ELSEVIER

Information and Software Technology 39 (1997) 449–462

INFORMATION AND SOFTWARE TECHNOLOGY

An evaluation of extended entity–relationship model

Hossein Saiedian

Department of Computer Science, University of Nebraska at Omaha, Omaha, NE 68182-0500, USA

Received 29 May 1995; revised 11 December 1996; accepted 18 December 1996

Abstract

The Entity–Relationship (ER) model allows a database designer to develop a high-level conceptual schema without having to consider low-level issues such as efficiency, the underlying database management system model, or physical data structures. The ER model has become very popular for database design and is used quite extensively. In order to strengthen its expressive power, many database researchers have introduced or proposed certain extensions to this model. Some of these extensions are important, while others add little expressive power but provide auxiliary features. Since the ER model is used so widely, it is important to know what extensions have been proposed for this model and what these extensions offer to the users. The objective of this article is thus to survey major extensions to the ER model and to evaluate their merits. We point out that lying behind the syntactical differences of the various extensions is the enriched semantics about relationships among entities. We also point out the close relationship between ER modeling and object-oriented data modeling. © 1997 Elsevier Science B.V. © 1997 Elsevier Science B.V.

Keywords: Conceptual modeling; Extended entity–relationship model; Object model

1. Introduction

In 1976 Peter Chen published the original entity–relationship (ER) model which provided an easy to use graphic approach to logical database design [1]. The model is comprehensive, yet it avoids the complications of storage and efficiency considerations, which are reserved for physical database design. In the two decades since then, many others have adopted the original model and used it enthusiastically after minor changes. In addition, a number of authors have extended the model to enhance its capabilities so that it is more appropriate for their particular endeavor. The most comprehensive extension includes dynamic icons in an adaptation to object-oriented database modeling.

In the preface to his paper, Chen states, "The entity–relationship approach provides an easy to understand yet comprehensive methodology for logical database design independent of storage or efficiency considerations" [1]. The problem he solves with the ER model is the complexity of logical database design. The conventional process of database design is based upon mapping real-world information directly to a user schema, specific to a certain type of database management system (DBMS). The designer is constrained by limited data structure types, access path considerations and efficiency of retrieval and updates, yet must produce a user schema while considering all these issues. The result can be a user schema that is difficult to understand and change. The ER approach simplifies this process by introducing an intermediate design called an *enterprise* view or enterprise schema. The enterprise schema, expressed as an ER diagram, is a conceptual database design which is a pure representation of the real world and yet is independent of storage and efficiency considerations. This enterprise schema can later be translated into a DBMS specific user schema. This two-phase approach makes the design process simpler and better organized. The enterprise schema is easier to design, and, in case of transition from one type of DBMS to another, can be remapped to a user schema suited to the new DBMS.

The ER model is widely used during requirements analyses and for conceptual database modeling. Because of its simplicity, it is more easily understood by non-technical individuals. Tests in the real world environment have shown it to be an effective communications tool between database designers and end users.

1.1. Components of the entity–relationship model

The main components of the entity–relationship model are *entity types*, *relationship types*, and *attributes*. An entity is defined as a "thing" which can be uniquely identified. It

PII S0950-5849(97)00002-5

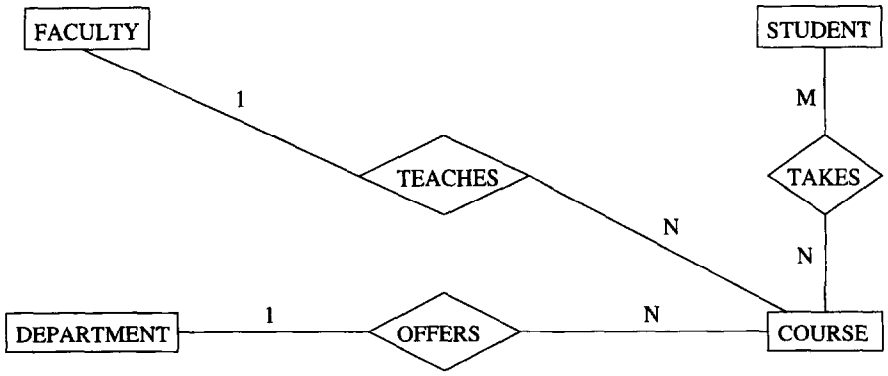


Fig. 1. An ER diagram for a university database.

can be a person, an item, or a concept about which an organization wants to store data. Entities sharing similar properties can be classified into entity types, such as EMPLOYEE and DEPARTMENT. Entities may have certain relationships with one another which can also be classified into relationship types. For example, MANAGES is a relationship type between entity types EMPLOYEE and DEPARTMENT. The relationship may be one-to-one, as MARRIAGE between exactly two PERSON entities, one-to-many as TEACHES between one PROFESSOR and many COURSE entities, or many-to-many as WORKS-ON between many EMPLOYEE and many PROJECT entities. In Chen's model, entities and relationships have properties, called attributes. For example, AGE is an attribute of an EMPLOYEE entity and HOURS-WORKED is an attribute of a WORKS-ON relationship between an EMPLOYEE and a PROJECT entity. An attribute can attain values of a certain value type. A multivalued attribute can have more than one value. An example is the DEGREE attribute of a PROFESSOR entity.

Each entity must have a unique identifier to distinguish it from other entities of the same type. This might be an attribute already in use, such as an employee's name, or might be an attribute introduced for its uniqueness, such as an employee's social security number. Chen compares the entity identifier to the concept of *primary key* in conventional databases. Relationships are identified by using the identifiers of all entities involved in the relationship. In the case of a relationship involving entities of the same entity type, "roles" may be assigned such as HUSBAND and WIFE in a MARRIAGE relationship. An entity may depend upon entities of another entity type for its existence. Chen calls this a "weak" entity. An example is DEPENDENTS (of an employee), since an employee's dependents would no longer be of interest if the employee left the company. An entity has an "ID-dependency" on another entity if it does not have its own identifier and can only be uniquely identified by its relationship with the other entity. For example, a city can only be uniquely identified within a particular state.

1.2. Entity–relationship diagrams

In the original ER model, an entity type is represented by a rectangle with the name of the entity type inside it. A relationship type is represented by a diamond, with the relationship type name inside. Related entity types are connected to this diamond by straight lines. Each line is marked with a "1", "*N*", or "*M*" to indicate 1 : 1, 1 : *N* or *M* : *N* relationship types. A weak entity type is enclosed within a double-lined rectangle, an "E" is placed in the relationship type diamond and an arrow is on the connecting line pointing toward the weak entity type. The double-lined rectangle is also used for an ID-dependent entity type, with an "ID" in the relationship type diamond and an arrow on the line, pointing to the dependent entity type. These are in the "upper conceptual domain" of the diagram (see Fig. 1). Attributes and their value types are shown in the "lower

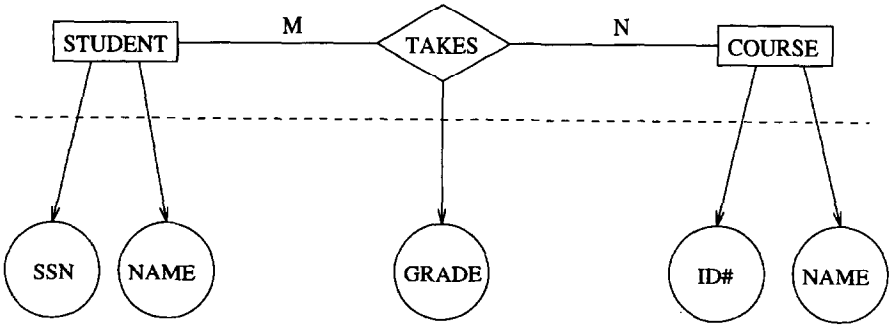


Fig. 2. Attributes and value types for STUDENT, TAKES and COURSE.

conceptual domain'' (see Fig. 2). An attribute's value type is represented by a circle with the value type name inside, connected by an arrow to its entity type. The attribute name is added by the connecting arrow unless the name is the same as the value type name. An example of different names is a value type DATE which is used for the BIRTH-DATE attribute of an EMPLOYEE entity. Multivalued attributes are indicated by putting ''1 : N'' beside the connecting arrow.

1.3. Data modeling emphasizing relationships

The original ER model was proposed to provide a unified view of data [1]. As noted by Hull and King [2], two opposing research directions in databases were initiated in the early 1970s. The relational model revolutionized the field by separating logical data representation from physical implementation. Semantic models were introduced primarily as schema design tools. The emphasis of the initial semantic models was to accurately model data relationships that arise frequently in typical database applications. Consequently, semantic models are more complex than the relational model and encourage a more navigational view of data relationships.

The ER model was the first semantic model centered around relationships rather than attributes. It views the world as consisting of entities and relationships between entities. The underlying philosophy is that an attribute is restricted to being a single fact about an entity, whereas a relationship can model the construction of more complex entities from other entities.

In recent years, object-oriented data modeling has become very popular, but the ER modeling approach is still also very popular and has found many new applications. The continuation of the international conference on ER modeling as well as many books and articles devoted to ER modeling in recent years bear witness to the importance of the ER approach. An examination of the evolution of the ER model is thus important to understand why this model is so popular. In this paper we examine and compare various extensions of ER models, pointing out how syntactical extensions have been made to enhance the representation and modeling power of the original ER model.

1.4. Organization

The modeling power of the ER approach largely depends on ER diagrams. That is why in this paper our discussion on the evolution of ER modeling will be largely centered around the evolution of ER diagrams. A good understanding of the semantics involved in ER modeling cannot be carried out without a careful study of syntactical representation in ER diagrams. Underlying a simple syntactical change may be an important semantic improvement, a shift of meaning, or some other changes crucial to the data modeling. A good example is the meaning of *ternary* relationship. Syntactically, a ternary relationship is very similar to binary relationships; however, until very recently [3], a detailed analysis of the binary/ternary cardinality combinations was not available. Seemingly minor syntactical differences may be the tip of big semantic icebergs; they should not be overlooked. Whenever appropriate, we will use a university schema (or its variations) as an example to compare different extensions. However, sometimes we have to use different examples for better illustration.

The rest of the paper is organized as follows. In Section 2 we provide a brief description of the enhancements to the original ER model. This includes a survey of terminology additions and diagram variations. In Section 3 we describe several extended ER models and include diagrams to illustrate their important features. Section 4 provides a similar account of object-oriented ER models. A brief sketch of object-oriented schema diagram is provided in Section 5. We evaluate the enhancements along with the extended models and close the paper by drawing some conclusions in Section 6.

We assume that the reader is familiar with the basic concepts of database systems. Some knowledge of object-oriented concepts is also desirable.

2. Expansions to the original ER model

More recent authors have added further explanation and used different terms to describe Chen's original model. Some of these follow.

2.1. Objects and classes

Batini et al. [4] speak of entity types and relationship types as classes of objects. According to them, an entity type represents a class of real world objects whereas a relationship type represents the aggregation of two or more entity types. They introduce the term ''rings'' to describe ''binary relationships connecting an entity to itself'', which are called recursive relationships by some authors. Teorey et al. [5] refer to Chen's ''three classes of objects: entities, attributes and relationships''. They explain, ''Entity sets were the principal objects about which information was to be collected and usually denoted a person, place, thing or event of informational interest. Attributes were used to detail the entities by giving them descriptive properties such as name, color and weight.'' In reference to Chen's ''basic ER model'', Markowitz and Shoshani [6] state, ''the atomic objects are called entities. Associations of entities are represented by relationships.'' They add, ''Objects are qualified by attributes and are classified into object sets.'' It is helpful to understand the basic model in terms of objects and classes as this terminology will be used to explain advanced concepts.

2.2. Relationship constraints

In Chen's original model, the connectivity of a relationship specifies the mapping of associated entity occurrences. The values for connectivity are either "one" or "many". Teorey et al. [5] define cardinality as the actual number associated with the term "many". Like Chen, they use a $1:1, 1:N, M:N$ notation in their model. Batini et al. [4] use a more specific notation which shows the minimum cardinality, called "min-card", and the maximum cardinality, called "max-card". This is expressed in the diagram as (min-card, max-card). For example, (1,5) means that an entity must participate in a minimum of 1 and a maximum of 5 relationship instances at all times. The minimum of 1 also implies that the entity's participation in a relationship is mandatory. Another example is (0,10) where the 0 means that the entity is not required to participate in a relationship. A value of n for max-card means "no limit." This convention is also used by Navathe et al. [7]. Czejdo et al. [8] use a similar notation, except an asterisk is used instead of n to represent no limit on the maximum number.

2.3. Composite attributes

As pointed out by Hull and King [2], the way to represent a composite attribute in the original model is to use an entity type and a relationship type. Their example is ADDRESS, which is an entity type having attributes STREET, CITY and ZIP. ADDRESS is associated with another entity type, PERSON, by a LIVES-AT relationship type. Batini et al. [4] add a composite attribute to their model. They also use ADDRESS as an example and show it in their diagram as a composite attribute in an oval. Each single attribute (STREET, CITY, etc.) is shown by a small circle with the name alongside, connected by a short line to the oval. The oval itself is connected by a line to the entity type PERSON and (min-card,max-card) shown for the composite attribute rather than each single attribute. Elmasri and Navathe [9] also allow a composite attribute in their basic model, showing a tier of attributes, each represented by an oval with the name inside.

2.4. Identifiers

Although Chen discussed identifiers, he did not include them in his ER diagram. Later authors generally do, usually by underlining the attribute name or names if attributes are included in the diagram.

Batini et al. [4] take a slightly different approach to identifiers in that either or both attributes and other entity types can be identifiers. They define an identifier for entity type E as a set:

$$I = \{A_1, \ldots, A_n, E_1, \ldots, E_m\},$$

where $n \geq 0$, $m \geq 0$, $n + m \geq 1$

where $A_1, \ldots, A_n$ are attributes and $E_1, \ldots, E_m$ are entity types other than E. An identifier is classified as simple if $n + m = 1$ or is composite if $n + m > 1$. An identifier is internal if $m = 0$, or is external if $n = 0$. An identifier is mixed if $n > 0$ and $m > 0$. They observe that entity types having internal identifiers are sometimes called "strong" entity types and those which have only external identifiers are sometimes called "weak" entity types. In their diagram, they use a small open circle for most attributes. If an attribute is simple and internal, the circle is blackened. A composite internal identifier is shown by connecting an additional blackened circle to the lines of the attributes which form the identifier. Mixed identifiers also have lines extending to the external entity type's connecting line. Because of this the double-lined weak entity type rectangle used by Chen is absent. Teorey et al. [5] view attributes as being of two types, "identifiers" and "descriptors." Identifiers uniquely distinguish between the occurrences of an entity type, while descriptors describe an entity occurrence.

2.5. Diagrams for basic models

All authors reviewed in this paper use a rectangle to represent an entity type and a diamond to represent a relationship type in their ER diagrams. Some, but not all, include attributes in the diagram, and those who do so use either an oval with the name inside (identifiers underlined), a small circle with the name by the side (identifiers have a darkened circle) or a bent line with the name by the side (identifier not indicated). Cardinality is either the $1:1, 1:N, M:N$ notation as used by Chen or the (min-card,max-card) notation discussed in Section 2.2. Elmasri and Navathe [9] use a double line connecting an entity type and a relationship type to show total (mandatory) participation. Teorey et al. [5] blacken the half of the relationship type diamond that is toward an entity type whose connectivity is many (toward a one remains unblackened), and put a small circle on the line near the diamond if membership is optional (mandatory is a line with no circle).

Wertz [10] discusses other diagramming styles. One of these, attributed to Clive Finkelstein, skips the diamond and puts a crow's foot symbol on the connecting line at the many side of a relationship type, and also uses a vertical bar (across the connecting line) to indicate a mandatory participation and an open circle for an optional participation. Another approach, attributed to Charles Bachman, also skips the diamond and uses an arrow on the many side of the relationship type. He uses a small open circle at the junction of the entity type rectangle and the connecting line to indicate optional participation and a darkened circle for mandatory. Regarding Chen's model, Wertz [10] also states, "minor variants include use of a single arrow to indicate the one side of a relationship and a double arrow to indicate the many side, use of a dot to indicate the many side of a relationship and use of a double box to indicate a weak entity."

2.6. Summary

Various syntactical revisions discussed in this section offer increased convenience for database designers. Expansions discussed in this section are concerned with entities as well as on relationships. Although these revisions or expansions do not significantly enrich the modeling power of the ER model, some of them have provided new ways to interpret the original model. For example, viewing an entity type as a class provides an interpretation of this model, thus enriching its semantics.

3. Extended ER models

A number of extended ER (EER) models have appeared in recent literature. In general, their contribution is to add the generalization abstraction to Chen's original model and variations thereof which were discussed in Section 2 above.

3.1. The EER model by Teorey, Yang and Fry

Teorey et al. [5] state that "The introduction of the category abstraction into the ER model resulted in two additional types of objects: subset hierarchies and generalization hierarchies." They describe the subset hierarchy as specifying possibly overlapping subsets and the generalization hierarchy as specifying strictly non-overlapping subsets. An entity type E_1 is a subset of another entity type E_2 if every occurrence of E_1 is also an occurrence of E_2. In a generalization hierarchy, where entity type E is a generalization of entity types E_1, E_2, ..., E_n, every occurrence of entity type E is also an occurrence of one and only one of the entity types E_1, E_2, ..., E_n. For example, in the EER diagram of Fig. 3, FACULTY and STUDENT are subsets of PERSON (assuming that a person can be a faculty member as well as a student), whereas COURSE is a generalization of CREDIT-COURSE and NON-CREDIT-COURSE. A generalization hierarchy is called an "IS-A" exclusive hierarchy. Each subset entity type is shown in the EER diagram with a fat double-lined arrow pointing to the entity type of which it is a subset. The generalization hierarchy is shown with a fat double-lined arrow from each subset to a hexagon which contains a name characterizing all subsets (COURSE TYPE in the above example), and another fat arrow from the hexagon to the generalization entity type. In the steps for using their model, Teorey et al. instruct the user to put identifier and generic descriptors in the generic entity type and to put identifier and specific descriptors in the subset entity types. Thus in our example EER diagram, SSN and NAME are shown as attributes of PERSON, while FACULTY and STUDENT have their own unique attributes besides the identifier SSN.

3.2. The EER model by Markowitz and Shoshani

Markowitz and Shoshani [6] offer an extended ER model which is similar to that of Teorey et al. [5] but has more features and definitions. They describe generalization as

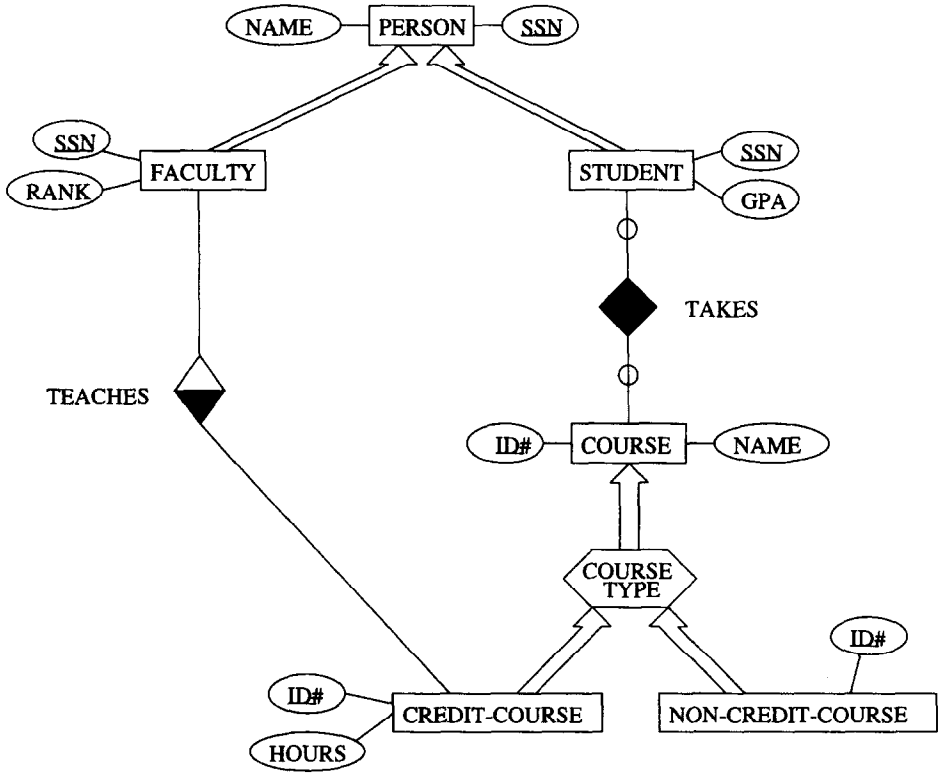


Fig. 3. An EER diagram for a university database [5].

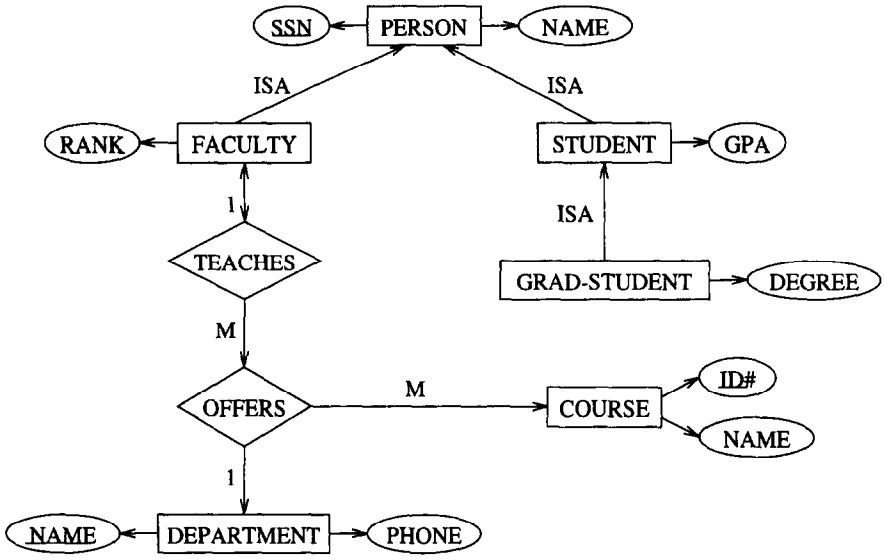


Fig. 4. A sample EER diagram [6].

"an abstraction mechanism that allows viewing a set of entity sets (e.g. SECRETARY,FACULTY) as a single generic entity set (e.g. EMPLOYEE)." Note that the authors use the terms "entity-set" and "relationship-set" to respectively denote entity types and relationship types. For purposes of clarity, we will use the terms "entity type" and "relationship type." Markowitz and Shoshani state that generalization defines a transitive relationship, so if entity type E_1 is a direct generalization of entity type E_2, and E_2 is a direct generalization of entity type E_3, then E_1 is a transitive generalization of E_3. Specialization is the inverse of generalization. A specialization entity type inherits the attributes of all its direct and transitive generic entity types, including the entity type identifier. Such attributes are called inherited attributes. Entity types which are not weak, and are not a specialization of other entity types, are called "independent" entity types. In the example EER diagram (Fig. 4), PERSON is a generalization of FACULTY and STUDENT, whereas STUDENT is a generalization of GRAD-STUDENT. Thus PERSON is the transitive generalization of GRAD-STUDENT. GRAD-STUDENT inherits all the attributes from STUDENT and PERSON, besides having its own unique attribute DEGREE. Markowitz and Shoshani also allow "full" aggregation, so relationship types can associate both relationship types and entity types rather than only entity types. For example, in the EER diagram of Fig. 4, full aggregation allows the association between the TEACHES relationship type and the OFFERS relationship type. Markowitz and Shoshani define their diagram as a directed graph. Arrows show the direction and represent interaction of elements and existence dependencies. The graph must be acyclic since directed cycles imply redundant entity types.

3.3. The ECER model

The extended conceptual entity–relationship (ECER) model developed by Czejdo et al. [8] uses graphical interfaces to formulate queries and updates. This concept retains the ER diagram advantages normally used only for database design and applies them to a "point and click" user interface. A number of operators for updates, retrievals, computations, etc., are included.

Czejdo et al. define three types of generalization/specialization. Type 1 involves exactly two entity types, one being a subset of the other. Note that in the original text, the authors use the term "entity set" to denote an entity type. For clarity's sake, we will stay with "entity type". A Type 1 generalization/specialization is shown in the diagram with a subset symbol on the line connecting the two entity types. In the sample ECER diagram of Fig. 5, GRAD-STUDENT is such a specialization of STUDENT. Type 2 involves one generalization entity type A and any number of specialization entity types $B_1, B_2, \ldots, B_n$, where $A = B_1 \cup B_2 \ldots \cup B_n$. This is represented in the diagram by connecting each entity type to a small circle with a "U" inside. For example, in Fig. 5, PERSON is the union of FACULTY and STUDENT, thus every person is a faculty member, a student, or both. Type 3 is like Type 2 except the specialization types are disjoint. They are connected in the diagram to a circle containing a plus symbol. Consider the example of CREDIT-COURSE and NON-CREDIT-COURSE being such specializations of COURSE.

3.4. The EER model by Hohenstein and Gogolla

The Hohenstein and Gogolla model [11] is defined in connection with proposing a calculus for the model. Their model is based on Chen's and is similar to Teorey et al. [5] in that it includes generalization. Hohenstein and Gogolla present generalization and specialization as "type construction" with input types and output types. The "already defined or basic" input types are used in the construction of

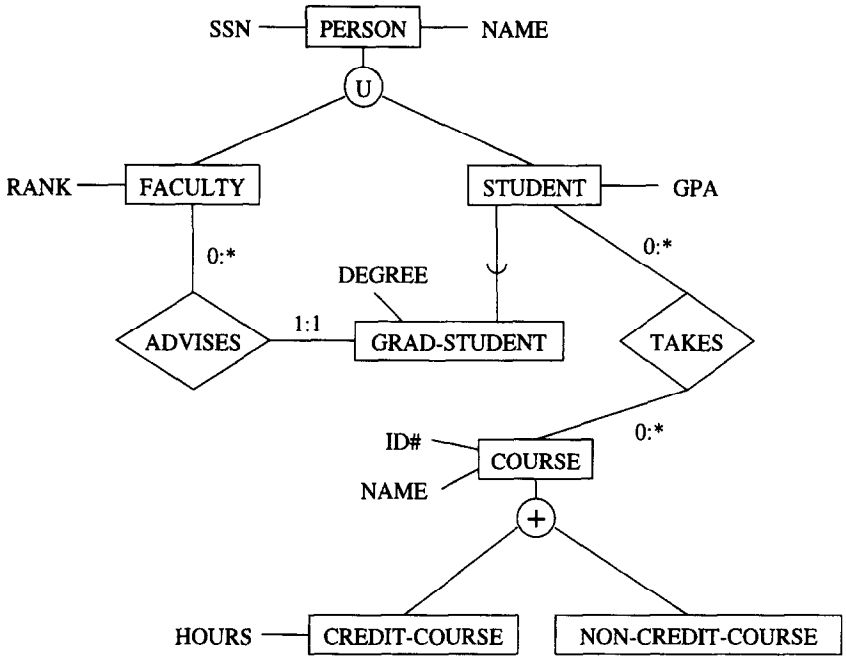


Fig. 5. An ECER diagram for a university database.

output types, meaning that "all entities from the input types are put together and are distributed over the output types". This is shown in the diagram by connecting all input types to the base of a triangle, and connecting the apex of the triangle to the output types. Either a subset symbol or an equality symbol is inside the triangle. For example, in the sample EER diagram of Fig. 6, FACULTY and STUDENT are the input types used in the construction of output type PERSON. The model also allows attributes to have values of complex and enumerated data types. Users can define operations to manipulate complex data items. In the diagram, an attribute's data type is indicated along with the attribute's name. Attributes may be multivalued and the set of values for a multivalued attribute can be classified into three different categories, namely "sets", "bags" and "lists." Elements in a set occur only once, elements in a bag may occur more than once, whereas elements in a list are stored in an enumerated form. Besides generalization and specialization, this model also includes the concept of "complex structured" entity types. An entity of a complex structured entity type is composed of other entities, called its "components." In our example diagram, DEPARTMENT is a complex structured entity type as it includes two components, namely CHAIR and GRAD-COMMITTEE (a list). Both these components are of another entity type FACULTY.

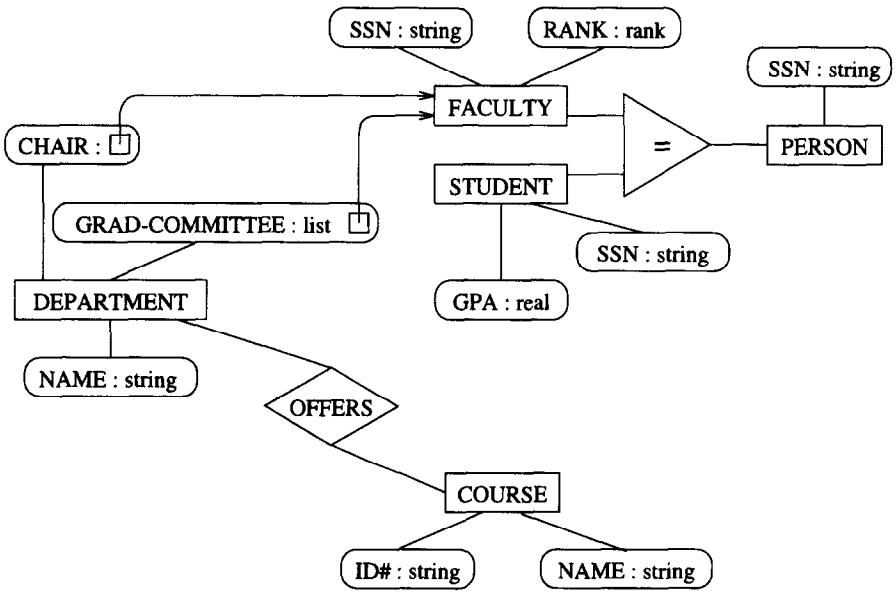


Fig. 6. An EER diagram for a university database [11].

3.5. The ECR model

Navathe et al. [7] propose an entity–category-relationship (ECR) model which they use for view integration. A category is "a subset of entities from an entity type." A set of entities is called an object class, whether the set is an entity type or a category. The categories usually share most attributes, but some are not shared. For example, FACULTY and STUDENT are categories of entity type PERSON. Both share the attributes SSN and NAME, yet have their own unique attributes RANK and GPA respectively. The ECR model uses diagram notations similar to Czejdo et al. [8] (See Fig. 5). The categories are shown in the diagram as hexagons, connected to the entity type with a subset symbol drawn on the line. Shared attributes are connected to the entity type and non-shared category attributes are connected to the hexagon. The categories are the focus for view integration. Categories are defined for each data source for the integration and those categories are compared during the process.

3.6. Summary

Discussion on the enhanced power for modeling has accompanied ER approach for two decades. Smith and Smith [12] present the concepts of generalization and aggregation. The semantic data model of Hammer and McLeod [13] introduced the concepts of class and subclass lattices, as well as other advanced modeling concepts.

Various extensions of the original ER model, as we have already summarized in previous sections, address the semantics in various degrees. Behind the syntactical differences of various extensions is the enriched semantics about relationships among entities. For example, many of the proposed syntactical changes are around generalization/specialization, a clear indication of semantic improvement. The models summarized in this section are stand-alone data models, but they can also be viewed as stepping-stones toward object-oriented data models or object-oriented ER models, as we compare in the next section.

4. Object-oriented ER models

With continuing advances in object-oriented databases, interest has increased in favor of including features of the object-oriented paradigm, in order to adapt the ER model for object-oriented database design. These features include generalization and inheritance, enforcement of the information hiding principle, abstract data type encapsulation and message passing. The following models incorporate object-oriented concepts into the ER model.

4.1. The OOER model

Navathe and Pillalamarri [14] use an object-oriented approach for their data model. Since semantic models accomplish their goals by using abstraction, which they define as a "semantically irreducible modeling primitive that allows us to model a fundamental type of data relationship in a way that hides detail and differences, and concentrates on the common properties of a set of objects", they present their model in terms of five commonly used abstractions. These are aggregation (IS-PART-OF), generalization (IS-A), classification (IS-A-CLASS-OF), association (IS-ASSOCIATED-WITH), and identification (IS-IDENTIFIED-BY).

Navathe and Pillalamarri examined the basic ER model because of its widespread popularity, but found it deficient for its failure to explicitly model generalization and classification, and for its limited capability for describing the interaction between constructs. This led to the object-oriented entity–relationship (OOER) model which retains many features of Chen's model, but expands its capabilities.

The aggregation abstraction involves the entity (also referred to as an object), which Navathe and Pillalamarri define as something that exists in the real world and has attributes to describe it. Attribute values of a similar type are grouped into attribute classes or domains. Entities having similar attributes are classified into entity classes or entity types, and an entity can be defined as either an aggregation of attributes or an aggregation of other entities (which distinguishes the OOER model from the ER model). Many operations, including update, relational (select, join and project), comparison, aggregation and arithmetic can be defined for the entity classes.

With the generalization abstraction, two or more object classes can be generalized to form a higher level object class. Specialization is the inverse notion whereby new object classes can be defined to be subclasses of one or more object classes. For example, in the OOER diagram of Fig. 7, EMPLOYEE is a generalization of MANAGER, ENGINEER and SECRETARY, whereas OVERSEAS-CUSTOMER is a specialization of CUSTOMER. The construct used is the object class of type superclass or subclass, which are relative terms since the same object class can be either depending on whether it is formed by generalization or specialization. In the case of multiple inheritance, the user must specify precedence. Generalization object class relationships can be further defined with *Set Exclusion*, *Set Intersection* or *Set Equality* constraints. Respectively, these indicate no common instances, some common instances, or a 1 : 1 relationship between objects participating in a generalization. All operations that have been defined for the entity classes are also applicable to the subclasses and superclasses.

The classification abstraction is based on the notion of a set. This concept forces the distinction between individual instances and a class of those instances. If certain properties apply to an entire set, then they can be modeled as class attributes. Grouping of instances can form an entity class, a relationship class, or a class of a superclass/subclass. In

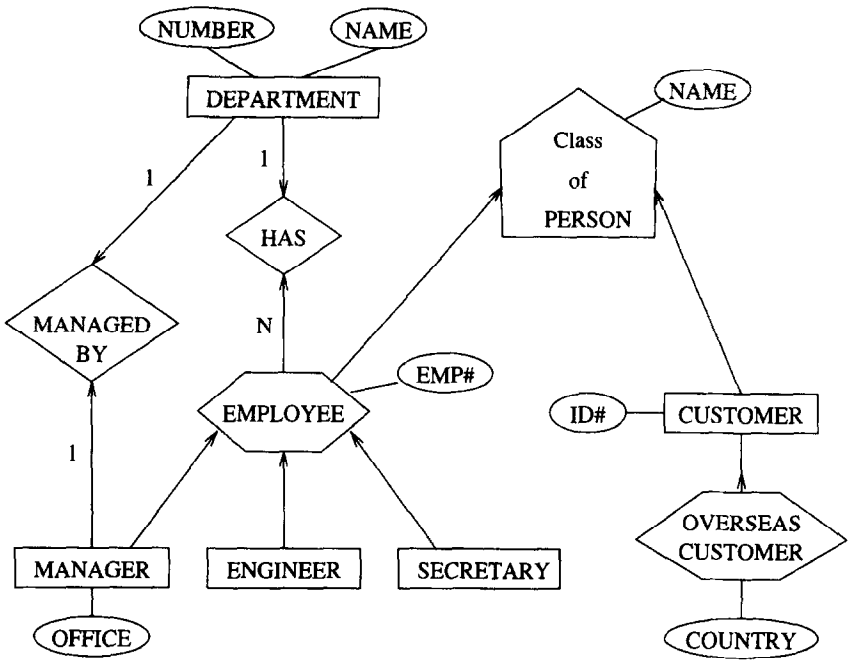


Fig. 7. An OOER diagram for a company database.

addition, a metaclass is a grouping in which all objects are themselves classes, in other words a set of sets of instances. As indicated in Fig. 7, the instances of metaclass PERSON are entity class CUSTOMER and generalization object class EMPLOYEE. Aggregate operations, like count, sum, etc., can be defined for the class attributes.

The association abstraction in the OOER model is the relationship, which represents the interaction between object classes as a higher level object class. It exhibits an existence dependency on the objects participating in the association.

Identification is the abstraction used to uniquely identify the object structures that have been formed. Each entity class has at least one attribute which is declared as its identifier (primary key) and is used to access an object. The instances of a generalization object class can be identified by the corresponding instances in the defining class(es) or by any key attributes of the generalization class itself.

Navathe and Pillalarnarri [14] evaluate their OOER model in terms of its level of object-orientation. They conclude that it has structural and operational but not behavioral object-orientation. Structural object-orientation refers to allowing for the defining of data structures to represent complex objects. Operational object-orientation is allowing the definition of generic operators to deal with complex objects in their entirety, rather than decomposing into primitive operations on simpler objects. The OOER model is defined such that structure and operations are encapsulated together in each object class. However, the syntactic details for specifying abstract data types, data encapsulation, message passing and operator overloading have yet to be formulated, so behavioral object-orientation is not in place.

The OOER model uses conventional ER diagram notations to denote entity classes, attributes and relationship classes, with the addition of arrows from related entity classes pointing to the relationship class diamond. Both generalization and specialization are represented in the diagram by a six-sided polygon. The difference is shown by the direction of the connecting arrow, which is toward the polygon for generalization and away from the polygon for specialization. A class is shown in the OOER model as a five-sided polygon. Objects which form the class are connected by an arrow pointed toward the polygon, whereas objects that are attributes of the class are connected without arrows.

4.2. The BIER model

In a behavior-integrated entity–relationship (BIER) approach to the design of object-oriented databases, Kappel and Schrefl [15] establish a static model and a behavior model. The static model represents the structural properties of real-world objects and the behavior model explains real-world events in terms of object behavior. While the behavior model is based on a Petri net graph, the static model uses an extended ER diagram approach. In the static model, real-world structures are represented as object types (similar to entity types in the ER model). Object characteristics are represented as attributes. The value of an attribute could be of a data type or an object type. This concept is used to represent inter-object relationships. Generalization and inheritance are also included.

The model defines the concepts of "primitive" and "complex" objects. Primitive objects are stand-alone as they are not defined upon other objects, whereas complex

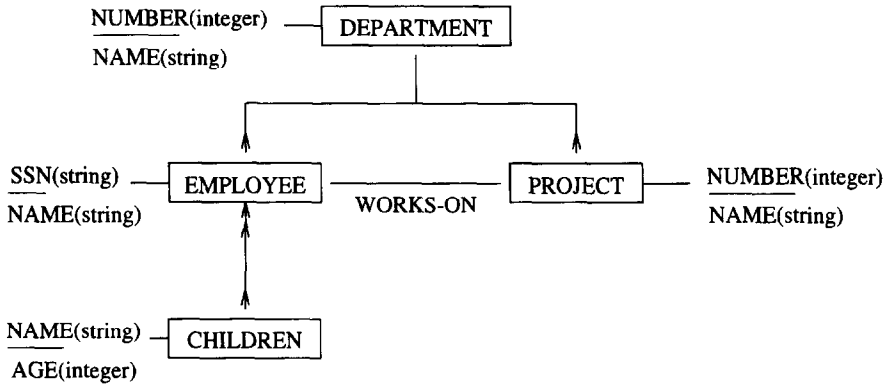


Fig. 8. A static BIER model for a company database.

objects are defined upon one or more lower level objects. Complex objects can be further classified into "group" objects and "aggregate" objects. A group object is defined upon lower level objects of the same class, called its "members"; this abstraction is called grouping. An aggregate object is defined upon lower level objects of different classes, called its "components," with this abstraction being referred to as aggregation. Aggregation and grouping employ the relationship concept. The lower level objects are called "independent" if they participate only in a relationship that is represented by the higher level object. They must exist before the higher level object is created, and they continue to exist if the higher level object is deleted. "Dependent" objects are private to a higher level object and are created and deleted along with that object. In our example diagram (Fig. 8), DEPARTMENT is an aggregate object type with dependent component types EMPLOYEE and PROJECT. This aggregation may be used to represent the WORKS-FOR relationship type between DEPARTMENT and EMPLOYEE, and the CONTROLS relationship type between DEPARTMENT and PROJECT. An EMPLOYEE object's WORKS-ON attribute can have a value of object type PROJECT. EMPLOYEE itself is a group object type with dependent member type, CHILDREN.

4.3. The OOERM model

Like the BIER model [15], the object-oriented entity–relationship model (OOERM) developed by Gorman and Choobineh [16] also includes behavioral object-orientation. Gorman and Choobineh present the object-oriented view as a natural extension of the entity–attribute-relationship view: "When we think of an object or entity (e.g. a bicycle), we consider its attributes, such as color. We also naturally consider the things we can normally do with it (ride it, lock it...)." In defining terms, they discuss an object as "a 'package' of information and a description of its manipulation". This corresponds to a set of related variables and a set of operations on these variables in procedural programming terms. A message is defined as a specification of one manipulation on an object, the specification consisting of the receiver (object to be manipulated), the selector (symbolic name of the manipulation), and optional arguments (possible other objects taking part). A message resembles a procedure call. A method is a description of a single type of manipulation, a procedure-like object found in receivers and selected in response to messages.

In object-oriented programming, a class or object type corresponds to an entity type in the ER model, and the object's set of variables and methods correspond to attributes. The visibility of an object, which refers to what objects it can access and what objects it can be accessed by, is similar to the relationship concept in the ER model. Message passing uses object visibility, allowing the sending object to access the variables of the receiving object.

An OOERM schema consists of a number of diagrams, called object-oriented entity–relationship diagrams (OOERD). OOERDs are used to represent both the structural and behavioral properties of the database. A "structural" OOERD is drawn to define the classification and hierarchy of objects, including inter-object relationships. In addition, a separate OOERD is drawn for modeling each application process. In an OOERD, entity classes and relationships use the normal rectangles and diamonds, while attributes are represented by circular nodes with the attribute's name beside (Fig. 9). A generalization hierarchy is represented by a fat double-lined arrow pointing to the higher level class. If the tail of the arrow is hollow, it represents a subclass that is a subset of the higher level class with no additional properties. If the tail is filled in, it represents a subclass that is a specialization of the higher level class with its own additional properties. As shown in Fig. 9, AUTHOR is a subset of PERSON whereas STUDENT is a specialization of PERSON.

Methods are shown in an OOERD as ovals with the method name inside. They are attached to their entity class by a solid line, similar to an attribute. Message passing is shown by a dotted-line arrow from the sending class to the receiving class. The position of the message in a sequence of messages is shown by an integer in parentheses. Optional parameters being sent are shown above the arrow and optional results returned are shown below the arrow.

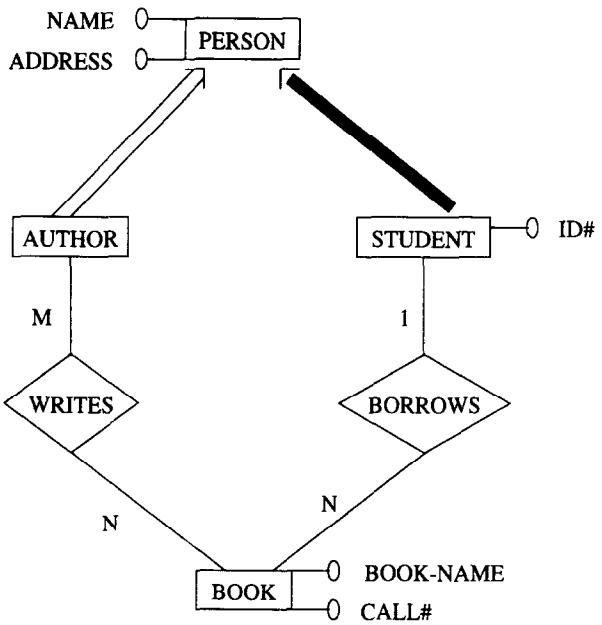


Fig. 9. A sample structural OOERD.

Forking dotted-line arrows are used to indicate conditional branching and each branch is marked with "T" for true or "P" for false. These methods, message passing and branching indicators are referred to as dynamic icons. All entity classes have five predefined operations as implicitly specified properties. These operations are "create", "add to subclass", "remove from subclass", "delete" and "select". They have the same default definition for all classes. Messages invoking these predefined operations can be specified interactively by the user or in the action portion of methods. A "where" clause, similar to SQL, is used to identify specific class instances to which the operation applies. The predefined operations are represented in the OOERD with a method oval superimposed over the entity class to which it applies. The letter representing the predefined operation is placed in the right portion of the oval and the ordinal sequence of the operation is in parentheses in the left portion of the oval. Another feature of this model is attribute derivation. In this case the value of an object's attribute can be derived as a result of a message received by that object.

A process can have a sequence of many operations. OOERDs are intended to model individual application processes encoded in the data model scheme description. The diagram provides a pictorial view of state changes resulting from process execution. Therefore, the authors recommend that a separate OOERD should be drawn for each application process and each OOERD should contain only the structural and operational details relevant to that process. For example, in the OOERD for the "Book-Borrowing" process (Fig. 10), only the relevant classes and operations are shown.

4.4. Comments

In the 1980s, we have witnessed a shift in the emphasis of data modeling: the general interests related to conceptual modeling (inspired partly by artificial intelligence) [17] have converged to object-oriented data models. According to Hull and King [2], essentially, semantic models encapsulate structural aspects of objects whereas object-oriented languages encapsulate behavioral aspects of objects.

Object-oriented data models have incorporated some important concerns addressed by conceptual modeling (such as aggregation) as well as concerns addressed by logical modeling models (such as non-first normal form in some extended relational models and navigation in the network model). As a consequence, the boundary between logical/conceptual models is blurred. An object-oriented data model can be viewed as a model at the level of conceptual modeling as well as a model at the logical level.

As pointed out by Loomis [18], an object model includes relationships between objects. These relationships, which are associations between objects, are like the relationships between entities in ER data modeling. An object model may also include containment relationships and recursive structures. An object DBMS builds these relationships into the object database and can use them directly at runtime when returning objects to applications. The relationships

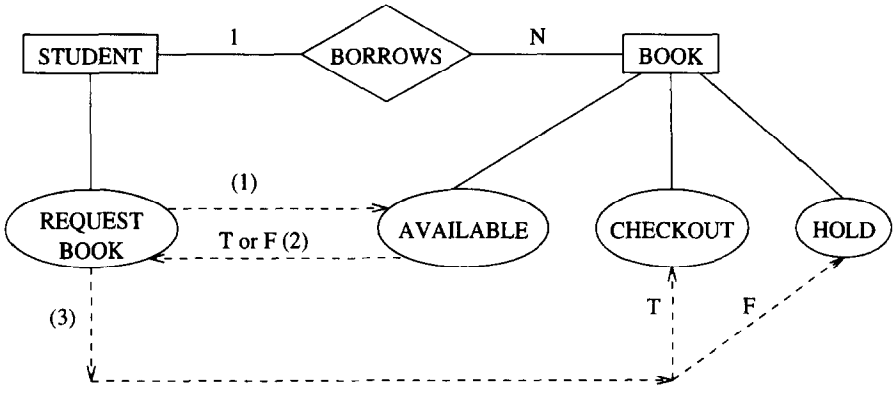


Fig. 10. An OOERD for the "book-borrowing" process.

in ER data models are purely abstract, while object-oriented data models carry relationship semantics into implementation.

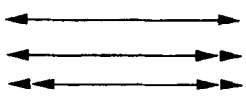
Another aspect that must be pointed out is the behavior part of a full fledged object-oriented database. Chen [19] argues that ER does have a behavior part, but without giving any explanation. He claims, "... in the ER world view, 'data' and 'processes' are on equal footing, while in the object-oriented world view, 'processes' are encapsulated with the data accessed. If we think deeply, the encapsulation of processes with data is a 'packaging' issue and has nothing to do with the argument on whether object-orientation has more dynamic behavior specifications than ER (or other paradigms). The object-oriented way of packaging processes with data will work well in certain kinds of applications but not in other applications." Although Chen himself does not provide any elaboration, some models as summarized in this section have indicated how object-orientation and ER can approach each other.

5. Object-oriented schema diagrams

Finally, let us take a look at some related issues *beyond* ER modeling. Particularly, we provide a brief sketch of object-oriented schema diagrams as used in ODMG-93 [20].

Graphical representations have been used in object-oriented data models to represent database schema. We will refer to them as *object-oriented schema diagrams*. Here we examine the graphical representation used in *Object Database Standard* ODMG-93, which narrows the gap between ER modeling and object-oriented modeling. ODMG-93 has adopted basic notations from ER diagrams with important extensions. The graphical representation uses the following notations:

1. object types are shown as rectangles;
2. relationship types are shown as labeled lines;
3. the cardinality permitted by the relationship type is indicated by the arrows on the ends of the lines:



one-to-one
one-to-many
many-to-many

4. large gray arrows point to super-type from subtype.

In this graphical representation, object types are represented in exact the same way as entity types in ER models. This is a clear indication that object types are extended from entity types. Relationship types have made significant changes. The traditional diamond representation in ER diagrams is replaced by labeled lines. The labels are the names of the relationship types; however, relationships in ODMG-93 have directions. A relationship in ER diagrams is now replaced by a *pair* of relationships (called *inverse relationships*) with opposite directions. For example, the connection between two object types `course` and `student` can be represented by relationship `take` from `student` to `course` and its inverse relationship `taken-by` from `course` to `student`. An ODL (Object Definition Language) definition for the interface of course should include the following:

```
interface Course {
...
relationship Set < Student > is_
taken_by inverse Student:takes;
...
}
```

Fig. 11 is an example of object-oriented schema diagram (attributes not included).

The improved representation of relationships in ODMG-93 diagrams, along with the explicit inclusion of cardinality as mentioned earlier (such as one-to-many or many-to-many), has brought significant semantic enrichment. The lack of directions in relationship types in the original ER diagram has always been a potential source of confusion in semantics; the problem is now solved. More importantly, the explicit inclusion of the directions can facilitate the construction of navigation paths in answering queries. Construction of navigation paths is crucial to object-oriented databases.

A sample query in SQL-like syntax is:

```
select teaching (student: x.name,
professor: z.name)
from x in Students, y in x.takes, z in
y.taught-by
where z.rank = ''associate_professor''
```

Note that directions of relationship are important in specifying query paths: the relationship `takes` follows the direction from `student` to `professor`, whereas the relationship `taught-by` follows the direction from `professor` to `student`.

6. Discussion

In this paper we started by viewing ER as the first relationship-centered data model and we examined and compared various extensions. Although simple extensions to the original model have only added minor improvements, extended ER models have significantly improved the modeling power, bridging the gap to object-oriented data models. We noted that behind syntactical extension lies semantical enrichment.

The entity–relationship model is a very powerful and useful tool for many modeling purposes. It is very popular and all authors speak highly of its capabilities. Its fundamentals are relatively simple, yet effective, which seems to explain its wide following. The diagram literally draws a

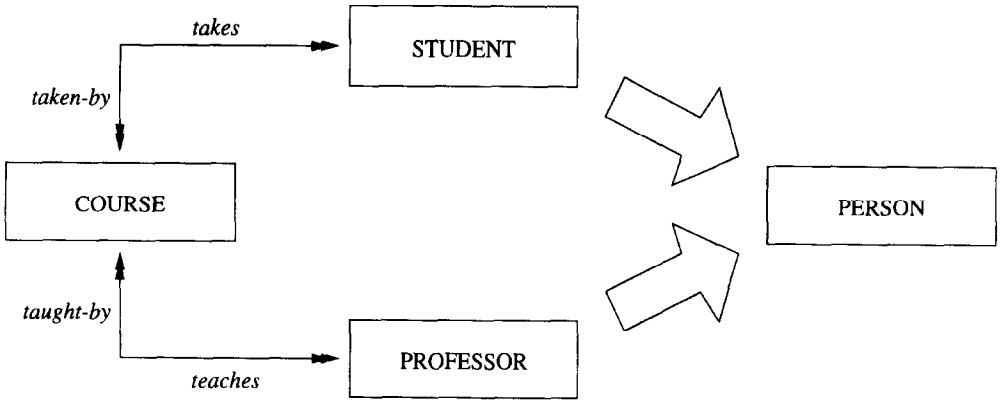


Fig. 11. An object-oriented schema diagram.

picture of the real world being modeled, and a picture can be understood by even the less sophisticated end users.

The enhancements and variations discussed in Section 2 add relatively little to the power of the model. Many of them are differences reflecting personal preferences, such as ways of drawing the diagrams. In fact, some of these may hinder the use of the basic model because they could cause confusion. It may help to have a standard for ER diagrams so that one set of symbols could be used by all database designers. This could be particularly helpful to end users who may interact with many designers but would not like to re-learn the notations each time.

Some of the differences shown in Section 2 are with respect to terminology or presentation. The original ER model [1] was very straightforward and practical. Later authors attempted to give more scholarly dissertations and spent more time on definitions and terminology. More recent authors have tended to use terms such as "object" and "class" that are more consistent with object-oriented modeling concepts. One concept from Section 2 that has a notable difference is cardinality. Chen's model relied on the $1:1, 1:N, M:N$ notation, but the (min-card,max-card) notation adds the capability to place lower and upper limits on an entity's participation in a relationship. This could be used to specify cardinality ratios as well as participation constraints. By contrast, the composite attribute concept may be lost by the the time the diagram is mapped to a relation. However, it is still helpful for understanding the relationship among attributes during the conceptual modeling and analysis stages.

The major contribution of the extended models in Section 3 is generalization. While there are some differences in definitions and diagrams, the point is that a hierarchy of classes is recognized and the inheritance concept is used either to share or to separate specific attributes. In Section 4, we examined the OOER model by Navathe and Pillalamarri [14] that adds more to the terminology discussion, but it is similar to those discussed in Section 3. A notable exception is the "class" concept represented by a five-sided polygon, particularly when used as a metaclass. They present a commendable dissertation in object-oriented terms.

The BIER model by Kappel and Schrefl [15] and the OOER model by Gorman and Choobineh [16] both represent a considerable advancement, as both can be regarded as truly object-oriented data models. The incorporation of methods, messages and operations makes these a better choice for object-oriented database design. As object-oriented databases become more popular, these models and their inevitable clones will undoubtedly find a more widespread audience. Finally the object-oriented schema diagrams as used in ODMG-93 reveal the prevalence of ER notations, as well as the conceptual relationship between these two data modeling approaches (i.e. entity–relationship and object-oriented).

7. Conclusions

In the past two decades we have witnessed a very powerful tool, the entity–relationship model, grow into a widely recognized technique. The enhancements to this model, particularly generalization and an object-oriented approach, have added capability and power to the original model. Many articles written within the past decade have helped by discussing and explaining the concepts, as well as by offering examples of the model's use. Many of the extensions have added a little to the expressive power of the ER model, but later extensions, especially those related to object concepts have introduced some added capabilities.

Acknowledgements

The author would like to acknowledge Munib Siddiqi's efforts and contributions during the earlier parts of this project. Comments by Z. Chen have been very helpful.

References

[1] P. Chen, The entity–relationship model: Towards a unified view of data, ACM Trans. Database Systems 1 (1) (1976) 9–36.

[2] R. Hull, R. King, Semantic database modelling: Survey, applications, and research issues, ACM Computing Surveys 19 (3) (1987) 201–260.
[3] T. Jones, I. Song, Analysis of binary/ternary cardinality combinations in entity–relationship modeling, Data and Knowledge Engineering 19 (1) (1996) 39–64.
[4] C. Batini, S. Ceri, S. Navathe, Conceptual Database Design, Benjamin/Cummings, Redwood City, CA, 1992.
[5] T. Teorey, D. Yang, and J. Fry. A logical design methodology for relational databases using the extended E-R model. ACM Computing Surveys, 18(2):197–222, 1986.
[6] V. Markowitz, A. Shoshani, Representing extended entity–relationship structures in relational databases: A modular approach, ACM Trans. on Database Systems 17 (3) (1992) 423–464.
[7] S. Navathe, R. Elmasri, J. Larson, Integrating user views in database design, IEEE Computer 19 (1) (1986) 50–62.
[8] B. Czejdo, R. Elmasri, M. Rusinkiewicz, D. Embley, A graphical data manipulation language for an extended entity–relationship model, IEEE Computer 23 (3) (1990) 26–35.
[9] R. Elmasri, S. Navathe, Fundamentals of Database Systems, Benjamin/Cummings, Redwood City, CA, 1994.
[10] C. Wertz, Relational Database Design, CRC Press, Greenwich, CT, 1993.
[11] U. Hohenstein, M. Gogolla, A calculus for an extended entity–relationship model incorporating arbitrary data operations and aggregate functions, in: C. Batini (Ed.), Entity–Relationship Approach: A Bridge to the User, Proc. 7th Int. Conf. on Entity–Relationship Approach, 1988, Rome, Italy, North-Holland, 1988.
[12] J. Smith, D. Smith, Database abstractions: Aggregation and generalization, ACM Trans. Database Systems 2 (2) (1977) 105–33.
[13] M. Hammer, D. McLeod, Database description with SDM: A semantic data model, ACM Trans. Database Systems 6 (3) (1980) 351–386.
[14] S. Navathe, M. Pillalamarri, Toward making the ER approach object-oriented, in: C. Batini (Ed.), Entity–Relationship Approach: A Bridge to the User, Proc. 7th Int. Conf. on Entity–Relationship Approach, 1988, Rome, Italy, North-Holland, 1988.
[15] G. Kappel, M. Schrefl, A behavior-integrated entity–relationship approach for the design of object-oriented databases, in: C. Batini (Ed.), Entity–Relationship Approach: A Bridge to the User, Proc. 7th Int. Conf. on Entity–Relationship Approach, 1988, Rome, Italy, North-Holland, 1988.
[16] K. Gorman, J. Choobineh, The object-oriented entity–relationship model (OOERM), J. Management Inf. Sys. 7 (3) (1991) 41–65.
[17] M. Brodie, J. Mylopoulos, J. Schmidt, On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages, Springer–Verlag, New York, NY, 1984.
[18] M. Loomis, Object Databases: The Essentials, Addison–Wesley, Reading, MA, 1995.
[19] P. Chen, Entity–relationships vs. object-orientation, in: G. Pernul, A.M. Tjoa (Eds.), Proc. 11th Int. Conf. on Entity–Relationship Approach, October 1992, pp. 1–2.
[20] R. Cattel (Ed.), Object-Database Standard: ODMG-93, Morgan Kaufmann, San Mateo, CA, 1994.