A Normal Form Object-Oriented Entity Relationship Diagram

Tok Wang LING and Pit Koon TEO
Department of Information Systems and Computer Science
National University of Singapore

Abstract

A normal form object-oriented entity relationship (OOER) diagram is presented to address a set of OO data modelling issues. These issues include the inability to judge the quality of an OO schema, the presence of inheritance conflicts in ISA hierarchies, the lack of explicit support for different relationship types (m-n, n-ary, recursive etc.) in some OO data models and the lack of general and flexible support for views. Three approaches are described to translate good quality OO schemas from a normal form OOER diagram. The first approach translates OO schemas whose underlying OO data models support the notion of relationship. There are no undesirable redundancies in the translated schemas. The second approach provides for OO data models that do not support the notion of relationship. Some redundancies may arise because of m-n, n-ary relationships, but these can be controlled. The third approach treats each OO schema as an external schema of a conceptual schema represented by a normal form OOER diagram. Redundancies may exist at the external schema level, but no redundancies exist at the conceptual schema level.

1 Introduction

The tremendous interest in the object-oriented (OO) approach has exposed a number of inadequacies [22] which are now widely recognised, e.g. lack of a formal foundation, a navigational, as opposed to a declarative interface, lack of a standard query language etc. Some of these issues have been addressed, but many remain unresolved. In this paper, we focus on a set of OO data modelling issues which can be resolved by leveraging techniques[15, 16, 17] in entity relationship (ER) data modelling[6]. These issues include (1) the inability to judge the quality of an OO schema design, (2) the presence of inheritance conflicts in class hierarchies, (3) the lack of explicit support for different relationship types (m-n, n-ary, recursive etc.) in some OO data models and (4) the lack of general and flexible support for views.

To address these issues, we introduce the notion of a normal form OOER diagram. A normal form OOER diagram extends the classical ER diagram[6] in three ways. First, a normal form OOER diagram incorporates the behavioural notions of methods and derived attributes. The additions of these active components and associated semantics for inheritance and complex objects to an ER diagram produce an OOER diagram. Second, the structural properties of the OOER diagram, which are in fact those of an ER diagram, are in normal form. This guarantees that relations generated from a normal form OOER diagram are either in 3NF or 5NF. A normal form OOER diagram does not have undesirable redundancies. It allows the quality of a schema based on it to be judged. Third, It does not have any inheritance conflicts in its ISA hierarchies. In [21], we have proposed a complete algorithm which resolves inheritance conflicts in ISA hierarchies.

In this paper, we describe the translation of OO schemas from a normal form OOER diagram. Each entity type can be translated directly into a class, but, generally, three approaches can be adopted to handle the translations for relationship sets. The first two approaches depend on whether the underlying OO data model for a translated OO schema provides explicit support for the notion of relationship. If the notion of relationship is supported, then each m-n, n-ary or recursive relationship set can be translated into a class in the OO schema. There are no undesirable redundancies in the translated schemas. Otherwise, an embedded, inter-class reference is used to link classes implicitly. In this case, some redundancies may arise because of m-n and n-ary relationships, but these can be controlled. The third approach is a flexible view mechanism that treats each OO schema as a view of a normal form OOER diagram. We propose a three level ER-based schema architecture with the conceptual schema represented by a normal form OOER diagram. Each OO schema is treated as an external schema that is generated from the conceptual schema by using a set of mapping rules. In [20], mapping rules have been defined to generate O2 external schemas from an OOER diagram. The same rules can be used to generate O2 external schemas from a normal form OOER diagram. Distinct sets of mapping rules[16, 17] can be used to generate hierarchical, network, ER diagrams, and nested relations external schemas from a normal form OOER diagram. This approach is flexible in the sense that users can choose to view the conceptual schema in ways that they wish to and are comfortable with. The advantages of these different data models (OO, hierarchical, network, relational, etc.) are preserved at the external schema level.

Section 2 discusses the motivation behind this paper. Section 3 describes the ER model and some relevant extensions. The concept of an OOER diagram is introduced in this section. In Section 4, the normal form OOER diagram is defined. Section 5 discusses the steps to transform an OOER diagram into a normal form OOER diagram. We also explain how the use of a normal form OOER diagram can be used to address the problems mentioned in Section 2. Section 6 discusses, through examples, the generation of OO schemas from a normal form OOER diagram. Section 7 concludes the paper.

2 Motivation

This paper is motivated by the following observations. First, it is difficult to judge the quality of an OO database schema design because, unlike the relational model, it does not have normalisation and dependency theories to assist in database schema design. Second, the OO approach uses inter-object references (through pointers) and the class hierarchy to represent relationships between classes. Different relationship types (m-n, n-ary, recursive etc.) are represented somewhat imperfectly. For example, consider the following schema adopted from [14]:

$(ob1, < name : "john", spouse : ob2 >)$

$(ob2, < name : "mary", spouse : ob1 >)$

where ob1 and ob2 are OIDs. There is an inverse relationship reference (a redundant relationship) in the spouse field that can introduce maintenance problems when there are changes in the spouse field, e.g. if "john" and "mary" are divorced, then references in the spouse field of both "john" and "mary" must be updated. This is a contradiction to the easy maintainability objective of the OO paradigm. To resolve this problem, the relationship between "john" and "mary" should be made explicit. As another example, consider the SUPPLIER (S), PART (P), SUPPLIES (SP) database [8], in which S and P are related by an m-n relationship type SP. Using inter-object references, each supplier object is associated with a set of part objects and their respective quantities. Such a structure does not facilitate symmetric queries. To circumvent this deficiency, each part object is then associated with a set of supplier objects and their respective quantities. This introduces redundancy which may lead to updating anomalies. The problems are similar to those of hierarchical models. These problems are amplified when n-ary ($n > 2$) relationships are considered. There is no feasible solution for representing an n-ary relationship using inter-object references in the OO paradigm.

Third, except for those OODBMSs that are based on the extended relational model (e.g. POSTGRES [23]), most OODBMSs do not fit into the 3-level schema architecture framework as spelled out in the ANSI/X3/SPARC proposal[2] for DBMSs. Users of most OODBMSs are often presented with a large-grained conceptual schema, with little or no facility for defining views or external schemas. Some proposals for views in OODBMSs have been made[1, 11, 24], but the incorporation of a flexible and general view mechanism in OODBMSs is still a research issue.

Fourth, there can be inheritance conflicts in ISA hierarchies. Several techniques have been proposed to resolve these conflicts, e.g. choosing the first in a list of superclasses, as in ORION[4], using type information as in IRIS[10], denying the creation of (sub)classes with conflicting attributes, as in POSTGRES[23], and explicitly choosing the required property to inherit[9], as in O2 etc. These techniques are generally not satisfactory. For example, Orion's technique is quite arbitrary and may not yield the required semantics, while POSTGRES's approach is rather inflexible. It is not clear how IRIS determines type specificity. In O2, if two conflicting properties have different semantics, then explicitly choosing one of the two properties to inherit will preclude the user from inheriting the other property. This may not be desirable. In [21], we have proposed a complete algorithm which resolves inheritance conflicts by examining the semantics of the conflicting properties.

3 An Object-Oriented Entity Relationship Diagram

The ER approach for database schema design uses the concepts of entity type and relationship set, each of which has associated structural properties or attributes. An attribute can be single valued, multi-valued or composite. Different relationship sets are allowed, e.g. m-n, n-ary, recursive, existence dependent (EX) and identifier dependent (ID) weak relationship sets, special relationship sets such as ISA, UNION, INTERSECTION etc. The structure of a database can be represented by using an entity-relationship diagram.

Many extensions have been made to the ER approach to improve its usefulness as a database design tool. In [15], a normal form for ER diagrams was introduced which allows the 'goodness' of ER diagrams to be judged. Good normal form relations, either in 3NF or 5NF, can be generated from a normal form ER diagram by using the techniques introduced in [15].

In [16, 17], the normal form ER diagram was used to represent the conceptual schema of a three level schema architecture. External schemas based on the hierarchical, network, ER and nested relations data models can be generated from the conceptual schema by using distinct sets of mapping rules. In effect, this approach allows a user to have different views of the normal form ER diagram.

We believe that this approach can be extended to support OO external schemas, and to resolve the problems mentioned in Section 2. Specifically, all the structural properties of the OO approach can be derived (generated) from an ER diagram[20]. For instance, an OO class hierarchy can be directly represented using the ER relationship type 'ISA'. A composite object[12], e.g. a VEHICLE and its component parts, can be represented by the ER relationship type IS-PART-OF. Existentially dependent objects [13] which cannot exist independently of their parent objects correspond to ER weak entities. Each weak entity type is associated with a parent (regular) entity type through an existence dependent (EX) relationship or an identifier dependent (ID) relationship. The OO approach also supports the concept of a *complex object*. Briefly, an object may refer to another object which, in turn, may reference other objects and so on, leading to a complex nested structure. In a complex object, the relationships with and among these component objects are not made explicit, but embedded within each object as inter-object references. Note that a composite object is a complex object in which the relationships between an object and its component objects are IS-PART-OF relationships. In the ER approach, the relationships among components of a complex object are made explicit. This is a strong advantage of the ER approach.

The ER model does not have the concept of methods. Furthermore, in the ER model, there are no concepts such as overloading and overriding of properties which are used in OO inheritance hierarchies. These concepts can be incorporated in the ER approach, but inheritance conflicts can occur in ISA hierarchies and must be resolved.

In this section, we first define the notion of an OOER diagram, and then explain what we mean by inheritance conflicts in ISA hierarchies. In the rest of this paper, when context requires, we use the term class to refer to entity type in order to conform to usual OO notations.

Definition 1: An object-oriented entity relationship (OOER) diagram is an ER diagram augmented with methods and derived attributes. An OOER diagram is specifically characterised by the following:

- Methods and derived attributes can be defined for both entity types and relationship sets in an OOER diagram. We use the generic term *properties* to refer to methods, attributes and derived attributes.

  Methods and derived attributes are represented in the same way as attributes, except that they are suffixed by a pair of parentheses. A line ending in a single (double) arrowhead(s) links an entity type or relationship set to a method or derived attribute that returns a single (multiple) value(s), similar to the notation for single-valued (multi-valued) attributes. A line without arrowhead is used to link an entity type or relationship set to a method that performs some generally useful functions and does not return a value, e.g. printing an object's status, firing an employee etc; the cardinality information associated with arrowheads is meaningless in this context.

- Suppose A and B are entity types such that A ISA B. Then A inherits, and possibly override, the attributes and methods of B.

- Components of complex objects belong to classes which are modelled as entity types in the OOER diagram. The relationship sets among these entity types are explicitly represented in the OOER diagram, using the usual notation for representing relationship sets in classical ER diagrams.

Figure 1 is an example OOER diagram which will be used to explain some of the concepts. In Figure 1, DOCTOR and NURSE both have a method called *bonus()* which returns a single value, while EMPLOYEE and PATIENT both have a derived attribute called *age()* which also returns a single value. PATIENT has a method called *update()*, which does not return a value. Note the line linking *update()* to PATIENT.

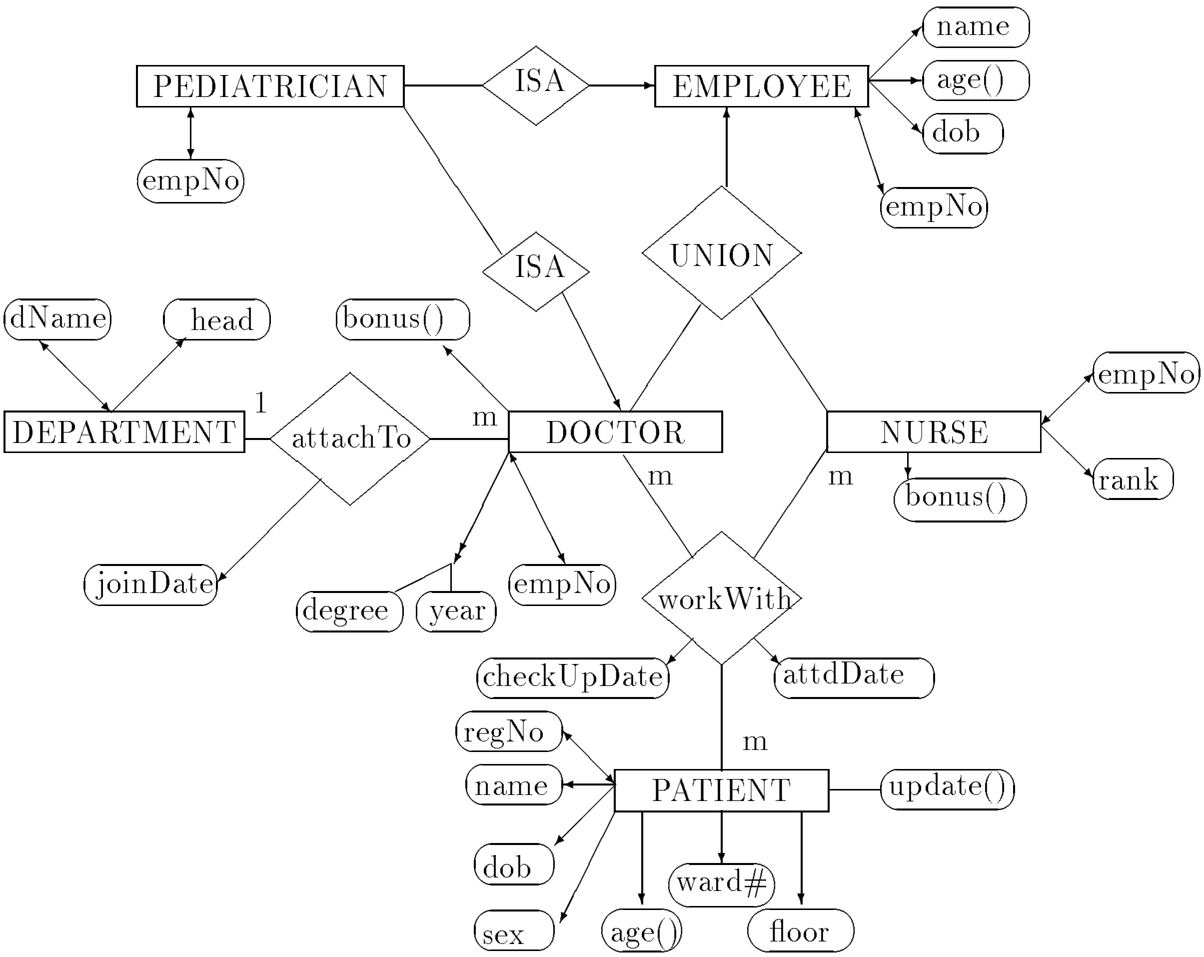

Fig 1. An OOER diagram

Entity types related through the ISA relationship are organised into an ISA hierarchy. Using OO terminology, in a single inheritance situation, a class (or entity type) A has only one direct superclass B. In a multiple inheritance situation, a class A has two or more direct superclasses B1, B2,.., Bn ($n > 1$). A class can inherit properties from its superclasses, if any.

Definition 2 A property is *specified* in a class if it is either defined or redefined for the class. A *redefined* property overloads a similarly named property in some superclass(es) of the class.

As a general rule, only properties specified in an entity type (class) are represented in OOER diagrams. Sometimes, for clarity, we allow inherited key attributes to be explicitly represented for an entity type. For example, in Figure 1, the attribute *empNo* is explicitly represented in DOCTOR and NURSE even though *empNo* is specified in EMPLOYEE, and inherited by both DOCTOR and

NURSE.

Definition 3 An inherited property is *well-defined* if it is specified in one and only one superclass, possibly indirect.

In Figure 2, method p1() and attribute p2 are defined in class Z. Method p1() is redefined in class C and hence are explicitly represented as a property of class C in the OOER diagram. Class B inherits p1() and p2 from class Z, while class C inherits p2 from class Z. Since these properties are inherited rather than specified in the classes, they are not explicitly shown on the OOER diagram.

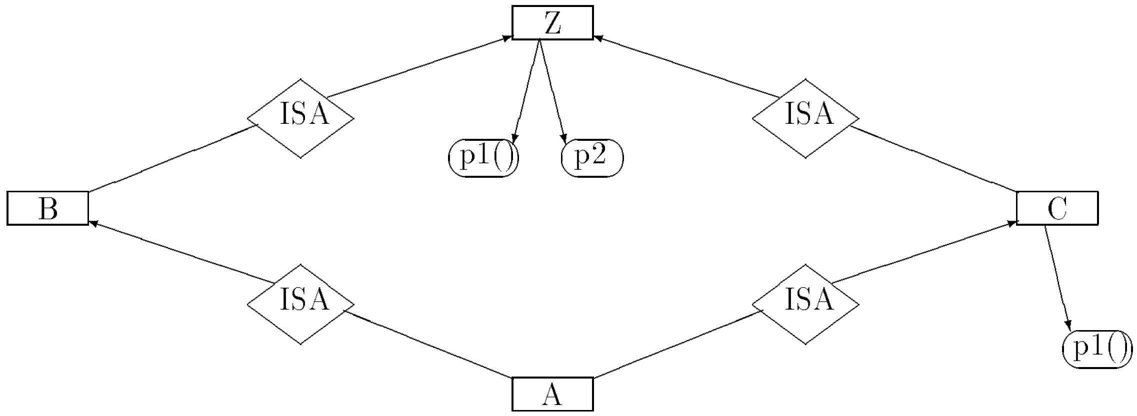

Fig 2. An Inheritance Diagram

Definition 4 A *conflict situation* exists when an inherited property is not well-defined, i.e. two or more superclasses specify the same property.

In Figure 2, classes B and C have commonly named properties p1() and p2, but only p1() contributes to a conflict situation in class A. Method p1() is not well defined in A (there are two classes Z and C which specify p1()), i.e. class A has an inheritance conflict involving p1(), but p2 is well defined (only class Z defines p2).

In [21], we propose that conflicts occur because of the following reasons: (1) redundant ISA relationships, (2) poor or erroneous design, (3) properties in superclasses with the same name and same semantics, and (4) properties in superclasses with the same name and different semantics. A complete algorithm is proposed there which resolves inheritance conflicts systematically by considering the semantics of properties.

An OOER diagram may have undesirable redundancies and inheritance conflicts in its ISA hierarchies. For example, in Figure 1, suppose the functional dependency: $ward\# \rightarrow floor$ holds for

PATIENT. Then, there will be redundancy if PATIENT records are stored based on the attributes for PATIENT shown in Figure 1. Furthermore, the dependency: $ward\# \rightarrow floor$ is not explicitly depicted on the OOER diagram. As another example, suppose CARING_DOCTOR is a subclass of both DOCTOR and NURSE (a multiple inheritance situation) in Figure 1. DOCTOR and NURSE both have a method *bonus()*. Then CARING_DOCTOR has a conflict situation which must be resolved. The OOER diagram can be improved by removing undesirable redundancies and inheritance conflicts from ISA hierarchies. In the next section, we define a normal form for OOER diagram that achieves these objectives.

4 Normal Form OOER Diagram

The objectives for defining a normal form OOER diagram from an OOER diagram are :

1. to capture and preserve the semantics of the real world, in terms of functional, multi-valued and join dependencies, by representing them explicitly in the OOER diagram.

2. to ensure that all the relationships represented in the OOER diagram are non-redundant, i.e. none of the relationships can be derived from other relationships.

3. to ensure that the relations translated from the ER diagram are in good normal form, either in 3NF or 5NF.

4. to ensure that all inheritance conflicts are eliminated from ISA hierarchies.

The concept of a normal form OOER diagram depends on the twin concepts of an *entity type normal form* and a *relationship set normal form*. Entity type and relationship set normal forms have been defined in [15]. The results there can be used directly because they improve on the structural properties of an OOER diagram and do not depend on the behavioural properties of the OOER diagram. We reproduce these results here for completeness, slightly modified to account for the concept of an OOER diagram.

<u>Definition 5</u>: Let E be an entity type and K be its identifier in an OOER diagram. The set of basic dependencies of E, BD(E), is defined as follows:

1. For each many-to-one attribute A of E, $K \rightarrow A$ is a functional dependency (FD) in BD(E).

2. For each one-to-many multivalued attribute A of E, $A \rightarrow K$ is a FD in BD(E).

3. For each one-to-many and many-to-many multivalued attribute A of E, $K \rightarrow\rightarrow A$ is a multivalued dependency (MVD) in BD(E).

4. For each key K1 of E which is not the identifier of E, $K \rightarrow K1$ and $K1 \rightarrow K$ are FDs in BD(E).

5. No other FDs or MVDs are in BD(E).

Informally, the set of basic dependencies of an entity type E are the functional dependencies and multivalued dependencies of E which are explicitly shown in the OOER diagram.

Definition 6: An entity type E of an OOER diagram is said to be in entity normal form (E-NF) if all functional dependencies and multivalued dependencies which only involve attributes of E, can be derived (or implied) from the set of basic dependencies of E, BD(E), by using Armstrong's axioms for functional dependencies and inference rules for multivalued dependencies[5].

In Figure 1, suppose the FD: $ward\# \rightarrow floor$ holds for PATIENT. Then PATIENT is not in entity normal form because this dependency is not in, nor derivable from, the set of basic dependencies for the entity type. We normalise PATIENT in the next section.

Lemma 1: Let E be an entity type of an OOER diagram. If E is in E-NF, then the following statements hold:

1. Each single valued attribute A of E is fully dependent on each key of E which does not contain A, and on each one-to-many attribute of E.

2. All components of any composite single valued attribute A of E are fully dependent on each key of E which does not contain A.

3. There is no non-trivial functional dependencies defined among components of any composite attribute of E.

4. For each one-to-many attribute A of E and for each many-to-many attribute B of E, $A \rightarrow\rightarrow B$ is a strong MVD, i.e. the functional dependency: $A \rightarrow B$ does not hold.

5. For each key K of E and for each multivalued attribute A of E, $K \rightarrow\rightarrow A$ is a strong MVD.

6. No multivalued attribute of E is multi-dependent on a part of a key of E.

7. No component of a composite multivalued attribute of E is multi-dependent on the identifier of E.

Lemma 2: An entity type E of an OOER diagram is in E-NF if and only if it satisfies the following conditions:

1. Any non-trivial canonical form full dependency (i.e. the right side of the FD is a single attribute) $A \rightarrow B$ which only involves attributes and components of composite attributes of E implies
   (a) A is a key of E or A is a one-to-many attribute of E, and
   (b) B is a single valued attribute or B is a component of a composite single valued attribute of E.

2. Any strong MVD: $A \rightarrow\rightarrow B$ which only involves attributes and components of composite attributes of E and in which B is not multi-dependent on any proper subset of A and no proper subset of B is multi-dependent on A, implies
   (a) A is a key of E or A is a one-to-many attribute of E, and
   (b) B is a multivalued attribute of E.

Lemmas 1 and 2 can be proved directly using the definition of an E-NF entity type, Armstrong's axioms for functional dependencies and inference rules for multivalued dependencies.

Definition 7: Let R be a relationship set in an OOER diagram with identifier K and F be the associated set of functional dependencies which only involve the identifiers of the set of entity types participating in R. The set of basic dependencies of R, denoted by BD(R), is defined as follows:

1. For each one-to-one attribute A of R, $K \rightarrow A$ and $A \rightarrow K$ are FDs in BD(R).

2. For each many-to-one attribute A of R, $K \rightarrow A$ is a FD in BD(R).

3. For each one-to-many multivalued attribute A of R, $A \to K$ is a FD in BD(R).

4. For each one-to-many and many-to-many multivalued attribute A of R, $E \to\to A$ is a MVD in BD(R).

5. Let $A \to B$ be a full dependency in F such that A is a set of identifiers of entity types participating in R, and B is the identifier of some entity type participating in R. If A is a key of R or B is part of a key of R then $A \to B$ is a FD in BD(R).

6. No other FDs or MVDs are in BD(R).

Definition 8: A relationship set R of an OOER diagram is said to be in relationship normal form (R-NF) if all functional dependencies and multivalued dependencies which only involve attributes of R and identifiers of entity types participating in R are implied by the set of dependencies of R, i.e. BD(R).

Informally speaking, the set of basic dependencies of a relationship set R includes those functional dependencies and multivalued dependencies which involve attributes of R and are explicitly shown in the OOER diagram. Item 5 of the definition of the basic set of dependencies of a relationship set is to ensure that all relations which correspond to an R-NF relationship set are at least in 3NF.

In Figure 1, suppose the attribute *attdDate* of the relationship set *workWith* is functionally dependent on *empNo* of NURSE and *regNo* of PATIENT only. Then, *workWith* is not in relationship normal form. We normalise *workWith* in the next section.

Lemma 3: Let R be a regular relationship set in an OOER diagram and BD(R) be the set of basic dependencies of R. If R is in R-NF, then the following statements hold:

1. All many-to-one attributes of R are fully dependent on each key, each one-to-one attribute and each one-to-many attribute of R.

2. All components of any composite single valued attribute A of R are fully dependent on each key of R, each one-to-many attribute and each one-to-one attribute of R which is not equal to A.

3. There is no non-trivial functional dependencies defined among components of any composite attribute of R.

4. For each one-to-many (or one-to-one) attribute A of R and for each many-to-many attribute B of R, $A \rightarrow\rightarrow B$ is a strong MVD.

5. For each key K of R and for each multivalued attribute A of R, $K \rightarrow\rightarrow A$ is a strong MVD.

6. No multivalued attribute of R is multi-dependent on a part of a key of E.

7. No component of a composite multivalued attribute of R is multi-dependent on the identifier of E.

This lemma is similar to lemma 1.

Lemma 4: Let R be a regular relationship set in an OOER diagram and BD(R) be the set of basic dependencies of R. R is in R-NF if and only if it satisfies the following conditions:

1. Any non-trivial full dependency: $A \rightarrow B$ which only involves attributes, components of composite attributes, and identifiers of entity types participating in R, and in which B is an attribute or a component of a composite attribute of R implies
   (a) A is a key of R, or A is a one-to-many or one-to-one attribute of R, and
   (b) B is a single valued attribute or a component of a composite single valued attribute of R.

2. Any non-trivial full dependency: $A \rightarrow B$ which only involves attributes of identifiers of entity types participating in R and in which B is a single attribute, implies either A is a key of R or B is part of a key or R.

3. Any strong MVD: $A \rightarrow\rightarrow B$ which only involves attributes, components of composite attributes, and identifiers of entity types participating in R and in which B is not multi-dependent on any proper subset of A and no proper subset of B is multi-dependent on A, implies
   (a) A is a key of R, or A is a one-to-one or one-to-many attribute of R, and
   (b) B is a multivalued attribute of R.

Lemma 4 is similar to lemma 2, except the extra condition (2) in lemma 4.

Definition 9: Let D be an OOER diagram. The set of basic dependencies of D, denoted by BD(D), is defined as the union of the sets of dependencies of all entity types of D and the sets of basic dependencies of all relationship sets of D.

Definition 10: An OOER diagram D is a normal form OOER diagram if it satisfies the following conditions:

1. All property names are distinct and of different semantics within an entity type or relationship set. All key attributes must be distinct, except for key attributes inherited from superclasses in ISA hierarchies.

2. Every entity type in the OOER diagram is in E-NF.

3. Every relationship set in the OOER diagram is in R-NF.

4. All relationships and dependencies are implied by the set of basic dependencies of D.

5. Every relationship set R with no associated attribute defined on it, satisfies the condition that R is not equal to a projection of the join of any two or more other relationship sets

6. There is no inheritance conflicts in its ISA hierarchies.

Informally, in condition one, we adopt a relaxed form of the universal relation assumption. We make use of this condition and explain in more detail in the next section. The second condition of the definition ensures that all relations generated for all entity types are in 5NF. The fourth condition of the definition ensures that the OOER diagram has captured all the relationships and dependencies of the given database. The third and fifth conditions ensure that all relations generated for all regular relationship sets are either in 3NF or 5NF and that there is no relation in BCNF but not in 4NF or 5NF.

5 Deriving a normal form OOER diagram

In this section, we convert the OOER diagram of Figure 1 to a normal form OOER diagram. The result of the conversion is shown in Figure 3. The following steps are adopted in order to convert an OOER diagram to a normal form OOER diagram:

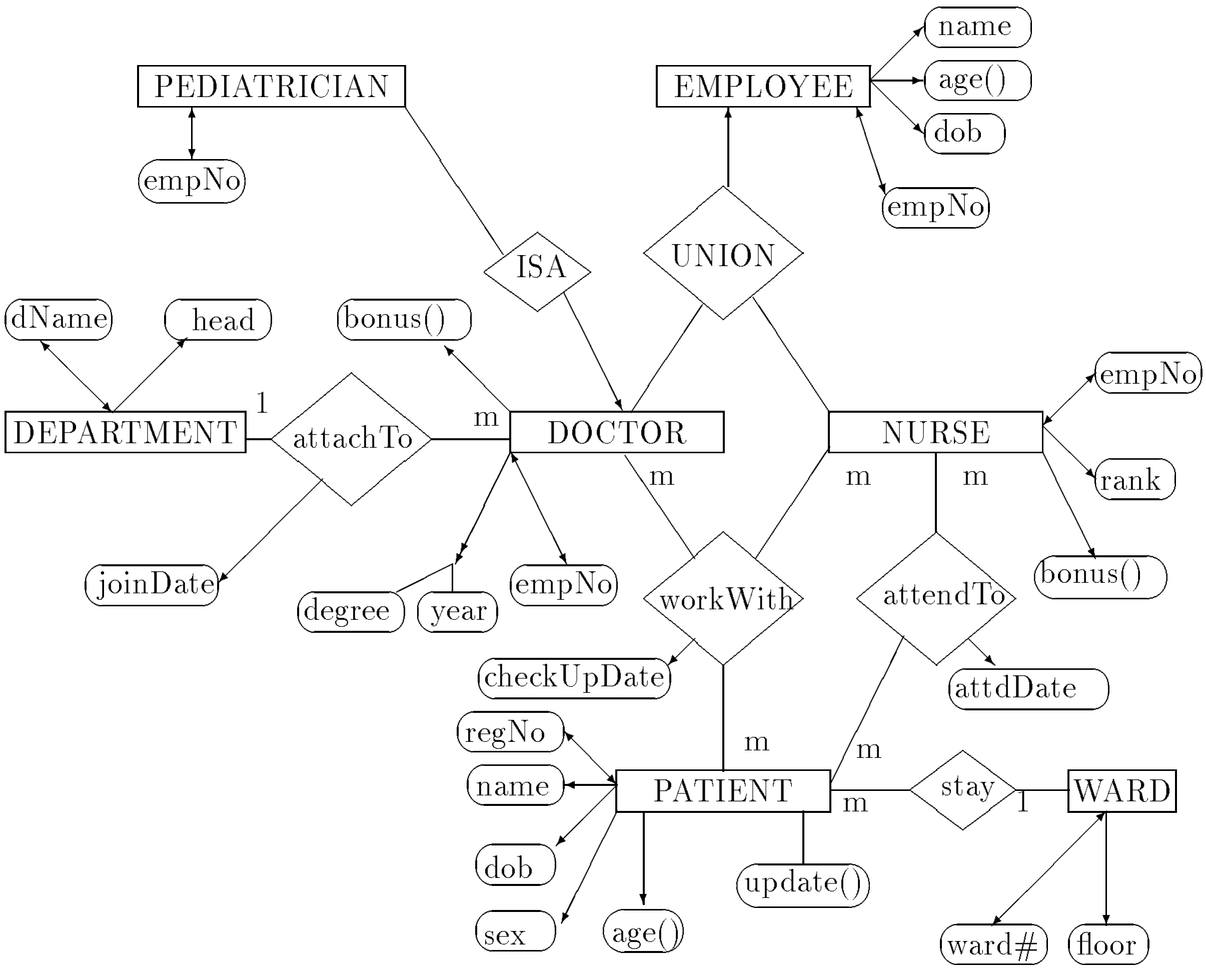

Fig 3. Normal Form OOER Diagram

(Step 1) Ensure that all property names within each entity type and relationship set are distinct and of different semantics. Ensure that all key attributes are uniquely named, other than those inherited from superclasses.

(Step 2) Convert any non E-NF entity type to E-NF. We remove all undesirable functional dependencies and/or multivalued dependencies by introducing new entity types and relationship sets.

(Step 3) Convert any non R-NF relationship set to R-NF. We remove all undesirable functional dependencies, multivalued dependencies, and/or join dependencies either by introducing new entity types and relationship sets or by splitting the relationship set into smaller ones.

(Step 4) Remove those relationship sets which have no associated attributes and is equal to a projection of the join of two or more other relationship sets.

(Step 5) Remove any inheritance conflicts from ISA hierarchies.

In the rest of this section, we examine these six steps in detail and then explain how the problems of Section 2 are addressed.

In step 1, we adopt a relaxed form of the universal relation assumption, as noted in Section 4, under Definition 10. Key attributes must be unique, except for those inherited from superclasses. In Figure 3, for example, DOCTOR and NURSE inherits the key attribute *empNo* from EMPLOYEE. The *empNo* key attribute is therefore not uniquely named in this ISA (or UNION) hierarchy. Strictly, we need not display the *empNo* attributes of DOCTOR and NURSE explicitly on the OOER diagram, since *empNo* is specified in EMPLOYEE. Non-key attributes and method names may not be unique. For example, in Figure 3, both EMPLOYEE and PATIENT have similarly named attributes *name* and *dob*, and derived attribute *age()*. In such cases, the context is sufficient to distinguish among these non-key attributes and methods.

In Step 2, if an entity type is not in normal form, we remove some of the attributes involved and create some new entity types and relationship sets. For example, the PATIENT entity type in Figure 1 is normalised by removing its *ward#* and *floor* attributes and creating a new entity type WARD and relationship set *stay*, as shown in Figure 3. This is to ensure that the FD: $ward\# \rightarrow floor$ is explicitly represented in the OOER diagram.

In Step 3, if a relationship set is not in normal form, we remove some of the attributes involved and create some new entity types and relationship sets, or split the relationship set into two or more smaller relationship sets. For example, in Figure 1, the relationship *workWith* is not in relationship normal form because its attribute *attdDate* is functionally dependent on *empNo* of NURSE and *regNo* of PATIENT only. In Figure 3, we normalise the *workWith* relationship set of Figure 1 by creating a new relationship set *attendTo* with an associated attribute *attdDate*.

In Step 4, we remove any relationship set with no associated attributes which is equal to a projection of the join of two or more other relationship sets. To detect such relationship sets, we require information about the semantic meaning of the relationship sets which can be provided by the database designer or owner. This step also removes any redundant ISA relationships. A direct ISA relationship A ISA C is redundant if there is some entity B such that A ISA B and B ISA C.

This redundancy is a result of the transitivity of the ISA relationship set. In Figure 1, the ISA relationship set between PEDIATRICIAN and EMPLOYEE is redundant, and must be removed, as shown in Figure 3.

In Step 5, inheritance conflicts in ISA hierarchies are removed by using the method introduced in [21]. We will not describe the details here, but give an example to illustrate how we can remove inheritance conflicts.

Example: In Section 3, we describe a conflict situation involving a class CARING_DOCTOR which is a subclass of both DOCTOR and NURSE (a multiple inheritance situation). DOCTOR and NURSE both specify a method *bonus()* which causes a conflict situation for CARING_DOCTOR. This conflict situation can be resolved by considering the semantics of *bonus()*. If the semantics (implementation) of *bonus()* is the same for both DOCTOR and NURSE, then it is better to factor *bonus()* to a more general class EMPLOYEE. If the semantics of *bonus()* is different, then several approaches can be taken to resolve the conflict[21]. First, we can rename the property *bonus()* for either DOCTOR or NURSE, or both, so that CARING_DOCTOR can inherit both these methods. This approach is not consistent with the use of similarly named properties in ISA hierarchies in order to support polymorphism. Second, CARING_DOCTOR may explicitly choose to inherit from either DOCTOR or NURSE, depending on the required semantics. This precludes CARING_DOCTOR from inheriting both *bonus()* methods. Third, CARING_DOCTOR may specify its own definition of the *bonus()* method, i.e. it overrides the definition of *bonus()* in its two superclasses.

We now explain how the normal form OOER diagram can be used to address three of the problems of Section 2. The problem related to views is addressed in the next section. Firstly, the normal form OOER diagram preserves application semantics by explicitly capturing entity types, relationship sets (m-n, n-ary, recursive, ISA etc.) and their attributes. By providing strong support for different relationship types among entity types, the ER approach eliminates the problems that the OO approach faces in representing these relationship types. For instance, the problems of the 'spouse' relationship of Section 2, or the relationship between suppliers and parts (e.g. presence of redundant pointers, problems in supporting symmetric queries etc.) are eliminated.

Secondly, a normal form OOER diagram uses established ER, normalisation and dependency theories to provide a strong theoretical foundation. It is therefore possible to judge the quality of a schema based on the ER approach.

Thirdly, a normal form OOER diagram does not have any inheritance conflicts in its ISA hierarchies. This is by its definition.

6 Generating OO Schemas

Each entity type in a normal form OOER diagram can be translated directly into a class of the OO schema. However, every translation of an OO schema from a normal form OOER diagram must handle relationship sets among entity types. Generally, three approaches can be adopted. The first two approaches depend on whether the underlying OO data model of the OO schema provides explicit support for the notion of relationship.

In approach one, the underlying OO data model supports the notion of relationship directly. Each m-n, n-ary or recursive relationship set can be mapped directly into a class in the OO schema. Pointers are then set up to link the relationship class to the classes participating in the relationship set. For example, we can define, using the O2 notation[9], classes for the NURSE entity type and the *attendTo* relationship set of Figure 3, as follows:

```
class NURSE inherits EMPLOYEE type tuple
    (rank : string
    )
     method bonus() : integer
end;

class attendTo type tuple
    (nurse : NURSE,
     patient : PATIENT,
     attdDate : integer
    )
end;
```

Note that EMPLOYEE and PATIENT are classes defined elsewhere, and that NURSE has a method *bonus()* that returns an integer value. Using approach one, each translated OO schema is of good quality. There are no undesirable redundancies because the schema is translated from a normal form OOER diagram.

Approach two provides for those OO data models which do not support the notion of relationship. In this approach, the class definition for each entity type that participates in a relationship set with other entity types must have an embedded reference that refers to the participating entity types, and any attributes associated with the relationship set[20]. For example, the NURSE entity type of Figure 3 will have a tuple-valued attribute[20] that references a PATIENT class and the *attdDate* attribute associated with the relationship set *attendTo*, as follows (again using the O2 notation):

```
class NURSE inherits EMPLOYEE type tuple
     (rank : string,
      attendTo : set(tuple(patient : PATIENT, attdDate : string)),
      workWith : set(tuple(doc : DOCTOR, patient : PATIENT, checkUpDate : string))
     )
      method bonus() : integer
end;
class PATIENT type tuple
     (regNo : string,
      name : string,
      dob : string,
      sex : char,
      attendTo : set(tuple(nurse : NURSE, attdDate : string)),
      workWith : set(tuple(doc : DOCTOR, nurse : NURSE, checkUpDate : string))
     )
      method age() : integer,
             update()
end;
```

Note that EMPLOYEE is a class defined elsewhere and PATIENT has a method *update()* which does not return any value. Using approach two, each translated OO schema does not contain undesirable redundancies except those arising from the existence of m-n and n-ary relationship sets. These redundancies can be controlled at the system level, similar in concept to the physical/virtual pairing feature in IMS[8]. The spouse schema of Section 2 is another example of using approach two.

The third and final approach treats each OO schema as a view of a normal form OOER diagram. We propose a three level ER-based schema architecture to support this view mechanism. The conceptual schema is represented by a normal form OOER diagram. Each OO schema is a view or external schema of the conceptual schema that is generated by using a set of mapping rules. In [20], a set of mapping rules has been defined to generate O2 external schemas from an OOER diagram. This set of rules can still be applied to a normal form OOER diagram.

Each entity type in the conceptual schema can be mapped to a class in the OO external schema.

If there is a relationship set between two entity types E_1 and E_2, then without loss of generality, E_2 can be a set-valued attribute in the class specification of E_1. For example, the class DOCTOR has an attribute Doc-Pat whose value is a set of entities in the class PATIENT. Note that we can have different views of how the entity types are related in the external schema. For example, an instance of the class PATIENT can have a set of associated doctors while an instance of the class DOCTOR can have a set of associated patients. Any redundancy in the external schema is virtual as no redundancy exists in the conceptual schema.

The following OO external schema, again using the O2 notation, is generated by applying the mapping rules given in [20].

```
class EMPLOYEE type tuple
    (empNo : string,
     name : string,
     dob : integer
    )
     method age() : integer
end;

class DOCTOR inherits EMPLOYEE type tuple
    (qual : set(tuple(year : string, degree : string)),
     Doc-Pat : set(PATIENT)
    )
     method bonus() : integer
end;

class NURSE inherits EMPLOYEE type tuple
    (rank : string,
     Nurse-Pat : set(tuple(patient : PATIENT, attdDate : integer)) /* via attendTo */
    )
     method bonus() : integer
end;

class PATIENT type tuple
    (regNo : string,
     name : string,
     dob : string,
     sex : char,
     Pat-Doc : set(DOCTOR),
     Pat-Nurse : set(NURSE) /* via workWith */
    )
     method age() : integer,
            update()
end;
```

Note that, in Figure 3, NURSE is related to PATIENT via two relationship sets, viz. *attendTo* and *workWith*. Therefore, there are two possible derivations for the view attribute Nurse-Pat in class NURSE and the view attribute Pat-Nurse in class PATIENT. These two possibilities can be automatically generated by a schema design system and presented to the user for selection[18]. In this example, the user chooses to use the *attendTo* relationship set for deriving the Nurse-Pat attribute in NURSE, and the *workWith* relationship set for the Pat-Nurse attribute in PATIENT. Note that it is also possible for the user to specify that Pat-Nurse is derived via *attendTo*, but with the *attdDate* attribute abstracted out.

The OO external schema can include methods which have been defined in the conceptual schema. For example, the entity type EMPLOYEE has a method (derived attribute) called *age()* in the conceptual schema; this method can be included in the class EMPLOYEE in the external schema. This method *age()* is inherited by the classes DOCTOR and NURSE in the external schema. Methods can also be specified in the external schema.

Notice that it is also possible for the user to define a view that computes the transitive closure of a relation, e.g. PART and its component parts. This can use the semi-naive or magic sets methods[3], but the exact implementation is transparent at the external schema level.

Sets of mapping rules can be defined to generate external schemas based on a number of data models (e.g. hierarchical, network, nested relations, relational, ER diagrams, etc.) from a conceptual schema that is represented by a normal form OOER diagram. In [16, 17], for example, mapping rules have been defined to generate external schemas based on hierarchical, network, relational, nested relations, and ER data models from a normal form ER diagram[15]. These rules are still valid for a normal form OOER diagram. The presence of methods and derived attributes in a normal form OOER diagram does not invalidate any of the guidelines in [16, 17]. This is because the guidelines pertain to the structural properties of a normal form ER diagram, whereas methods and derived attributes provide an orthogonal behavioural perspective. Therefore, depending on the needs of users, the appropriate external schemas can be defined. External schemas may not necessarily be normalised, and therefore users may perceive some form of 'redundancy'. However, this redundancy is virtual in the sense that no redundancy exists at the conceptual schema level and data is not stored redundantly.

7 Conclusion

In this paper, we first define the notion of an OOER diagram. An OOER diagram is a classical ER diagram augmented with methods and derived attributes. It has OO semantics of inheritance and provides support for representing complex objects. An OOER diagram may have redundancies and inheritance conflicts in its ISA hierarchies. The concept of a normal form OOER diagram is then proposed which improves on an OOER diagram by removing undesirable redundancies from the OOER diagram, and any inheritance conflicts from its ISA hierarchies. An OOER diagram is of 'good' quality if the OOER diagram is in normal form. A normal form OOER diagram incorporates object-oriented concepts such as methods, inheritance (through the ISA, UNION, DECOMPOSE, INTERSECTION relationships), class (entity or relationship), existentially dependent objects (weak entities) etc. A design criteria is incorporated into the design of a normal form OOER diagram so that inheritance conflicts in ISA hierarchies are avoided or resolved satisfactorily.

By using the normal form OOER diagram as a tool to design OO schemas, we can address a number of OO data modelling inadequacies. These inadequacies include: (1) the inability to judge the quality of an OO database schema, (2) the presence of inheritance conflicts in ISA (class) hierarchies, (3) the lack of explicit support for different types of relationship (m-n, n-ary, recursive etc.) in some OO data models, and (4) the lack of flexible support for views. Object-oriented schemas can be translated from a normal form OOER diagram, but such translations must handle relationship sets that are explicitly represented in the normal form OOER diagram. In this paper, we provide three different approaches to handle the translations. The first approach can be used to translate OO schemas whose underlying OO data models support the notion of relationship. Using this approach, the translated OO schemas do not have any undesirable redundancies because they are translated from a normal form OOER diagram. The second approach is used when the underlying OO data models do not support the notion of relationship. In this case, some redundancies may arise because of the existence of m-n and n-ary relationship sets, but these redundancies can be controlled at the system level. The third approach provides a flexible view mechanism built on an ER-based three level schema architecture. In the third approach, the normal form OOER diagram is the conceptual schema while each OO schema is a view or external schema that is generated from the conceptual schema by using a set of mapping rules. There may be redundancies at the external schema level, but these redundancies are virtual. No redundancies exist at the conceptual schema level.

References

[1] Abiteboul S., *Objects and Views*, Proc Intl. Conf on Management of Data, 1991.

[2] ANSI/X3/SPARC Study Group on Data Base Management Systems, Interim Report, FDT (ACM Sigmod bulletin) Vol 7 No 2,1975.

[3] F. Bancilhon, R. Ramakrishnan, *An Amateur's Introduction to Recursive Query Processing*, Proc. ACM Sigmod Intl Conf. on Management of Data, 1986, pp. 16-52.

[4] J. Banerjee et al, *Data model issues for object-oriented applications*, ACM Transactions on Office Information Systems, Vol 5 No 1, Jan 87, pp. 3-26.

[5] C. Beeri, R. Fagin, J. Howard, *A complete axiomatization for functional and multivalued dependencies in database relations*, 3rd ACM SIGMOD Int. Conf. Management of Data, 1977.

[6] Chen, P.P., *The entity-relationship model: toward a unified view of data*, ACM TODS, Vol 1, No 1, 1976.

[7] -, *Third generation database system manifesto*, the Committee for Advanced DBMS Function, Memo No. UCB/ERL M90/28, University of California, Berkeley, Apr 1990.

[8] C. Date, *An introduction to database systems Vol I*, Addison Wesley, 4th edition, 1986.

[9] O. Deux et al, *The O2 system*, Communications of the ACM, Vol. 34, No. 10, Oct 1991, pp. 35-48.

[10] D. Fishman et al,*IRIS: An object-oriented database management system*, ACM Trans Office Information Syst., Vol 5 No 1, Jan 87, pp. 48-69.

[11] B.Hailpern, H.Ossher, *Extending objects to support multiple interfaces and access controls*, IEEE Transactions on Software Engineering, Vol 16, No 11, Nov 1990.

[12] W. Kim et al, *Composite object support in an object-oriented database system*, Proc OOPSLA '87, Orlando, Florida, Oct 87.

[13] W. Kim, *An introduction to object-oriented databases*, MIT Press, 1990.

[14] Lecluse, C., Richard, P. and Velez, F., *O2, an object-oriented data model*, Proc. Int. Conf. on Management of Data, Chicago, Ill, Jun 1988.

[15] T.W. Ling, *A normal form for entity-relationship diagrams*, Proc. 4th International Conference on Entity-Relationship Approach, 1985, pg 24-35.

[16] T.W. Ling, *A three level schema architecture ER-based data base management system*, Proc. 6th International Conference on Entity Relationship Approach, 1987, pp 181-196.

[17] T.W. Ling, *External schemas of entity-relationship based data base management systems*, in Entity-Relationship Approach, C. Batini (Eds.), Elsevier Science Publishers, 1989.

[18] T.W. Ling, M.L. Lee, *A Prolog implementation of an entity-relationship based database management system*, 10th Int. Conf. on the Entity Relationship Approach, Oct 23-25, San Mateo, Ca., 1991.

[19] T.W. Ling, P.K. Teo, *On rules and integrity constraints in database systems*, Information and Software Technology, Vol 34 No 3, Mar 1992.

[20] T.W. Ling, P.K. Teo, L.L. Yan, *Generating object-oriented views from an ER-based conceptual schema*, 3rd Int. Symposium on Database Systems for Advanced Applications, Apr 6-8, Taejon, Korea, 1993.

[21] T.W. Ling, P.K. Teo, *Inheritance conflicts in object-oriented systems*, Proc. Database and Expert Systems Applications (DEXA), Prague, Czech Rep., Sep 1993.

[22] T.W. Ling, P.K. Teo, *Toward resolving inadequacies in object-oriented data models*, Information and Software Technology, Vol 35, No. 5, May 1993.

[23] Lawrence Rowe and Michael Stonebraker, *The Postgres data model*, in The Postgres Papers, Memo No UCB/ERL M86/85 Jun 87 (Revised), University of California, Berkeley.

[24] J. Shilling, P. Sweeney, *Three steps to views: extending the object-oriented paradigm*, Proc. OOPSLA 89.

[25] D. Touretzky, *Implicit ordering of defaults in inheritance systems*, Proc AAAI-84, Austin, Texas, 1984, pp 322-325.