A Knowledge Based System Converting ER Model into an Object-Oriented Database Schema

Il-Yeol Song and Heather M. Godsey
College of Information Studies
Drexel University
Philadelphia, PA 19104

Abstract

In this paper, we discuss a knowledge based system, KERO, which implements a methodology converting an entity-relationship (ER) model into a structurally object-oriented database schema. A set of rules that can convert most semantic constructs of ER models into an OODB schema is discussed. Our method is an improved one which can properly convert many-to-many relationships with non-key attributes and aggregation in ER model. The ideas which can further enhance the converted OODB schema are also presented. We think that the translation methodologies from semantic data models into OODB schema are important since most existing relational database design methodologies are based on the semantic approach and OODB systems must subsume existing relational database systems. The KERO system has been implemented in MIKE, which is a meta-interpreter written in Prolog.

Keywords: Entity-Relationship, Object-oriented databases, database schema, database design, knowledge based system

1 Introduction

The purpose of this paper is to present the design and implementation of a knowledgebased system, KERO, that converts an entity-relationship (ER) model into a structurally object-oriented (OO) database schema.

Our work is a continued effort in developing an easy to use OODB design methodology which could prove to be useful independent of database application domains. We have done a comprehensive survey of object-oriented database design methodologies [Song 1992]. We think that the translation methodologies from semantic data models are important for the following main reasons: Most existing relational database design methodologies are based on the semantic approach. By utilizing existing methodologies, we will not lose the valuable knowledge of many information engineers, obtained throughout the design and use of relational database systems. So the transition from existing systems to OODB systems can be smooth once a set of guidelines is provided. This is particularly true since OODB systems must subsume relational database systems and open to other subsystems [Stonebreaker+ 1990, Kim 1990].

Based on these observations, we have developed a knowledgebased system, KERO, that converts ERDs into OO database schema. We define a meta ER model to keep the input ER model structure, and a meta OO model for an output OODB schema. We present a set of rules converting each of ER model constructs defined in the input ER model into a corresponding OO schema component. Cattell [1991] and Hughes [1991] both have examples converting ERD into OODB schema. Cattell does not show how to deal with aggregation, and Cattell and Hughes do not handle many-to-many relationships with non-key attributes properly. Our work is an enhanced version of the previous work in this area in that we solve some problems of handling many-to-many relationships in [Cattell 1991, Hughes 1991] and discuss aggregation in more detail than [Hughes 1991, Song and Godsey 1993]. The converted OO schema can serve as an initial conceptual schema which can be further enhanced depending on the target OODBMS. We also suggest several ideas about how the converted schema can be semantically further improved. The KERO system has been implemented in MIKE [Eisenstadt+ 1990], which is a public-domain meta-level interpreter of Prolog.

Other approaches using ER-like models for OODB schema design include Gorman and Choobinch [1991], Nachouki, et al. [1991]. Gorman and Choobinch propose OOERM by extending ERD to include methods and message passing structures. However, the diagram becomes very messy, loses intuitiveness, and poses difficulty in representing all messages for most real-world applications. Nachouki, et al. discus how to integrate logical data access in the definition of an OODB schema from an ERD.

The rest of this paper is organized as follows: Section 2 discusses the conceptual design and implementation of our knowledge based system, and Section 3 presents an example of schema translation. Section 4 concludes the paper.

2. Methodology

In this section, we present the design and implementation aspects of our knowledge based system, KERO, which converts ERDs into OODB schema. The design of KERO can be broken down in three main segments: ERD representation, OODB representation, and translation rule development. The ER model which will be converted into an OO model is represented in a meta ER model understood by KERO (see Section 2.1). KERO converts the ER model into a meta OO model for OODB schema (see Section 2.2). The conversion rules are discussed in Section 2.3, the architecture of KERO in Section 2.4, and semantic enhancement to the converted OO model is discussed in Section 2.5.

2.1 ERD Representation

ER model and its variations [Chen 1976, Teorey+ 1986, Batini+ 1991] are the most popular approach for relational database design, and have been widely adopted in CASE tools. As such it is natural to consider the ER model as a starting point of designing an

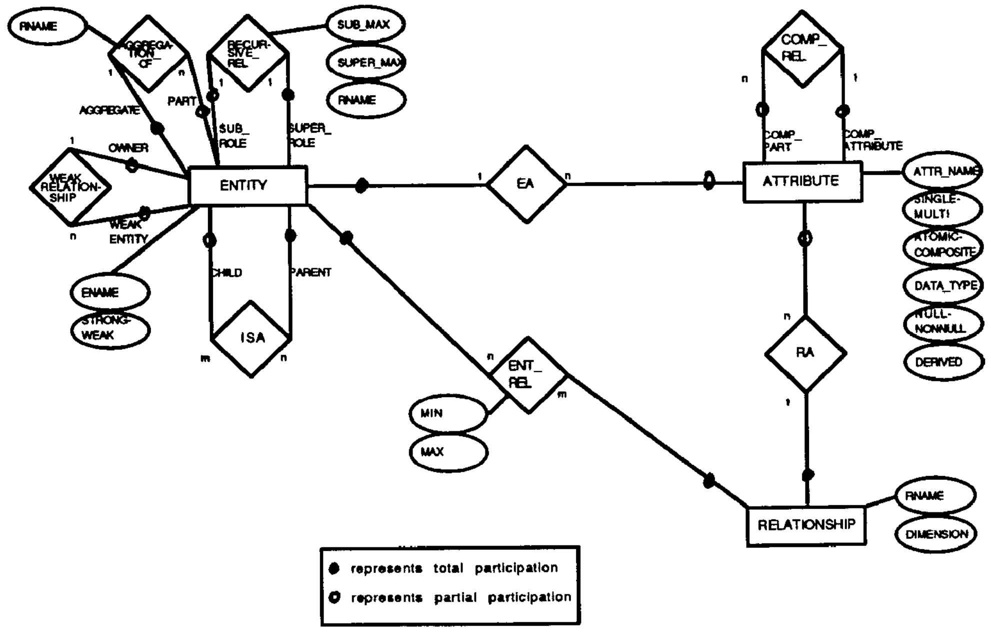

Figure 1 Meta ER Model

OODB schema. There are many complex semantics which may be modeled by the entity relationship diagram. The semantic constructs of the EER (ER henceforth) model which we have chosen for the KERO system include entities, relationships, recursive relationships, ternary relationships, generalization/specialization, inheritance, set-valued attributes, and composite attributes as discussed in Elmasri and Navathe [1989].

The ERD for the representation of the meta-model for the ER model (called *meta ER* model) is shown in Figure 1, and its important semantics are explained below.

- ENTITY represents either a regular (strong) entity or a weak entity.
- ENTITY can have weak entities (WEAK_RELATIONSHIP), and they have an owner entity.
- ENTITY can have recursive relationships (RECURSIVE_REL).
- ENTITY can have many children (sub) entities and many parent (super) entities.
- ENTITY participates in (ENT_REL) relationships, and has (min, max) cardinality constraints.
- ENTITY can have many attributes.
- ATTRIBUTE belongs to either an ENTITY via EA relationship or a RELATIONSHIP via RA.
- ATTRIBUTE can be either a single,multi-valued, or composite attribute.
- A composite ATTRIBUTE consists of several simple attributes.

What is shown in Figure 2 is a relational representation of the above meta ER model.

```
ENTITY(ENAME, WEAK/STRONG, OWNER)
ATTRIBUTE(ENAME, ATTR_NAME, NULL/NON-
        NULL, SINGLE/MULTI_VALUED,
        ATOMIC/COMPOSITE, DATATYPE)
RELATIONSHIP(RNAME, DIMENSION)
ENTITY_RELATIONSHIP(ENAME, RNAME, MIN,
        MAX)
RELATIONSHIP_ATTR(RNAME, ATTR_NAME,
        SINGLE/MULTI, ATOMIC/COMPOSITE,
        DATATYPE)
RECURSIVE(ENAME, SUBROLE, SUPERROLE,
        SUB_MAX, SUPER_MAX, RNAME)
ISA(PARENT, CHILD)
COMPOSITE_ATTR_PARTS(COMP_ATTR, PART,
        SINGLE/MULTI, ATOMIC/COMPOSITE,
        DATATYPE)
```

Figure 2. Relational Schema for meta ER model

For efficiency of implementation, the following was added for ternary relationships and 1:1 and M:N binary relationships.

```
/* for binary: */
MEMBER(RNAME, MEMBER1, MEMBER2)
/* for ternary */
MEMBER(RNAME, MEMBER1, MEMBER2,
        MEMBER3)
```

Note that a ternary relationship can also be represented using the above meta model. The maximum cardinality (MAX attribute in ENT_REL relationship) of a binary relationship is represented using the typical "Look Across" notation (i.e., look across the relationship to find the cardinality, the cardinality is written in the other side of the entity), while that of a ternary relationship is represented using the "Look Here" notation (cardinality is written near the entity side). For example, the CONTROLS and SUPPLY relationships in the ERD of Figure 3 can be represented as in Figure 4 below:

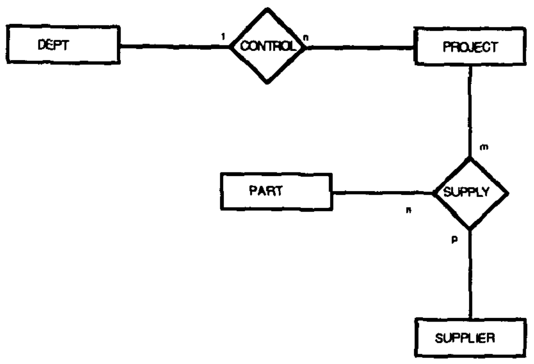

Figure 3. ERD with a Ternary Relationship

relationship(control, 2)
relationship(supply, 3)
ent_rel(department, control, 0, n)
ent_rel(project, control, 1, 1)
ent_rel(project, supply, 1, n)
ent_rel(part, supply, 1, n)
ent_rel(supplier, 0, n)
member(control, department, project)
member(supply, project, part, supplier)

Figure 4. Meta ER Representation of CONTROLS and SUPPLY in Figure 5

An example of a complete ERD is shown in Section 3. It should be pointed out that the ER design approach to OODB schema does not produce operations. Other methodologies should be used to come up with and represent them, but these are beyond the scope of this paper.

2.2 OODB Representation

An OODB schema consists of a collection of *classes*. Each class has a set of *instances*, which are the objects belonging to that class. Each class is an abstract data type which has a group of *attributes* (called instance variables in some OO systems) and a set of operations (or methods) defined on them (classification). These attributes describe properties of instances of a class. Each object has an object identity independent of the values that it contains (identification). Classes form a set of class hierarchies, forming superclass and subclass relationships (generalization). Subclass objects inherit the attributes and operations of superclasses. The class hierarchy becomes a class lattice in the form of directed acyclic graph (DAG) when a class can have more than one parent class (multiple inheritance). It is common practice to refer to a class lattice, as a class hierarchy. Attributes in OODB could be simple attributes, collection attributes (set values, lists, arrays), derived attributes (whose value are computed by procedures or methods), class attributes (whose value represent properties of the class as a whole or summary values, rather than each individual object instance) or *reference attributes* (which point to other object(s)). Simple attributes are atomic attributes which do not have their own components. Examples of the simple attributes are integers, numbers, strings, and other simple values. Set attributes are those which have a set of simple values or other objects. Examples of class attributes are average number of players or uniform color of a baseball team. Reference attributes are used to represent relationships between objects (association). A reference attribute can represent either a single object or a set of objects (*reference-set attribute*). These attributes may be aggregated to form an object, and objects may be aggregated to form composite objects (aggregation).

Our target OO model, currently implemented in KERO, has classification, generalization, and aggregation abstractions. Identification is automatically supported by the target OODBMS in the form of an object identifier. Associations are implemented as a pair of reference attributes and their associated inverse attributes (See Section 2.3 for more on this).

The template class definition used for output from KERO is similar to the syntax used in Hughes [1991], and is shown in Figure 5 as follows:

```
Class <class_name>
        Inherit <other_class>
        Properties
                /* simple attribute: */
                <attribute_name>: <simple data type> ;
                 /*set-valued attribute:*/
                <attribute_name>: set (<simple data type>);
                 /* reference attribute: */
                <attribute_name>: (other_class_name);
                 /* reference-set attribute:*/
                <attribute_name>: set (<other_class_name>);
                  inverse is <other_class_name>.
                        <attribute_name>;
                ...
        Operations
                <operation_name>;
                ....
End.
```

Figure 5. Template for Class Definition

The "Inherit" clause defines the superclass, "Properties" clause defines attributes of the class, and "Operations" clause defines operations which will be added later using other methods. The "Properties" clause shows four different types of attributes.

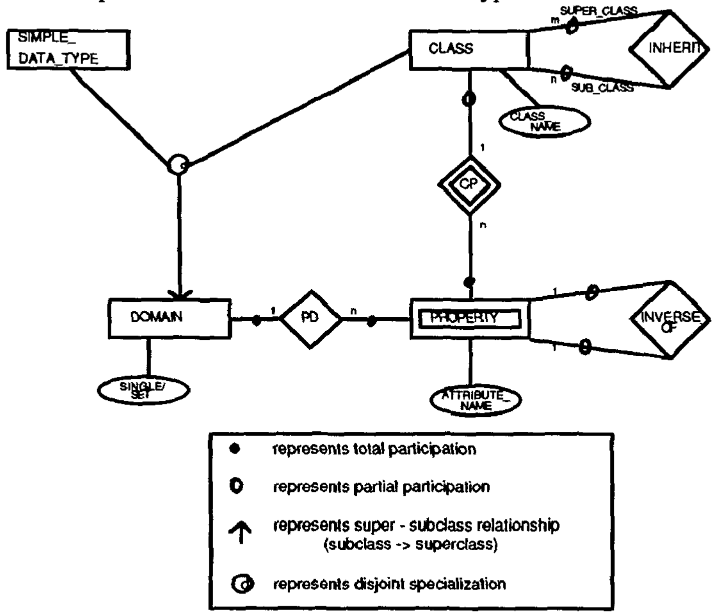
SIMPLE_
DATA_TYPE
CLASS
SUPER_CLASS
m
INHERIT
n
SUB_CLASS
CLASS_NAME
1
CP
n
DOMAIN
1
PD
n
PROPERTY
1
INVERSE OF
1
SINGLE/SET
ATTRIBUTE_NAME
● represents total participation
represents partial participation
represents super - subclass relationship (subclass -> superclass)
represents disjoint specialization

Figure 6 Meta OO model

The ERD representation of meta OO model in Figure 6 can be explained as follows:

- A CLASS has attributes.
- A PROPERTY (Attribute) has a domain.
- A DOMAIN is either a simple data type or a CLASS type.
- Generalization/specialization is handled through INHERIT *relationship*. Since the SUPER_CLASS has many cardinality, it supports multiple inheritance.
- Aggregation is handled via the recursive application of CP relationship and PD relationship from CLASS via ATTRIBUTE through DOMAIN. An aggregated class has a set of attributes via CP relationship, which some of them are again defined by DOMAIN class via PD relationship. This process can be recursively applied until each attribute is broken down to simple attributes.

The relational representation of meta OO model is shown below in Figure 8.

```
CLASS(CNAME)
SUPER(CNAME, SUPER_CNAME)
PROPERTY(ATTRIBUTE_NAME, CNAME, DATATYPE,
    SINGLE/SET)
    /* Variations of PROPERTY are: */

    /* Handles recursive relationships between CNAME */
    PROPERTY(SUBROLE/SUPERROLE, CNAME,
        CNAME, SINGLE/SET)

    /*Handles relationships REL_NAME between CNAME
        and OTHER_CLASS */
    PROPERTY(REL_NAME, CNAME, OTHER_CLASS,
        SINGLE/SET)

    /* Handles aggregation with an INVERSE clause below*/
    PROPERTY(HAS_PARTS, CNAME, OTHER_CLASS,
        SINGLE/SET)

INVERSE(ATTRIBUTE, CNAME, OTHER_CLASS,
    OTHER_ATTRIBUTE)
    /* Variation of INVERSE is:*/

    /* Handles aggregation with a PROPERTY clause */
    INVERSE(HAS_PARTS, CNAME, OTHER_CLASS,
        IS_A_PART_OF)
```

Figure 7. Relational Schema for meta OO model

2.3 Translation Rules from EER Model to OO Model

The representation of relationships is one of the most fundamental ways in which data models differ. As there are several translation techniques in converting an ERD into a relational model structure depending on how relationships are handled [Song 1992], there could be at least two translation paradigms in translating an ERD into an OODB schema. The first approach is to convert each entity and each relationship into one object. The relationship object is called a link object in Rumbaugh et al. [1991]. Thus, the resulting schema will keep the structure of the original ER model. We will call this approach *stable translation,* as we did for the relational model [Song 1992]. There are two approaches using this idea [Navathe+ 1988, Ku+ 1991], and both are called the OOER model. Another approach is to combine certain relationships (1:1 and 1:N) into object classes using a pair of a reference attribute and its inverse attribute. For example, if there is a WORK_ON relationship between EMP and DEPT with many to one cardinality, then the EMP object has a reference attribute DEPT and the DEPT object has an inverse attribute EMP as a reference-set attribute (see Rule 2.2 below). These inverse relationships are necessary for the maintenance of relationship integrity and for two ways of accessing related objects. However, many-to-many and ternary relationships are created as separate object classes, called *link object classes.* And thus in this approach the original structure of the ER model is not kept in the target OODB schema. We call this approach *mapped translation,* as the original structure of the ERD is mapped away. This method is more complex than stable translation, but will result in a more compact OODB schema.

The stable translation method is advantageous since it can be easily converted from the EER diagram and the structure of the original ER model remains within the target model such as the relational, network model, and OO model. Also the maintenance of OODB schema in this structure is easier than in other structures because of its explicit relationship representation. However, this does not allow us to take full advantage of OO concepts where composite objects are allowed. This also results in a proliferation of objects as compared with other methodologies for OODB schema design. Whereas the stable translation can still explicitly show the ER's cardinality constraints in the OO diagram, the mapped translation only implicitly shows the cardinality by single-valued and set-valued attributes, unless otherwise defined by integrity constraints. Another advantage of mapped translation is that it can enforce one property of encapsulation by incorporating all the relevant concepts under one definition. However, it violates another property of encapsulation in that whenever a reference attribute in one class is modified, there is a need to modify its associated inverse attribute defined in another class. As shown in Rumbaugh et al. [1991], however, the link object in a stable translation schema can be physically implemented as a pointer at the implementation stage.

In KERO, the mapped translation approach was used instead of a stable one since the mapped translation takes advantage of the OODB quality of encapsulation using the inverse clause and can be directly implemented by most commercial OODBMSs. The shear number of classes is also reduced, and the connections between two objects are more direct without accessing the third object to link them (except in many-to-many relationships).

Converting Rules from ER model to object-oriented schema

(1) Entity type:
- Each entity type becomes an object class.

(2) Binary relationships (from entity type A to another entity type B)

(2.1) 1:1: Object class A has a reference attribute B, and object class B has a reference attribute A.

(2.2) 1:N: Object class A has a reference-set attribute B and object class B has a reference attribute A.

(2.3) M:N: Create a link object class L for the M:N relationship. Add a reference attribute A and B within the link object class L. The non-key attributes of the relationship then become properties of L. Object class A and object class B have a reference-set attribute of type L, respectively.

(3) "isa" relationship between A isa B
Declare as super class B in object class A.

(4) Attributes

(4.1) Single-valued attributes: They become simple attributes of the object class with atomic data types.

(4.2) Multi-valued attributes: If the order of values is important, they become list attributes. Otherwise they become set-valued attributes.

(4.3) Derived attributes: They become derived attributes whose values are computed by operations.

(4.4) Composite attributes: They become a reference attribute whose domain is an object class which consists of attributes making up the composite attribute in ERD.

(4.5) Attribute of 1:1 relationship: The attribute can be an attribute of a more permanent object (e.g., Department is more permanent than manager).

(4.6) Attribute of 1:N relationship: The attribute becomes an attribute of the N-side object.

(4.7) Attribute of M:N relationship: The attribute becomes an attribute of the link object class created in Rule 2.3.

(5) Weak entity type

Weak entity type becomes a set-valued attribute of the owner object, where the domain of the set-valued attribute is the weak entity type.

(6) Ternary relationship

Create an object class corresponding to the ternary relationship.

(7) Recursive relationship with entity type A with roles of Sub-Role and Super-Role.

(7.1) 1:1: Object class A has a reference attribute Sub-Role and Super-Role.

(7.2) 1:N (Super-Role: Sub-Role): Object class A has a reference-set attribute Sub-Role and a reference attribute Super-Role.

(7.3) N:1 (Super-Role: Sub-Role): Object class A has a reference-set attribute Super-Role and a reference attribute Sub-Role.

(7.4) M:N: Create a link object class L for the recursive relationship. Add a reference attribute Super-Role and Sub-Role within the link object class L. The non-key attributes of the relationship then become properties of L. Object class A have reference-set attributes Super-Role and Sub-Role of type L, respectively.

(8) Aggregation in ER model

(8.1) Create an aggregated object class G from the relationship enclosed in the aggregation entity.

(8.2) Add two participating entity types within G as a reference attribute.

(8.3) Add non-key attribute of the relationship as simple attributes of G.

(8.4) Each participating entity type has either reference attribute or reference-set attribute of type G, depending on the cardinality of relationship between them as in Rules (2.1), (2.2), and (2.3).

This approach is similar to Hughes [1991], Cattell [1991]. However, our approach is different from them by the way we handle many-to-many relationships and aggregation. Cattell does not show how to deal with aggregation; Hughes explain the ideas of the aggregation, but does not show explicit rules for the conversion to OO schema. Cattell and Hughes do not properly handle many-to-many relationships with non-key attributes. Cattell and Hughes represent a many-to-many relationship between A and B using the following rule, instead of our Rule 2.3 above:

Object class A has a reference-set attribute B, and
object class B has a reference-set attribute A.

In this case, if the relationship does not have its own attribute (non-key attribute in ER model, such as HOURS attribute in WORKS_FOR relationship in Figure 11), then this method is usable. However, if it does have its own attribute, then the attribute becomes awkward to handle. Furthermore, if the relationship has some constraints, then these may be specified redundantly and distributed to participating objects rather than encapsulating them into a single object class [Rumbaugh 1988]. Many real-world applications have many-to-many relationships with non-key attributes. Note that this problem can be easily handled in a one-to-many relationship, since any attribute or constraint imposed on the one-to-many relationship can be imposed on many-side object class.

We illustrate a simple example in this section. The ERD with an aggregation is shown in Figure 8 and derived OO schema is shown in Figure 9.

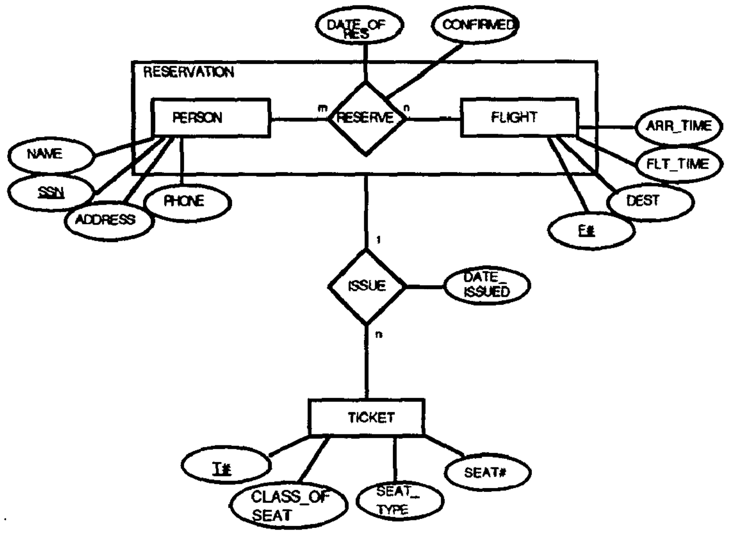

Figure 8 ERD with Aggregation

```
class Reservation
        Properties
                date_of_res: Date;
                confirmed: Boolean;
                issues: set(Ticket)
                        inverse is Ticket.issued_by;
                made_by: Person
                        inverse is Person.reserves;
                made_for: Flight
                        inverse is Flight.reserved_by;
end.

class Person
        Properties
                name, address, phone, SSN: String;
                reserves: Set(Reservation)
                        inverse is Reservation.made_by;
end.

class Flight
        Properties
                flight#: Integer;
                destination: CityCode;
                flight_time, arrival_time: Time;
                reserved_by: Set (Reservation)
                        inverse is Reservation.made_for;
end.
```

```
class Ticket

        Properties
                ticket#, seat#: Integer;
                class_of_seat: (Business, Fst_class,
Economic);
                seat_type: (Window, Aisle, Middle);
                date_issued: Date;
                issued_by: Reservation
                        inverse is Reservation.issues;
end.
```

Figure 9. Derived OO Schema from Figure 8

None of the literature we surveyed explicitly spelled out the translation rules as we have done in this paper. A more complete example is illustrated in Section 3.

The process of converting an ERD into an OO schema is not deterministic. For example, relationships in an ERD can be mapped into either an attribute, an operation, or an object class. Relationships with non-key attributes can become a separate object class. In addition, handling participation constraints, ternary relationships, and other constraints in OO databases are not clearly analyzed yet.

2.4 Architecture and Implementation of KERO

KERO is classified as a *design* expert system. The design system needs iteration for refinements until certain conditions are satisfied or all the rules have been exhaustively applied and no more changes to the working memory can be made. These iterations are typically implemented as a forward chaining system. The overall flow diagram of KERO is shown in Figure 10.
We have implemented KERO using MIKE, which is a meta-interpreter of Prolog based on a public domain Prolog. KERO has a total of 58 rules in MIKE syntax. MIKE was quite slow because of the nature of meta-interpreter and the looping of the forward chaining. Currently, we are reimplementing KERO using KnowledgePro for Windows for a faster performance and better user interface.

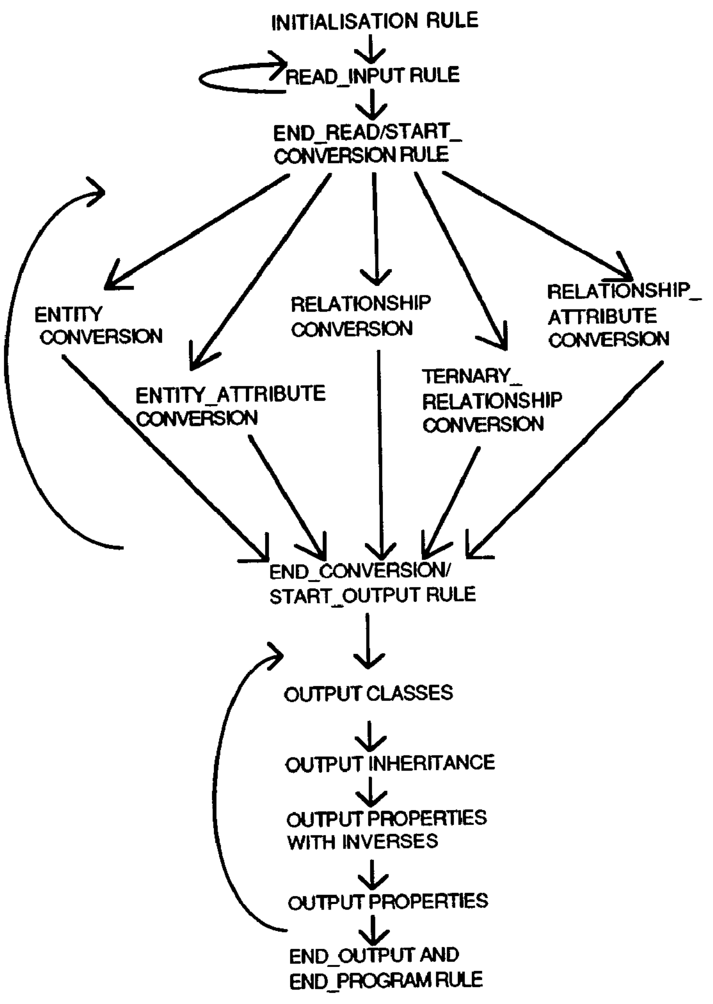
INITIALISATION RULE
READ_INPUT RULE
END_READ/START_CONVERSION RULE
ENTITY CONVERSION
RELATIONSHIP CONVERSION
RELATIONSHIP_ATTRIBUTE CONVERSION
ENTITY_ATTRIBUTE CONVERSION
TERNARY_RELATIONSHIP CONVERSION
END_CONVERSION/START_OUTPUT RULE
OUTPUT CLASSES
OUTPUT INHERITANCE
OUTPUT PROPERTIES WITH INVERSES
OUTPUT PROPERTIES
END_OUTPUT AND END_PROGRAM RULE

Figure 10 Flow Diagram of KERO System

2.5 Enhancing Converted OO Schema

We think that the OODB schema converted from the ERD can serve as a conceptual model which is a result of a conceptual *analysis*. This analysis can be further semantically enhanced for the *design* as discussed in this section. By design, we mean to add more implementation-oriented aspects to the conceptual model to create a logical model, which is ready to be implemented.

(1) Abstract class

The notion of entity in ER model is not exactly the same as the class in OO model. Every entity in the ER model can be a class in OO model, but not every class in OO model will be an entity in ER model. An entity in ER modeling is a real-world object which will store meaningful data. Thus, an entity type without any significant attributes in the ER model may exist at the analysis stage, but can be deleted at the logical design stage. However in the OO model, a class even without any significant attributes may exist to support the inheritance of operations. This kind of class is called the abstract class. The abstract class is a class which is declared as a class, but which will not have any instantiated objects. C++ supports the abstract class via the pure virtual function.

(2) Type class and object class

Type class is a data type used as a domain of an attribute, while an object class is a meaningful class modeled for the semantics of application. Even though both of them can be implemented as classes in OO system, they are semantically different. For example, composite attributes ADDRESS (COUNTRY, CITY, NUMBER, STREET and NAME (LAST, FIRST) could be a class type in OO system, but they are not very meaningful in OODB unless they are related to meaningful classes such as person and employee. Programming convenience may introduce this kind of class in OO system.

(3) Class attribute (method)

A class attribute is one which has only one value for all the objects of a class. They typically represents a summary or a collective value for a class (such as the total number of instantiated objects in the class). The ER model does not capture the semantics of the class attribute (and, of course, class method, also). So identify them and add to the OO model.

(4) Constraint modeling

We can impose some constraints on the OODB schema depending on the attribute type and relationship as follows:

- If an attribute has an enumerated data type, then that can be specified as a constraint.
- If an attribute has a range of values, then that can be specified as a constraint.

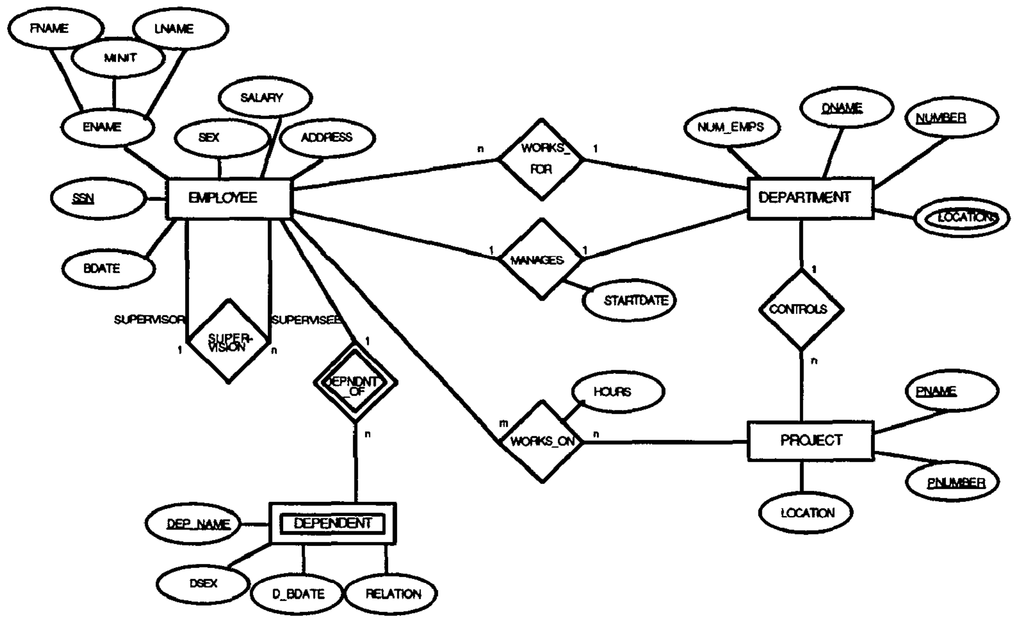

Figure 11. Company ER Model (taken from Elmasri & Navathe1989)

- For each set-valued attribute or reference-set attribute
  - if they require a sequence, declare as a list data type
  - if it allows a redundant value, declare as a bag data type
- For each relationship in ER model
  - if it has a specific maximum cardinality such as 1:6, then specify as a constraint
  - if it needs to specify the participation constraint, then specify as a constraint methods and message introduction (they are application specific)

(5) Operations

Attribute methods can be automatically generated. Typical operations such as constructor, destructor, accessor operations can be generated as a form of attribute methods. However, operations are application-specific, and thus other methodologies must be used to generate a more complete list of operations, such as transformer and interface operations with external systems.

3. Example

In this section, we illustrate our methodology using an example taken from Elmasri and Navathe [1989]. The ERD is shown in Figure 11 and output OODB schema is shown in Section 3.2.

3.1 Input ER Model

3.2 Output OODB Schema

```
class(works_on)
  properties
    participant: employee
      inverse of employee.participates;
    object: project
      inverse of project.done_by;
    hours: integer;
end.
```

```
class(employee)
  properties
    manages: department
      inverse of department.manages;
    works_for: department
      inverse of department.works_for;
    has_property: ename
      inverse of ename.is_property_of;
    participates: set(works_on)
      inverse of works_on.participant;
    has_parts: set(dependent)
      inverse of dependent.is_part_of;
    supervisor: employee;
    ssn: string;
    bdate: date;
    sex: m/f;
    address: string;
    salary: integer;
    supervisee: set(employee);
end.
```

```
class(department)
  properties
    manages: employee
      inverse of employee.manages;
    works_for: set(employee)
      inverse of employee.works_for;
    controls: set(project)
      inverse of project.controls;
    startdate: date;
    dname: string;
    number: integer;
    num_emps: integer;
    locations: set(string);
end.
```

```
class(project)
  properties
    controls: department
      inverse of department.controls;
    done_by: set(works_on)
      inverse of works_on.object;
    pname: string;
    pnumber: integer;
    location: string;
end.

class(dependent)
  properties
    is_part_of: employee
      inverse of employee.has_parts;
    dep_name: string;
    dsex: m/f;
    relation: string;
end.

class(ename)
  properties
    is_property_of: employee
      inverse of employee.has_property;
    fname: string;
    minit: character;
    lname: string;
end.
```

4. Summary and Conclusion

Our research focuses on advancing methodologies for object-oriented database (OODB) schema design. While OO database systems themselves have been widely researched and regarded as next generation database systems, there is no well accepted methodology for OODB schema design. We think that as a short term solution the translation methodologies from semantic data models are important for the following main reason: Most existing relational database design methodologies are based on the semantic approach. By utilizing existing methodologies for OODB design, we will not lose the valuable knowledge of many information engineers, obtained throughout the design and use of relational database systems. So the transition from existing systems to OODB systems can be smooth once a set of guidelines is provided, as done in this paper. The complexities of the OODB design process make an automated design assistance extremely desirable, as well. For the long term, we need to develop direct object modeling approaches, for both structure and behavior, which may prove to be useful independent of database application domains.

Based on these, we have presented the design and implementation of KERO, a knowledge based system that converts an ER model into a OODB schema. Our translation scheme, called mapped translation, can convert various semantic constructs of the ER model such as entity, relationship, ternary, recursive, weak entity, generalization/specialization, inheritance, and aggregation. We have spelled out the step-by-step transformation rules for each construct of the ER model. We have presented in detail how to properly convert many-to-many relationships with non-key attributes and discussed the translation of the aggregation in ER model into OODB schema with examples. We have also suggested ideas about how the converted schema may be further semantically enhanced. The output from KERO can be directly implemented in most commercial OODBMSs. We will add to KERO increased interaction with designers to facilitate customized OODB output. The KERO system has been implemented in MIKE, a public domain meta interpreter of Prolog.

References

Batini, C., Ceri, S. & Navathe, S.B. (1991). *Conceptual Database Design: An Entity-Relationship Approach.* The Benjamin Cummings.

Cattell, R.G.G. (1991). *Object Data Management: Object-Oriented and Extended Relational Database Systems*, Addison-Wesley, Reading, MA.

Chen, P.P. (1976). "The Entity Relationship Model - Toward a Unified View of Data," *ACM TODS*, 1:1, pp. 9-36.

Eisenstadt, M. and Brayshaw, M. (1990). "A Knowledge Engineering Toolkit," *BYTE*, October 1990, pp. 268-282.

Elmasri, R. and Navathe, S.B. (1989). *Fundamentals of Database Systems*, The Benjamin/Cummings.

Gorman, K. and Choobineh, J. (1991). "An Overview of of the Object-Oriented Entity-Relationship Model (OOERM), in *Proc. of the Twenty-Third Annual Hawaii Int'l Conf. on System Sciences*, Hawaii.

Hughes, J.G. (1991). *Object-Oriented Databases*. Prentice Hall.

Kim, W. (1990). "Object-Oriented Databases: Definition and Research Directions," *IEEE Transaction on Knowledge and Data Engineering*. 2(3), pp. 327-341.

Ku, C.S., Youn, C. & Kim, H.-J. (1991). "An Object-Oriented Entity-Relationship Model," 1991 ISMM International Conference on *Computer Applications in Design, Simulation and Analysis*," Las Vegas, Nevada, March 19-21, 1991, PP. 55-58.

Nachouki, J., Chastang, M.P. and Briand, H. (1991). "From Entity-Relationship Diagram to An Object Oriented Database," in *Proc. of 10th Int'l Conf. on Entity-Relationship Approach*, pp. 459-481.

Navathe, S.B. & Pillalamarri (1988). "OOER: Toward Making the E-R Approach Object-Oriented," in *Proc. of 8th Int'l Conf. on Entity-Relationship Approach*, pp.55-76.

Rumbaugh, J. , Blaha, M., et al. (1991). *Object-Oriented Modeling and Design*, Prentice-Hall.

Song, I.-Y. (1992). "Translation Techniques from ERD to Relational Model," Submitted for publication.

Song, I.-Y. (1992b). "A Survey of Object-Oriented Database Design Methodologies," in Proc. of *First Int'l Conf. on Information and Knowledge Management*, Baltimore, MD, Nov. 9-11, 1992, pp. 52-59.

Song, I.-Y. (1993). "A Knowledge Based Object-Oriented Database Schema Generator," To appear in Proc. of *1993 IEEE/ACM Int'l Conf. on Developing & Managing Intelligent System Projects*," Washington, D.C, March 29-31, 1993.

Stonebraker, M., Rowe, L.A., etc. (1990). Third-Generation Database System Manifesto," *SIGMOD Record*, 19(3), pp.31-44.

Teorey, T.J., Yang, D., and Fry, J.P. (1986). "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model," *Computing Surveys*, 18:12, June, pp. 197-222.