For example, if all persons can play the role of secretary, then all persons can play the role of employee.

Space limitations prevent us from working this out in more detail.

A final remark about taxonomy is that we see no reason why roles could not play roles. Possible examples of $R_1 \rightsquigarrow R_2$ are employees who become project manager, employees who become secretary, students who become teaching-assistant, etc. (cf. figure 5). In each of these cases, we have a choice between $R_1 \rightsquigarrow R_2$ and $R_2 \leq R_1$. In $R_1 \rightsquigarrow R_2$, we can have several R_2 instances as role of an R_1 instance, and we can represent persistence of an R_2 instance through change of state of that role instance independently from the identity of

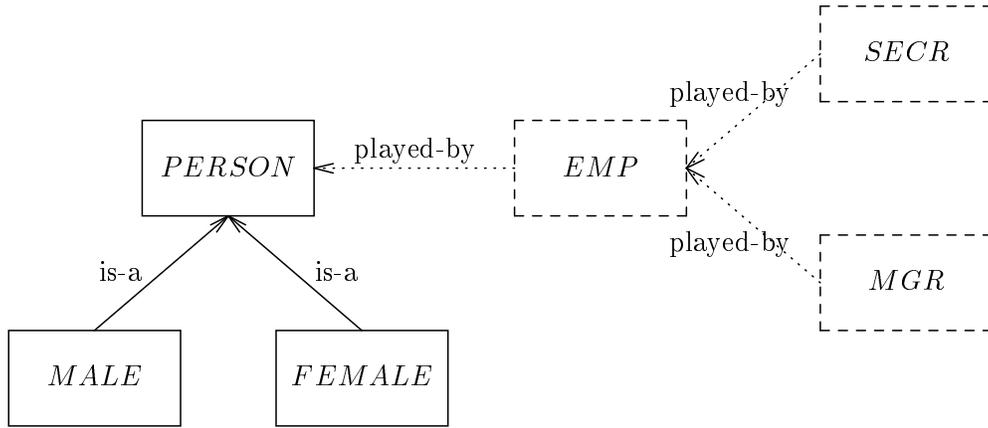


Figure 5: Subroles of roles.

the R_1 instance. So $R_1 \rightsquigarrow R_2$ makes it possible, for example, to let an object play role R_1 and let this R_1 instance play role R_2 later, or never. In the employee example, we think it is better to choose figure 5 as representation, because one employee can change functions without becoming a new employee.

The advantages of $R_1 \rightsquigarrow R_2$ are absent when we choose $R_2 \leq R_1$. In addition, the choice for $R_2 \leq R_1$ may involve partial instantiation. For example, in figure 4, we could have an *EMP* instance that is not a *SECR* or *MGR* instance since it did not migrate (yet) to one of its subclasses. Partial instantiation may be a bad idea, because it introduces null values for attributes that are not (yet) applicable, and this introduces a host of problems in formalization as well as implementation.

7 Problems with oid's

Pure oid's as defined in this paper are often not used, for a number of reasons that we briefly review here. First, objects like pins, nails, screws etc. produced by a factory usually remain nameless because no one cares to assign oid's to them.

Second, objects may have an oid in the sense defined in this paper, but we may not be in a position to know it. For reasons of privacy, people often prefer to remain anonymous [5, 6], and to respect this privacy, their oid's are not used. Bus passengers are usually not known by their globally unique person oid, even if they would have one.

Third, to realize the use of oid's, we need an **oid distributor**, by which we mean an entity or a group of entities that assigns oid's to objects. Database administrators often have this function, or clerks, or even a special organization. The oid distributor must give out a fresh oid to objects according to a procedure which guarantees that no oid is given out more than once. Even if an oid is never given out more than once, the oid distributor must determine whether an object given to it to be "baptized" has not already been assigned an oid before. Making a wrong decision here will lead to one object being assigned two proper names by the oid distributor as if they were the oid of the object. There is no fail-safe way to circumvent this problem, and in practice many mistakes are made. This means that the proper names given out cannot be guaranteed for 100% to be in fact oid's.

Finally, in current practice one does not even *attempt* to implement an oid distributor which assigns *one* globally unique oid recognized as such by all other organizations. Governments issue social security numbers, companies issue employee numbers, etc. and they treat

all these numbers as if they were oid's, but these numbers do not satisfy the requirements of figure 1. (Note that social security numbers and employee numbers can be considered to be rid's.) We prefer to call them **pseudo-oid's**, by which we mean a proper name, issued by an authority for a class of relevant objects, and that is globally unique and unchangeable for instances of that *class*. The issuing authority is called a **pseudo-oid distributor**. In the case of database systems, we assume a DB administrator is appointed who is responsible for the assignment of pseudo-oid's to objects that become relevant for the DB system.

Multiple pseudo-oid's per object create the **identity exchange problem**, mentioned already in section 2.2 in connection with heterogeneous DB's, EDI networks, DB merges, etc. The problem is simply that, given two pseudo-oid's, there is no way of determining whether they denote the same object, unless we have more information. This extra information can consist of the presence of the denoted object(s) and the information that there is no fraud: if there is one object and there is no fraud, then the two pseudo-oid's apparently denote the same object. If we merge two large DB's that use pseudo-oid's from different distributors, this is not a realistic option. Thus, contrary to what is often suggested, the (pseudo-)oid's used in practice do *not* solve the DB merge problem, although they help to reduce it. An organization can, for example, decide to use a pseudo-oid already given out by a high-level authority, such as a social security number. This is still far from the ideal scheme of a globally unique oid, however, because there are people without social security number and there are others with more than one. The identity exchange problem therefore remains in these situations.

We hasten to add that there are situations in which the identity exchange problem ought not to be solved at all. For reasons of privacy, persons often want to keep their identity secret and prefer to use one of their (many) rid's as their identification in a transaction [5, 20]. They then disclose the identifier of one of their roles, but do not reveal their identity as a person.

8 Comparison with other approaches

The relationship between object identification, counting and object classification has been known in philosophical logic for some time [15, 23, 27] but has been ignored in data modeling. Surrogates were introduced in 1976 by Hall et al. [10] because of two reasons: surrogates make it possible for key values to be changed (without representing this as the deletion of an entity and the subsequent creation of another entity) and surrogates make it possible to represent the difference between two different entities that would otherwise be indistinguishable in the DB. These are basically the two reasons we listed for introducing identifiers, except that Hall et al. apply these reasons to DB systems and not to models of the UoD. Hall's concept of surrogate has been adopted in RM/T [7]. Khoshafian and Copeland [13] introduced the concept of oid, using the same arguments as Hall et al. use to introduce surrogates, but in addition, they argue that the identity exchange problem would be solved by the use of oid's. We think this is true only if one takes the concept of oid literally, which means that it is taken in a form that is practically unrealizable and sometimes undesirable. In practice, pseudo-oid's are used and these do not completely solve the identity exchange problem.

The concepts of oid, key and surrogate have been known for quite some time, but we think the differences between them are not realized sufficiently. Some authors use for example a concept of oid that is closer to our concept of a surrogate than to our concept of an oid (e.g. [8, 14]). We think that our oid concept has the advantage of making clear what the issues around object identification are and what the use of oid's is, while at the same time making clear that there are practical (and ethical) limits on its realizability. Kent [12] gives a good analysis of the use of oid's.

The distinction between role classes and object classes corresponds roughly to the distinction between nominal kinds and natural kinds in philosophical logic [21]. The concept of a role seems to have surfaced in computer science around 1977. Bachman and Daya [2] introduced it in the context of a network approach to data modeling and Bobrow and Winograd used the closely related concept of “perspective” in the knowledge representation language KRL [4]. Reimer [18] surveys the uses of the concept till 1985.

For Bachman and Daya, a role is a “behavior pattern which may be assumed by entities of different kinds.” They allow a role to be played by instances of different types of entities (e.g. persons, companies and government organizations can all play the role of employer) but seem to confuse this with the possibility of one role to be played by several entities simultaneously. They do not consider the possibility of roles being played by other roles and have no concept of role (or object-) identifier.

Sciore [22] defines roles in a system that combines features of inheritance by delegation [16] and class-based inheritance. There is in his model strictly no inheritance relationship between classes, but instances of classes are all roles (our terminology) which can be related to each other by what we call a *played-by* relationship. There are no explicit identifiers, as we have, and there is no distinction between roles and objects. We have given our arguments for explicit rid’s and for the role/object distinction earlier. The *played-by* relation must be a partial order according to Sciore, although he only considers trees in his paper. A role can delegate the answering of a message to its player in the *played-by* tree. As in all prototype systems, there is no fixed delegation structure, but unlike other prototype systems, there is a set of roles that can be used as templates for role instantiation and that are related by the intended class inheritance structure. Roles need not be instantiated according to this inheritance structure though, nor do they have to keep the inheritance pattern defined by the templates. This gives the flexibility to change roles, but we think this brings in the disadvantages of prototype-based systems in data modeling. Our model is a lot closer to the class-based end of the spectrum and yet allows (the analog of) class migration.

Richardson and Schwarz [19] introduce roles under the name of aspects. They do allow roles to be played by other roles, but do not use rid’s and hence cannot represent the possibility of one object (or role) playing several instances of one role class simultaneously. Roles can be played by entities of different types in their model. Su [24] gives a declarative semantics of transactions consisting of sequences of role changes and proves some decidability results concerning soundness and completeness of transactions with respect to trace sets. These results have general validity and are applicable to our work as well as to others’.

Pernici [17] introduces roles in a language for office automatization. She does not consider role identifiers but does consider the problems related to the parallel execution of life cycles connected to different roles played by one object. We did not consider this problem in this paper and refer the reader to work reported elsewhere [25, 26].

Finally, a comparison with ER modeling may be useful. In extensions of the ER model, there is usually an *is-a* relationship to denote specialization. The information that an instance of the specialization is identical to an instance of the generalization is part of the meaning of the *is-a* relationship. In figure 6, for each *CAR* instance there is exactly one *VEHICLE* instance identical to it and for each *VEHICLE* instance there is at most one *CAR* instance identical to it. When the schema in figure 6 is transformed into a relational DB schema, the result will typically be two relation schemas, one for *CAR* and one for *VEHICLE*, both using the same identifier as primary key [3].

To accommodate the *played-by* relationship in extended ER diagrams, it could also be given special status, with relaxed cardinality constraints so that one person could be two

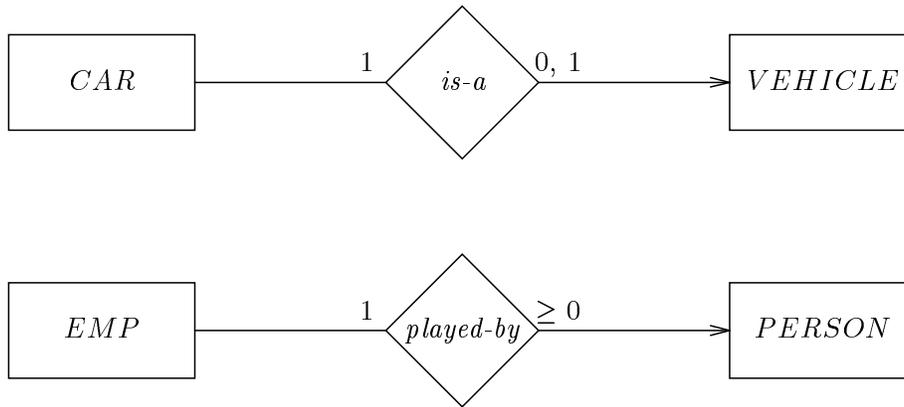


Figure 6: ER representation of *is-a* and *played-by* specializations.

or more employees (figure 6). The information that roles have their own globally unique identifiers would be part of the meaning of the *played-by* relationship. When a *played-by* relationship is translated into a relational DB schema, it would result in two relation schemas, where the role schema would contain its own identifier as primary key and the identifier for its player as a foreign key. This implies the referential integrity constraint plus the constraint that null is not allowed.

Although existence dependence is crucial for roles and weak entities, they are not the same. Roles are identified in a globally unique way whereas a weak entity only needs a unique identifier within the set of weak entities dependent on one parent entity. In addition, even if a weak entity would receive a globally unique identifier (thereby ceasing to be a weak entity), we would probably get an object and not a role. The children of an employee are not roles played by an employee and order lines are not roles of an order.

9 Conclusions

In this paper, we made precise the distinctions between oid's, keys and surrogates. We argued that the distinctions between these three are not sufficiently realized in object-oriented modeling. We also argued that there is an important relationship between object identification and classification that comes out if we analyze the concept of object identification.

Making the relationship between object identification and classification clear, made it possible to bring into focus the reasons why we need to distinguish object classes from role classes. An existing object can play zero or more roles, and a role can only exist by being part of the state of an existing object. To discover whether a class is a role class or an object class, the question whether the class instances can exist independently, can be asked.

We distinguished two kinds of inheritance relations, which we have called *is-a*- and *played-by* inheritance, respectively. Subclass inheritance can exist between object classes and also between role-classes, but not between the two kinds of classes intermixed. Role-oriented inheritance can exist from an object class to a role class, and from role classes to other role classes. Each object and role has a globally unique unchangeable identifier, and this is the basis for the mechanism of *is-a* inheritance. A possible mechanism to implement *played-by* inheritance is delegation. This imports the flexibility advantage of prototype systems without sacrificing the advantage of well-structured models of class-based systems. Role-oriented inheritance is often confused with *is-a* inheritance, as the ubiquitous but erroneous

examples of *STUDENT is-a PERSON* and *EMP is-a PERSON* show.

Finally, we listed some problems connected to pure oid's and showed that in practice, pseudo-oid's are used. Since the database merge problem is only solved by pure oid's, this problem persists in practice, even with object-oriented databases.

References

- [1] 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, *The First International Conference on Deductive and Object-Oriented Databases*, pages 40–57, december 1989.
- [2] C.W. Bachman and M. Daya. The role concept in data models. In *Proceedings of the Third International Conference on Very Large Databases*, pages 464–476, 1977.
- [3] C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin/Cummings, 1992.
- [4] D.G. Bobrow and T. Winograd. An overview of KRL, a knowledge representation language. *Cognitive Science*, 1:3–46, 1977.
- [5] D. Chaum. Security without identification: transaction systems to make Big Brother obsolete. *Communications of the ACM*, 28:1030–1044, 1985.
- [6] R.A. Clarke. Information technology and dataveillance. *Communications of the ACM*, 31(5):498–512, May 1988.
- [7] E.F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4:397–434, 1979.
- [8] C.J. Date. *An Introduction to Database Systems*, volume 1. Addison-Wesley, 5th edition, 1990.
- [9] L.T.F. Gamut. *Logic, Language and Meaning 2: Intensional Logic and Logical Grammar*. University of Chicago Press, 1991. L.T.F. Gamut is a pseudonym for J.F.A.K. van Benthem, J. Groenendijk, D. de Jongh, M. Stokhof, and H. Verkuyl.
- [10] P. Hall, J. Owlett, and S. Todd. Relations and entities. In G.M. Nijssen, editor, *Modelling in Database Management Systems*, pages 201–220. North-Holland, 1976.
- [11] W. Kent. The breakdown of the information model in MDBs. *Sigmod record*, 20(4):10–15, December 1991.
- [12] W. Kent. A rigorous model of object reference, identity, and existence. *Journal of Object-Oriented Programming*, 4(3):28–36, June 1991.
- [13] S.N. Khoshafian and G.P. Copeland. Object identity. In *Object-Oriented Programming Systems, Languages and Applications*, pages 406–416, 1986. SIGPLAN Notices 22 (12).
- [14] W. Kim. *Introduction to Object-Oriented Databases*. MIT Press, 1990.
- [15] J. van Leeuwen and H. Zeevat. Identity and common nouns in intensional logic. In J. Groenendijk, M. Stokhof, and F. Veltman, editors, *Proceedings of the Sixth Amsterdam Colloquium*, pages 219–241, Amsterdam, 1987. Institute for Language, Logic and Information.
- [16] H. Lieberman. Using prototypical objects to implement shared behavior in object oriented systems. In N. Meyrowitz, editor, *Object-Oriented Programming: Systems, Languages and Applications*, pages 214–223, October 1986.
- [17] B. Pernici. Objects with roles. In *IEEE/ACM Conference on Office Information Systems*, Cambridge, Mass., 1990.

- [18] U. Reimer. A representation construct for roles. *Data and Knowledge Engineering*, 1:253–251, 1985.
- [19] J. Richardson and P. Schwartz. Aspects: Extending objects to support multiple, independent roles. In *ACM-SIGMOD International Conference on Management of Data*, pages 298–307, Denver, Colorado, May 1991. ACM. Sigmod Record, Vol. 20.
- [20] M. Rotenberg. Prepared testimony and statement of Marc Rotenberg, Director, Washington Office, Computer Professionals for Social responsibility (CPSR), on the use of the social security number as a national identifier, before The Subcommittee on Social Security, Committee on Ways and Means, U.S. House of Representatives, February 27, 1991. *Computers and Society*, 21:13–19, October 1991.
- [21] S.P. Schwartz. Natural and nominal kinds. *Mind*, 1980.
- [22] E. Sciore. Object specialization. *ACM Transactions on Information Systems*, 7(2):103–122, 1989.
- [23] P. Strawson. *Individuals*. Methuen, 1959.
- [24] J. Su. Dynamic constraints and object migration. In G.M. Lohman, A. Sernadas, and R. Camps, editors, *Proceedings of the Sixteenth International Conference on Very Large Databases*, pages 233–242, Barcelona, Spain, September 3-6 1991.
- [25] R.J. Wieringa. *Algebraic Foundations for Dynamic Conceptual Models*. PhD thesis, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, May 1990.
- [26] R.J. Wieringa. Steps towards a method for the formal modeling of dynamic objects. *Data and Knowledge Engineering*, 6:509–540, 1991.
- [27] D. Wiggins. *Sameness and Substance*. Basil Blackwell, 1980.
- [28] S.B. Zdonik and Maier D. Fundamentals of object-oriented databases. In S.B. Zdonik and D. Maier, editors, *Readings in Object-Oriented Database Systems*, pages 1–36. Morgan Kaufmann, 1990.

Contents

1	Introduction	1
2	Object identifiers, keys, and surrogates	2
2.1	Reasons for introducing object identifiers	2
2.2	Oid's and keys	3
2.3	Oid's and surrogates	4
3	Object classification and counting	4
4	The representation of roles	5
5	Objects, roles and existence	7
6	Taxonomy	8
7	Problems with oid's	9
8	Comparison with other approaches	10
9	Conclusions	12