December 1993

Further Evolution of Semantic Data Models

Hock Chan
National University of Singapure

FURTHER EVOLUTION OF SEMANTIC DATA MODELS

HOCK C. CHAN, Department of Information Systems and Computer Science
National University of Singapore, Singapore

ABSTRACT

Semantic data models for modeling application domains are continually evolving and acquiring better modeling features, which lead to better management and usage of database systems. The present stage of evolution of two major model groups: the entity relationship (ER) models and the object oriented (OO) models are examined. ER models have evolved mainly by adding more semantic concepts, with less emphasis on operations and implementational support. Consequently, while ER models are widely used for modeling purposes, there are very few full-fledged ER systems. On the other hand, evolution of OO models is less towards addition of semantics and more towards implementation. Class hierarchy with inheritance is well-implemented, and method definition greatly enhances the modeling power. Hence there are increasingly more OO databases available commercially. The next step in the evolution is to bring the two major models closer, by integrating OO's implementation superiority with ER's conceptual superiority.

INTRODUCTION

While the traditional hierarchical, network and relational data model have set database systems firmly as important components of information systems, they do not have enough semantic content to model application domains easily. Database designers and users have struggled to model and operate application domains using the simple constructs available in these data models. Many semantic models are proposed, including functional model, NIAM, enhanced relational models, ER model and OO model. Presently, the two major models groups are ER models and OO models.

The semantic models have yet to achieve their potential in the design and operation of information systems. While many are used for design purposes, few are used for operational data management. The main reason is that advances in semantic structures are not matched by supportive advances in operations and implementations. Further balanced evolution of semantic models, combining the best features of ER and OO models, is proposed to help realize the potential.

The ER model, since its initial proposal in 1976 (Chen, 1976), has evolved in many ways, acquiring new semantic concepts such as generalization and specialization (Smith and Smith, 1977b), aggregation (Smith and Smith, 1977a), category (Elmasri et al., 1985, Elmasri and Navathe 1989), relationships among relationships (Thalheim 1991), and nested ER concepts (Carlson et al. 1990). With these additions, ER models provide very good semantic concepts for direct modeling of application domains. They are widely used for conceptual designs (Elmasri and Navathe 1989, Batini et al., 1992, Teorey, 1990).

OO models have evolved mainly from object-oriented programming. There is as yet no generally agreed single model. However, it is quite easy to identify the core features, which include objects, classes, class hierarchy, inheritance, polymorphism, methods and encapsulation.

There are three components of a model: structures, operations and integrities (Date 1990). The next few sections present these components at the present stage of evolution. Section 2 presents in detail the main structural features of ER models and OO models, section 3 the integrities and section 4 the operations. Section 5 shows how ER models and OO models can be combined to evolve a better model. The conclusion of the paper is in section 6.

STRUCTURAL COMPONENTS

Structural Components of ER Models

The most important structural component of ER models is the entity. An entity instance represents a physical or non-physical thing in the world, such as a building or a legal document. An entity instance has one or more attributes. For example, a building has a floor area, an address, and a year of completion. Entity instances can be grouped into entity types. An entity type will have instances that have the same properties. It may have one or more key attributes to uniquely identify the instances. Some ER models do not insist on key attributes. System surrogates are used for unique identification, serving the same function as objects identifiers in OO models.

The next important structure component is relationship. Relationships exist among entities. A relationship type must involve two or more entity types, not necessarily distinct. Role names are compulsory where an entity type has more than one role, so that participations can be distinguished. For example, if type E has two roles in the relationship R, then the roles must be specified to state unambiguously the exact relationship between two instances of E in R.

For most ER models, both entities and relationships can have attributes. It is noted that some ER models do not allow attributes for relationships. Usually an entity will have at least one attribute. Some models do not insist on this, but it is hard to imagine the use of entities without attributes. Relationships on the other hand often do not have any attributes. In that case, they serve merely to connect the entities. For a relationship, none of its attributes will form the key. Normally, it's key is composed of the participating entities keys. Alternatively, system surrogates can be used to avoid compound keys, which are harder to process.

Most ER models allow the specification of constraints on the number of times that an entity instance can, or must, participate in a relationship. These constraints are known as cardinalities. The usual constraints are 1:1, 1:m and n:m, indicating a one-to-one, one-to-many and many-to-many relationship. These indicate the maximum number of participation that an entity instance can have. A more advance version indicates the minimum number as well, which is the minimum number of times that each entity instance must participate in the relationship.

One of the earlier ER models (Chen, 1981, Chen, 1984) is the binary ER model that does not show a relationship diamond. An entity points to another entity directly without going through a relationship type. Although this direct relationship appears to have lost its popularity in ER models, it is still common in OO models. Sometimes, these are called entity or object models. In some situations, omission of relationships allows the model and the query to be more concise.

Attributes may be single valued or multi-valued, and may be simple or complex. Complex values are composed of other values. For example, address can be viewed as complex with sub-values for house number, building name, street name, postal code, city and country.

Most ER models, including the original, has the concept of weak entities. A weak entity type usually has no key attribute (Chen, 1976). Its identification and existence depends on a normal entity.

With more demand for direct modeling of the application domain structure, more components are added. These include is-a hierarchy, category, aggregation, relationship among relationships and ER clusters, which are described below.

An is-a hierarchy is a group of entities connected by supertype-subtype relationships. It can be produced from generalization or specialization processes (Teorey, 1990). Constraints on an is-a hierarchy includes *total* , where every supertype instance must appear in at least one subtype, *partial*, where some supertype instances need not appear in any of the subtypes, *overlap*, where some instances belong to more than one subtypes and *disjotnt*, where no instances can appear in two or more subtypes. The essential property of an is-a hierarchy is the inheritance of attributes and relationships of the supertypes by the subtypes. It is now common to find multiple inheritances, i.e. an entity is subtype to two or more entity types directly. Clashes can happen in inheriting from two sources. Resolution can be done by naming constraints, which avoid the clashes, priority setting for the system to automatically resolve ambiguities, or explicit declaration to resolve ambiguities.

Some situations require a entity type whose instances each belongs to one and only one of two or more other entity types. The relationship between this entity type and the other types is called the category (Elmasri et al., 1985, Elmasri and Navathe, 1989). For example, an vehicle owner is either a person or a corporation. Inheritance is selectively done from only one supertype depending on the particular instance.

The concept of aggregation is introduced so that two entities and their relationship can form an aggregation, which can itself be considered an entity (Korth and Siberschatz, 1986, Smith and Smith, 1976a, 1976b). It can form relationships with other entities. Some ER models allow relationships to form relationship, without the need to first form an aggregation (Thalheim 1991). Now that there are relationships between relationship and relationship, and between entity and entity, the missing relationship is one between entity and relationship.

Some ER models include the concept of entity (and relationship) clusters, or nested entities and relationships (Teorey, 1990, Teorey et al., 1989, Carlson et al., 1990, Feldman and Miller, 1986). Part of an ER diagram can be replaced by an entity (or a relationship). This simplifies ER diagrams and leads to the concept of layered-ER diagrams, similar to the layered data flow diagrams. This is different from aggregation in that the cluster itself does not form new relationships. Any relationships can be traced to some entity within that cluster. Aggregation, as noted above, can form new relationships.

Structural Components of OO models

The major structural components can be identified from the numerous OO models (Bertino et al., 1991, Catell, 1991, Deux et al., 1991, Lamb, 1991, Fishman et al., 1990, Shaw and Zdonik, 1989, Bancilhon et al., 1989, Wand, 1988). The most important is object, which has its own identity regardless of its attributes. An object can be used to represent any concept in ER models, whether attribute, entity, relationship, or cluster.

Objects have properties. An object without property is of little use. The value of a property is another object, such as an integer 7, a list [1, 2, 3] or any object including itself. In ER terms, properties can represent attributes of an object entity. They also represent relationships (Bertino et al. 1991, Catell 1991, Shaw and Zdonik 1989). An example is a *car* object with a property *dept* that points to a department object. This property represents the department owning the car.

OO systems may classify properties into *private*, *read* or *public* (Deux et al. 1991, Lamb et al. 1991). *Private* properties are not accessible by other objects. *Read* properties can be read by other objects. *Public* properties can be read and updated by other objects. Not all classifications are available in all OO systems. A very important feature - *inverse* properties - can be found in some OO systems (Lamb, Catell). It helps to maintain the integrity of the database, although only for binary relationships. Let us assume the following class definition, using the ObjectStore syntax:

```
class student
{ public
        professor* supervisor
                inverse_member professor::supervisee;
 ...}
class professor
{ public
        student* supervisee
                inverse_member student::supervisor;
 ...}
```

When an object of *student* is assigned to an object of *professor* as the value of the *supervisee* property, the later object is automatically assigned to the earlier object as the value of the *supervisor* property. Similarly, a deassignment in either object will lead to automatic deassignment in the other object..

Objects can have method which, unlike properties that simply access some variables, can perform general computations. A method have two parts: a signature and an implementation. The signature shows the method's parameters and the type of result returned by the method. For example, consider this signature

$M(O_1, O_2, ..., O_n):O_x$

where M is the name of the method which needs n parameters P_1 to P_n, each of which is of the n stated object types (O_1 to O_n), and the method returns an object of type O_x. The implementation consists of program codes to execute the method. The codes are internal to the object. Only the signature is required in order to invoke the method, e.g. method M in object Y can be invoked with

$Y\ M(O_1, O_2, ..., O_n)$,

depending on the particular syntax of the system.

Another important structural component is object class. An object class defines the structure (properties, methods, superclasses) of a class of objects. All objects of this class will have this structure. Some models distinguish between class and type, where type is the structure and class is the set of all instances of that type. There are class attributes that belong to the class, and not to the instances of that class. These may contain summary attributes,, such as the total number of instances, or attributes with values applicable to all instances, e.g. sex with value male for the class of male employees.

The object classes can form superclass-subclass relationships. A subclass inherits all the properties and methods of all the superclasses. If the inheritance from different classes clash, some mechanism is required to resolve the conflict. A simple way is to get the user to explicitly define new properties or methods during the class definition (Lamb 1991). More complex systems may allow the conflict and try various resolution methods during run-time.

In OO programming, inherited properties and methods can be modified or changed totally. Some OO database systems are more restrictive, allowing only limited and compatible changes (Lamb 1991, Deux et al. 1991, Lecluse et al. 1988). For example, Deux et al. 1991 requires the preservation of semantics, i.e. objects in the signature can be redefined to be subtypes of the original objects.

Encapsulation implies that an object can only be accessed, both read and write, via methods. However, it was found through experience that this will be very restrictive. Hence the need for three types of properties: private, read and public. Without these, every property to be accessed will require a method. For example, a building object with an address property will need two

methods, one for reading and one for. Many methods will exist just to handle the simple read and write of properties. Some systems, e.g. Bancilhon et al. 1989, allow violation of encapsulation only during ad hoc queries, but not within the normal programs or methods.

OO models include many data structures commonly used in programs. The three commonly used for database modeling are: set, list and tuple. Set and list have their usual meanings and tuple has the same meaning as in the relational model. These structures are used to hold a collection of objects. They are used to model one-to-many, many-to-many relationships and multi-valued attributes.

INTEGRITY COMPONENTS

Integrity Component of ER model

Er ER model possess the following implicit integrity rules:
1. a relationship cannot exist without the participating entities.
2. a instance of a subclass implies its existence in all its superclasses.
3. the constraints *total, partial, overlap* and *disjoint* apply to instances of the is-a hierarchy
4. participation in relationships are constrained by the cardinalities.
5. an instance of a category implies its existence in one and only one of its superclasses.

Integrity Component of OO models

OO models have these integrities:
1. a subclass's existence by definition requires its superclasses to exist.
2. the inverse feature implies that the double-sided reference is maintained.
3. a property cannot have a non-existent object.

OPERATION COMPONENTS

Operation Component of ER models

The format operations for ER models are defined by the numerous ER algebras, calculi, and query languages (Shoshani, 1978, Poonen, 1979, Atzeni and Chen, 1981, Markowitz and Raz, 1983, Chen, 1984, Elmasri and Larson, 1985, Parent and Spaccapietra, 1986, Hohenstein and Gogolla, 1988, Chan, 1991, Siau et al., 1991). The operations determine what information can be inserted and retrieved. The basic operations include retrieval of an entity instance, retrieval of related instances, and the attribute values. With each additional structure component, new operations are needed. For example, the inclusion of is-a hierarchy requires the operation of defining the is-a structure as well as modification to the retrieval and update operations to incorporate inheritance. While many structure components with new semantics have been added to ER models, the operations to maximize their benefits have not been fully studied.

Operation Component of OO models

There are also numerous algebra, calculi and query languages proposed for OO models, including integrations with general programming languages such as C, C++ or Smalltalk (Shaw and Zdonik 1989, Bancilhon et al. 1989, Bertino et al. 1991, Rundensteiner and Bic 1992). In addition to the predefined operations, methods in OO models allow the definition of arbitrarily complex operations on any number of classes and instances.

FURTHER EVOLUTION

In this section, the major similarities and differences of ER and OO models are examined. This highlights the features the can be transplanted from ER to OO, or from OO to ER, in order to obtain the combined benefits of both models. It is obvious that ER models have many more structural features than OO models. OO models use a single concept object to represent many different things. This is similar to the relational model that uses a single concept relation to represent many different concepts. OO models can support the various ER concepts, but require extra design and programming effort.

The major common structural component is ER's entity and OO's object. An entity is very similar to an object. For example, a entity cat in a CAT entity type can be a cat object belonging to a CAT object class. The is-a hierarchy in ER models mirrors the class hierarchy in OO models. The minor differences between objects and entities include: objects but not entities contain variables pointing to related objects, objects have programs while entities do not, and inheritance in OO's can be redefined, but not inheritance in ER.

Beside entities, is-a hierarchy and inheritance, OO models lack most of the other structural features in ER models, such as weak entities, relationships, category, aggregation and clustering. Although these features can be implemented, the operations need to be tediously defined to maintain the implicit ER integrities. Consider weak entities. A strong object can point to a weak object, e.g. through a property. Deleting the first object will not automatically lead to the deletion of the other objects, and this violates the integrity rules on weak entities.

The most serious deficiency of OO models is the lack of the structural feature of relationships. A ternary relationship is modeled as an object with pointers to the participating entities and it's attributes (Catell 1991). A binary relationship with attributes has to be modeled as a ternary relationship. Since OO models do not have as many structural concepts as ER models, their implicit integrity rules are also fewer. The implication is that database designers have to program many more integrities into the classes and methods.

There are many similarities in the operational components of ER and OO models. This is a bit unexpected, given that ER has many more structural features. The reason is that the operations have not been developed for these new features. The minor operational differences are significant. There are two very basic operations in ER models: testing attribute values and testing relationship connections. Conditions on attributes are very similar, if not identical, for ER models and OO models. Most ER languages allow a condition such as *part.color='blue'*, and so do many OO languages.

Conditions involving relationships are quite different. Compare a common notation for ER:

where supplier supply part and part.name='filter'

with OO's

s.supplies.part_of.name includes 'filter',

where *s* is an object of class supplier. There are two significant differences. ER uses the names of the entity and relationship types directly. OO uses property names in the classes. Second, OO languages require the distinction between single values and set / list values. In OO, the value of s.supplies is a set, and requires a set operation such as *member-of* or *include*, and not a single value operation such as '='.

These differences carry forward into more complex conditions, such as negation and statistics such as sum, average, max, min and count, or exists. ER allows relationship conditions to be chained (Shoshani, 1978) such as

employee.work.department.serve.client,

where *employee, department* and *client* are entity types and *work* and *serve* are relationship types. OO also allows chaining (Bertino et al., 1992), but based on the method/property names, such as *emp.work_for.department_of.serving.served_by* where we assume that the relationships are ternary, and therefore represented by intermediate objects between the two entity objects. Again, the user has to worry about the data structure: is it a single value, a set or something else?

Methods in OO models give a definite advantage. Derived attributes and derived relationships can be easily defined as methods. In ER models, these have to be done separately from the model, either as views or in other programs.

CONCLUSION

The next step in the evolution of semantic data models is to combine and improve on the better features of the two major model groups - ER and OO models. The main obstacles in the evolution of ER models are operational clarity and implementational support. The main obstacle in OO models is lack of semantic features to directly model the application domains. Lack of semantic features leads to lack of operations to directly reflect the operations in application domains. On the other hand, OO systems are very powerful since they are coupled with powerful programming languages such as C++.

One way to combine ER and OO models is to design and implement object classes that behave in the same ways as semantic features in ER models. Their behaviors should incorporate the implicit integrities found in ER models. For example, weak objects should self-destruct if the strong objects that they are attached to are deleted, and relationship should also self-destruct if any of the participating entities are deleted. Such an implementation will realize the potential of ER models, and enhance the semantic power of OO models. Ultimately, the evolution will lead to models that can directly model the structures, and more importantly, the operations and integrities of application domains. This will allow more effective management by database administrators. It will also increase the productivity of database designers and users.

REFERENCES

Atzeni, P. and P.P. Chen, "Completeness of Query languages for the Entity Relationship Model" pp109-122, Entity-Relationship Approach to Information Modeling and Analysis, ed. P.P. Chen, North-Holland, 1981.

Bancilhon, F., Cluet, S and C. Delobel, "A Query Language for the O_2 Object-Oriented Database System", in Proceedings of the 2nd Int. Workshop on Database Programming Languages, June 1989.

Batini, C., Ceri, S., and S.B. Navathe, Conceptual Database Design, An Entity Relationship Approach, The Benjamin/Cummings Publishing Company Inc., USA, 1992.

Bertino, E. et al., "Object-Oriented Query Languages: The Notion and the Issues", IEEE Tran. on Knowledge and Data Engineering, Vol. 4, No. 3, June 1991, p 223-237.

Carlson, C.R., Ji, W. and Arora, A.K., "The Nested Entity-Relationship Model - A Pragmatic Approach to E-R Comprehension and Design Layout", Proceedings of the Ninth International Conference on Entity-Relationship Approach, 1990, pp203-217.

Catell, R.G.G., Object Data Management, Object-Oriented and Extended Relational Database Systems, Addison-Wesley, USA, 1991.

Chan H.C., "An Entity-Relationship Enhanced Logic System", The Second International Symposium on Database Systems for Advanced Applications, Tokyo,p401-410, April 1991.

Chen, P.P., "The Entity-Relationship Model: Toward a Unified View of Data", ACM Transactions on Database Systems, Vol. 1, No. 1, 1976, pp 166-192.

Chen, P.P., "A Preliminary Framework for Entity-Relationship Models", ER Approach to Information modeling and analysis, ed. P.P. chen, North-Holland, 1981.

Chen, P.P., "An Algebra for a Directional Binary Entity-Relationship Model", First International Conference on Data Engineering, pp37-41, 1984.

Date, C.J., An Introduction to Database Systems, Volume 1, Fifth Edition, Addison-Wesley, USA, 1990.

Deux, O. et al., "The O_2 System", Communications of the ACM, Vol. 34, No. 10, Oct 1991, p34-48.

Elmasri, R.A. and J.A. Larson, "A Graphical Query Facility for ER Databases", pp236-245, the 4th international conference on entity-relationship approach, IEEE computer society press, 1985.

Elmasri, R., Hevner, A. and Weeldreyer, J., "The Category Concept: An Extension to the Entity-Relationship Model", Data Knowledge Engineering, Vol. 1, No. 1, 1985, pp 75-116.

Elmasri, R. and Navathe, S.B., Fundamentals of Database Systems, Addison Wesley, 1989.

Feldman, P. and D. Miller, "Entity Model Clustering: Structuring a Data Model by Abstraction", The Computer Journal, Vol. 29, No. 4, 1986, pp348-360.

Fishman, D.H. et al. "Iris: An Object-Oriented Database Management System", in Readings in Object-Oriented Database Systems, S.B. Zdonik and D. Maier (ed), Morgan Kaufmann Publishers, Inc, USA, 1990, p216-236.

Hainaut, J.L., "Entity Relationship Models: formal specification and comparison", in Entity Relationship Approach, The core of conceptual modelling, (ed) H. Kangassalo, Elsevier Science Pub. (North-Holland), pp433-444, 1991.

Hohenstein, U. and M. Gogolla, "A Calculus for an Extended Entity-Relationship Model Incorporating Arbitrary Data Operations and Aggregate Functions", Proceedings of the Seventh International Conference on the Entity-Relationship Approach, 1988, pp129-148.

Korth, H.F. and A. Siberschatz, Database System Concepts, MacGraw-Hill, 1986.

Lamb, C., Landis, G., Orenstein, J. and D. Weinreb, "The ObjectStore Database System", Communications of the ACM, *October 1991*, Vol. 34, No. 10, pp50-77.

Lecluse, C., Richard, P. and F. Velez, "O2, an Object-Oriented Data Model", in Readings in Object Oriented Database Systems, M. Zdonik and D. Maier (eds), Morgan Kaufmann, 1990, pp227-236.

Markowitz, V.M. and Y. Raz, "ERROL : An Entity Relationship, Role-oriented, Query Language", pp329-345, Entity-Relationship Approach to Software Engineering, C.G.Davis, S.Jajodia, P.A.B.Ng and R.T.Yah (eds), North-Holland, 1983.

Parent, C. and S. Spaccapietra, "Enhancing the Operational Semantics of the Entity Relationship Model", Database Semantics (DS-1), T.B. Steel, Jr. and R. Meersman (ed), Elsevier Science Publishers, 1986, p159-173.

Poonen, G., "CLEAR: A Conceptual Language for Entities and Relationships", pp194-215, Centralized and Distributed Data Base Systems, ed. W.W. Chu and P.P. Chen, IEEE Computer Society, 1979.

Rundersteiner, E. A., and Bic, L., "Set Operations in Object-Based Data Models", IEEE Transactions on Knowledge and Data Engineering, Vol. 4 No.3, June 1992, pp382-398.

Shaw, G.M. and S. B. Zdonik, "An Object-Oriented Query Algebra", in Proceedings of the Second International Workshop on Database Programming Languages, June 4-8, 1989, Oregon, USA, p103-112.

Shoshani, A., CABLE: A Language Based on the Entity Relationship Model, Lawrence Berkeley Lab., Berkeley, CA, 1978.

Siau K.L., Chan H.C., Tan K.P., "Visual Knowledge Query Language as a Front-End to Relational Systems", The 15th Annual International Computer Software and Applications Conference, Tokyo, Japan, 11-13th Sep 1991, p373-378.

Smith, J.M. and Smith, D., "Data Base Abstractions: Aggregation", Comm. ACM, Vol. 20, No. 6, June 1977, pp 405-413.

Smith, J.M. and Smith, D., "Data Base Abstractions: Aggregation and Generalization", ACM Transaction on Database Systems, Vol. 2, No. 2, June 1977, pp 105-113.

Teorey, T.J., Database Modeling and Design, the entity relationship approach, Morgan Kaufmann Publishers, USA, 1990.

Teorey, T.J., Wei, G., Bolton, D.L., Koenig, J.A., "ER Model Clustering as an Aid for User Communication and Documentation in Database Design", Communications of the ACM, August 1989, Vol. 32, No. 8, pp975-987.

Thalheim, B., "Extending the Entity-Relationship Model for a high-level Theory-Based Database Design", in Next Generation Information System Technology, J.W. Schmidt and A.A. Stogny (Ed.), Springer-Verlag, Germany, 1991.

Wand, Y. "A Proposal for a Formal Model of Objects", in Object-Oriented Concepts, Databases and Applications, W. Kim and F.H. Lochovsky (eds), USA, 1988.