An Ontological Analysis of the Relationship Construct in Conceptual Modeling

YAIR WAND
University of British Columbia
VEDA C. STOREY
Georgia State University
and
RON WEBER
The University of Queensland

Conceptual models or semantic data models were developed to capture the meaning of an application domain as perceived by someone. Moreover, concepts employed in semantic data models have recently been adopted in object-oriented approaches to systems analysis and design.

To employ conceptual modeling constructs effectively, their meanings have to be defined rigorously. Often, however, rigorous definitions of these constructs are missing. This situation occurs especially in the case of the relationship construct. Empirical evidence shows that use of relationships is often problematical as a way of communicating the meaning of an application domain. For example, users of conceptual modeling methodologies are frequently confused about whether to show an association between things via a relationship, an entity, or an attribute.

Because conceptual models are intended to capture knowledge about a real-world domain, we take the view that the meaning of modeling constructs should be sought in models of reality. Accordingly, we use ontology, which is the branch of philosophy dealing with models of reality, to analyze the meaning of common conceptual modeling constructs.

Our analysis provides a precise definition of several conceptual modeling constructs. Based on our analysis, we derive rules for the use of relationships in entity-relationship conceptual modeling. Moreover, we show how the rules resolve ambiguities that exist in current practice and how they can enrich the capacity of an entity-relationship conceptual model to capture knowledge about an application domain.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*Data*

Authors' addresses: Y. Wand, Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, BC V6T 1Z2, Canada; V. C. Storey, College of Business Administration, Georgia State University, Atlanta, GA 30302-4015; R. Weber, Department of Commerce, The University of Queensland, 4072, Australia.

models; K.6.1 [**Management of Computing and Information Systems**]: Project and People Management—*Systems analysis and design*

General Terms: Design, Theory

Additional Key Words and Phrases: Conceptual modeling, database design, entity-relationship model, object-oriented modeling, ontology, semantic data modeling

1. INTRODUCTION

Conceptual modeling (or semantic modeling) focuses on capturing and representing certain aspects of human perceptions of the real world so that these aspects can be incorporated into an information system. Most conceptual modeling approaches are concerned with "things," which are often referred to as *entities*, and associations among things, often referred to as *relationships* [Brodie 1984]. More recently, these two constructs also feature in object-oriented (OO) approaches. The entity construct is often replaced by the object construct, although both constructs have much in common. The traditional notion of a relationship, however, is used in only some OO approaches [Embley et al. 1992; Rumbaugh et al. 1991]. Nonetheless, as we argue below, it has much in common with the message-passing construct that is ubiquitous in OO modeling.

While both entities and relationships are fundamental to conceptual modeling, relationships prove to be more problematical. To illustrate some of the difficulties, consider the often-cited example of *Marriage* as a real-world fact to be modeled. In the context of the entity-relationship (ER) model, Chen [1976] argues that marriage can be considered as a separate *entity type* or as a *relationship type* between two *Person* entity types. In this light, Hansen and Hansen [1992] chose to model marriage as a relationship type, *is-married-to*, and as an aggregate object (entity) type, *married-couple*, which in turn participates in other relationship types. Date [1995] and Kent [1978] also mention that a relationship type can be conceived as an entity type, and Batini et al. [1992, p. 31] describe relationship types as representing *aggregations* of two or more *entity types*. An aggregation of entity types, however, is itself likely to be perceived as an entity type [Smith and Smith 1977]. Still other examples of multiple representations for relationship types can be found. In the ER model, for example, Elmasri and Navathe [1994] sometimes represent them as *weak entity types* and sometimes as *attributes*. In short, the theory underlying the nature of and representation of relationships in conceptual modeling is unclear.

Empirical evidence also indicates that database designers find relationships more difficult to model than entities. For example:

(1) Prietula and March [1991] studied the *physical* design processes used by expert and novice database designers. They found that failure to specify "file connections" (relationships) was a major problem.

(2) Batra et al. [1990] studied the *logical* design processes used by database designers. They concluded (p. 137) "that the most commonly occurring errors pertain to the connectivity of relationships."

(3) Goldstein and Storey [1990] studied how users employed an automated database design tool. They found that users had difficulty distinguishing attributes from relationships.

In our view, problems arise with relationships in conceptual modeling because their nature and underlying meaning are unclear. For example, usually a relationship is defined simply as an *association* between two or more entities [Batini et al. 1992; Chen 1976; Goldstein 1985; McFadden and Hoffer 1985]. But what, exactly, is meant by an "association"? In an attempt to answer this question, Teorey et al. [1986, p. 200] argue that: "Relationships have semantic meaning, which is indicated by the connectivity between entity occurrences (one to one, one to many, and many to many) and the participation in this connectivity by the member entities may be either optional or mandatory." The cardinalities of a relationship and whether they are optional or mandatory indeed describe some *characteristics* of relationships, but they do *not* describe the underlying *meaning* of relationships. We contend that a clear, unambiguous definition of this meaning is a prerequisite for conceptual modelers to be able to use relationships correctly to represent phenomena in the domains that interest them.

The purpose of this paper, therefore, is to undertake a formal analysis of the *meaning* of the relationship construct and to show the *practical implications* of this analysis for *conceptual modeling* activities. We begin by arguing that the meaning of all conceptual modeling constructs needs to be articulated within the context of a theory of *ontology* (see also Ashenhurst [1996]; Guarino [1995]; and Wimmer and Wimmer [1992]). Ontology is "That branch of philosophy which deals with the order and structure of reality in the broadest sense possible" [Angeles 1981]. Because relationships are constructs that have been devised to model certain types of real-world phenomena, it should be possible, therefore, to derive their meaning via theories of ontology. Having clarified the meaning of relationships, we then seek to provide rules that will assist information system designers to use them in their conceptual modeling activities. Such rules can be the basis for ensuring that different designers will model the same phenomena in the same way. In this regard, note that our focus in this paper is *not* on *data* modeling. Clearly, a single conceptual model might be mapped into different data models, depending, for example, on constraints imposed by the underlying technology platform used to store and manipulate data.

The paper proceeds as follows. Section 2 provides an overview of the ontological model we use to analyze the meaning of relationships. Section 3 shows how this model can be employed to formalize various conceptual modeling constructs that are essential to understanding the nature of and meaning of relationships. Section 4 then derives some conceptual modeling

rules based on the formalization we propose. Section 5 shows how these rules can be applied to the use of the relationship construct in ER conceptual modeling. Finally, Section 6 discusses some conclusions and suggestions for further work.

2. ONTOLOGICAL FOUNDATIONS

To provide the basis for our analysis of the relationship construct, we use an ontological framework developed by Bunge [1977; 1979]. His model articulates a set of high-level, abstract constructs that are intended to be a means of representing all real-world phenomena. Bunge reports [Bunge 1977, Preface] that his aim was to draw upon ample ontological traditions arising from the work of Aristotle, Aquinas, Descartes, Spinoza, Leibniz, Hobbes, Helvetius, d'Holbach, Lotze, Engles, Peirce, Russell, and Whitehead. He further claims his work differs in a "matter of method" because it also draws upon contemporary research and has been elaborated using mathematics (see also Agassi [1990]). Bunge intends his ontology to be "both exact and scientific." We might have attempted to use other ontologies to analyze the meaning of a relationship. We chose Bunge's, however, because we are unaware of any others that are as rigorous and comprehensive. Moreover, we have already found his ontology useful in analyzing other phenomena within the computer science and information systems domains [Paulson and Wand 1992; Wand 1989; Wand and Weber 1989; 1990; 1993; 1995; Wand and Wang 1996; Weber and Zhang 1996].

The specific ontological constructs we require to analyze the nature and meaning of a relationship are *thing, property, attribute, functional schema, state, law, interaction, class, kind,* and *composition of things*. In this light, the following sections provide an overview of these constructs. In the interests of brevity, henceforth we mean Bunge's ontological model when we use the term "ontology." Postulates and definitions taken from or adapted from Bunge's work are indicated by "*".

2.1 Thing

POSTULATE 1*. The world is made of *things* that possess *properties*.

Ontology distinguishes between concrete things, which are called *substantial individuals* or *entities*, and conceptual things (e.g., mathematical concepts such as sets and functions). We assume that any domain can be described in terms of concrete things and the linkages that exist among them. Therefore, we use the word "thing" to refer only to substantial individuals or concrete things. We further assume that domain modeling is based upon someone's view of *existing or possible* reality. Therefore, the notion of a concrete thing applies to anything *perceived* as a *specific object* by someone, whether it exists in physical reality or only in someone's mind. In this light, a bank account is considered a thing, as well as a product that has been designed but not yet produced. Both are concrete things in someone's mind.

2.2 Properties

There are no things without properties (there are no "bare" individuals [Bunge 1977, pp. 36, 58]). Moreover, *properties are always attached to things* (" every property is possessed by some individual or other: there are no properties that fail to be paired to any individuals" [Bunge 1977, p. 62]). Properties of concrete things (substantial individuals) are called *substantial properties*, or simply *properties*. Properties of conceptual things are called *formal properties, attributes*, or *predicates*. It is important to recognize that *not* having a property is *not* a property. For example, "not being red" is not a property.

A property can depend on one or more things. A distinction is made between

- *intrinsic* properties that depend on one thing only; and
- *mutual* or *relational* properties that depend on two or more things.

For example, the height of a person is an intrinsic property because it depends only on the existence of the person. The property of being a university student is a mutual property, however, because it depends upon the existence of *both* a person and a tertiary institution.

Properties themselves can *not* have properties [Bunge 1977, pp. 98–99]. For example, at first glance, it might seem that the weight of a person itself has a property associated with the time at which the weight was measured. The "real" meaning here, however, is that the person has a variable weight (the property is not just *weight* but *weight at time t*). The possibility of properties having properties is only contemplated when we have not fully specified (or properly understood) a property in the first place.

2.3 Attributes

The properties of a thing exist, whether or not humans are aware of them. Humans conceive of things, however, in terms of *models* of things. Such models are *conceptual things*. Recall that the properties of conceptual things are termed *attributes*. Attributes are characteristics assigned to (models of) things according to human perceptions. Depending upon circumstances, humans may use different models of the same thing, and therefore assign different sets of attributes to the same thing. Thus, humans conceive of properties of things in terms of the *attributes* of their conceptual models, and properties are known to humans only as attributes. An attribute, however, may or may not reflect a substantial property (or properties). Moreover, in a given model, not every property will be represented as an attribute.

To illustrate the notions of property and attribute, consider the following examples:

- The *height* of a person is an attribute that reflects a *substantial property*.

- The *name* of a person does not represent any *specific* substantial property. It is an attribute that stands for the individual as a whole (assuming that the name is unique).

- The I.Q. of a person is an attribute representing many properties that are not known explicitly.

- Having a student number in a given university represents the *mutual property* of being a student at that university.

Attributes can be formalized in a *predicate form* or in a *functional form*. The predicate (propositional) form for a general property is

$$A : T_1 \times \ldots \times T_n \times V_1 \times \ldots \times V_m \rightarrow \text{Statement regarding } A \quad (1)$$

where $T_k (k = 1, \ldots, n)$ represents a set of things and $V_j (j = 1, \ldots, m)$ represents a set of *values*. It will usually be assumed that a set T_i contains the same "type" of thing (that is, things perceived similar in some respect).

For example, the property "a person works for a company" can be represented as $W : T_1 \times T_2 \times D \rightarrow P$, where T_1 is a set of people, T_2 is a set of companies, D is a set of dates, and P is a set of statements of the form: "p (from the set T_1) works for c (from the set T_2) at d (from the set D)."

Based on this form, a *specific* property (of a particular individual, t_1) can be represented as an attribute predicate of the form:

$$A(t_1, \ldots, t_n, v_1, \ldots, v_m) \text{ where } t_i \in T_i, v_j \in V_j. \quad (2)$$

For example; *works-for* (p, c, d) means person p, works for company c, at date d.

The following ontological premise ensures that every property, in principle, can be modeled as follows:

POSTULATE 2*. Every property in general can be represented by a propositional (attribute) function; $A : T_1 \times \ldots \times T_n \times V_1 \times \ldots \times V_m \rightarrow$ Statement regarding A; and every specific property can be represented as an attribute function of the form; $A(t_1, \ldots, t_n, v_1, \ldots, v_m)$ where $t_i \in T_i, v_j \in V_j$.

Note: The propositional form of attributes can be converted to a functional form in which an attribute can be assigned values (we allow an attribute to be multivalued). Below, we use the functional form of attributes.

The examples in Table I illustrate the notion of an attribute function and how it represents properties. Note that sometimes properties that appear to be intrinsic are actually mutual. For example, at first glance, being an employee appears to be an intrinsic property of a person. It is a mutual

Table I. Properties Represented By Attribute Functions

Property	Attribute (representing the property in general)	Value of Attribute (representing the specific property of an individual)
Height	A : persons $\times$ U $\times$ P (where U is a set of units of measure and P is the set of nonnegative numbers)	"John's height is 1.70 metres" Height ('John','metre',1.70)
Being an employee	A : persons $\times$ companies	"Mary works-for company X" Works-for('Mary','X')
A supplier supplies a part	A : parts $\times$ suppliers	"Supplier X supplies part P" Supplies('P','X')
A supplier supplies a part at a certain price	A : parts $\times$ suppliers $\times$ C $\times$ P (where C is a set of currencies and P is the set of nonnegative numbers with two decimals)	"Supplier X supplies part P at a price of \$50.00" Price('P','X','\$',50.00)

property, however, because it depends upon the existence of both a person and an employer.

2.4 Class and Kind

The concept of a *class* is now widely used within the conceptual modeling literature. It is important because we usually conceive of a thing as an instance of a certain *type* of thing. Thus, we formalize the concept of a class and the related notion of a *kind*:

Definition 1.* The *scope* of a property is the set of things that possess the property. That is, if Θ is the set of all things and P is the set of all properties, the scope function $\mathbf{S}$ is the mapping $\mathbf{S} : P \rightarrow 2^{\Theta}$.

Definition 2.* A subset of things is called a *class* if and only if a property exists such that the subset is the scope of that property. That is, a subset X of the set of things Θ is called a *class* of things iff $\exists p \in P$ such that $X = \mathbf{S}(p) \in 2^{\Theta}$.

Definition 3.* A subset of things X is a *subclass* of another set of things Y if and only if X is a proper subset of Y. Conversely, Y is a *superclass* of X.

Corollary 3. If $\mathbf{S}(p_1) = X$ and $\mathbf{S}(p_2) = Y$, $p_1, p_2 \in P$, then X is a subclass of Y if and only if $\mathbf{S}(p_1) \subset \mathbf{S}(p_2)$.

Note that if a set of things is a subclass of another set of things, all things in the former set must also possess the property possessed by all things in the latter set. Indeed, it is common to define subclasses by first identifying an additional property that some members of a class possess. The subclass is then defined in terms of the scope of the *conjunction* of the property used initially to define the class and the additional property of interest. In short, subclasses are formed by "adding" properties to the set of properties possessed by things in a class. Conversely, superclasses are formed by

"subtracting" properties from the set of properties possessed by things in a subclass.

*Definition 4**. Let **R** be a set of properties. An **R**-kind is the intersection of all scopes of properties in **R**.

If the set **R** is finite $\mathbf{R} = \{p_1, \ldots, p_n\}$, we can define a *compound property*–one having all properties in **R**. In that case, an **R**-kind is a class.

Finally, when a kind contains only things whose combination of properties abide by "laws," it is termed a *natural kind*. Things in a natural kind can only assume states that are "allowed" by a set of laws. (Note that we define the notion of "law" formally, below.)

In short, a class is defined by a set of things possessing a common property, a kind is defined by a set of properties, and a natural kind is defined by a set of lawfully related properties.

2.5 Functional Schema, State, and Law

Similar things can be represented by the same model. One specific model of things is a *functional schema*. In a functional schema, the attributes of the conceptual model of a thing are represented in a *functional* rather than a *propositional* form. A functional schema, therefore, is a finite sequence of attribute functions defined on a certain domain:

*Definition 5**. Let T be a set of things all possessing a common set of properties. A *functional schema* $X_m = < M, \mathbf{F} >$ is a nonempty set M, and a finite sequence $\mathbf{F} = < F_1, \ldots, F_n >$ of functions defined on M; that is, $F_i : M \rightarrow V_i$ where V_i is a domain of values, and each (attribute) function represents a property of the things.

Several aspects of this definition need clarification. First, in a functional schema, attributes are represented by nonpropositional functions and not predicates. Bunge proposes this alternative form because it is more common in the sciences; it is more "economical." In particular, it is a convenient form for defining the notion of *state* (see below). Second, the domain M reflects any conditions under which the value of the function F_i is "observed." Often, M is a set of time instances. It may , however, include other variables such as who observes the thing. In essence, the existence of M recognizes that all real-world observations are relative. Third, the functions F_i are evaluated for each thing in the set T (i.e., the set of things all possess the same properties for which the functional schema is defined).

For example, assume the set of things T is the set of employees in an organization. A function F_1 represents the salary of employees in dollars. For a particular employee, it maps time instants $t_i \in M$ into, say, the set of positive integers. Thus, the function shows an employee's salary level at each point in time. Other functions F_i can be used to represent other attributes of the employee (e.g., educational qualifications and prior work experience).

Note, also, that a functional schema is based on a representation of a *partial* set of properties. Hence, a functional schema can be conceived as a *model* of a class or kind. Because a functional schema is just a model created for a certain purpose, the properties represented in a functional schema depend upon the circumstances and purpose of modeling the things. For example, a person may be viewed as an employee, a customer, or a taxpayer. Similarly, a university student may be viewed differently by the department responsible for academic records and the department responsible for tuition fee payments. Thus, a functional schema can be interpreted as a formalization of a *view* of a set of similar things [Takagaki and Wand 1991].

The idea of a functional schema also allows us to define the notion of the *state* of a thing. The state concept is useful for reasoning about how things may change.

Definition 6.* Consider a thing X described by the functional schema $<M, \mathbf{F}>$. The function $F_i : M \rightarrow V_i$ is termed the ith *state function* of the thing. The set of values $F(t) = <F_1(t), \ldots, F_n(t)>$ at a certain point $t \in M$, is said to represent the *state* of X at t.

The definition of state allows any combination of the values of the functions $F_i(t)$ to be a possible state. Not all combinations of the values of $\{V_i, i = 1, \ldots, n\}$ might occur, however. Restrictions on the possible combinations are termed *laws*:

Definition 7.* Any restriction on the possible values of the components of a functional schema of a thing or their combinations is termed a *law*.

For example, if the functional schema of an employee contains rank and salary as state variables, a law may constrain the allowed combinations of rank and salary.

Finally, these constructs allow us to address two fundamental questions that have implications for conceptual modeling: Can two things possess exactly the same properties? Does a thing remain the "same" when it acquires or loses properties? These questions are addressed by the following two ontological principles:

Principle 1.* No two things possess exactly the same set of specific properties.

Principle 2.* (Nominal invariance): "A thing, if named, shall keep its name throughout its history as long as the latter does not include changes in natural kind–changes which call for changes of name" [Bunge 1977, p. 221].

The former principle implies that *keys* are *unnecessary* attributes in conceptual modeling. Indeed, it makes clear that keys are simply implementation mechanisms. Thus, key attributes have no informative role in a functional schema nor in a conceptual model of a domain.

The latter principle implies that a thing can change properties and still be considered the same thing. It will cease being considered the same only when the laws that specify its possible states undergo change. The acquisition or loss of a property has implications for how a thing is "classed," however. Moreover, it has implications for how we deal with optional properties. We return to this matter below.

2.6 Interaction

Things can interact. When two things interact, one may cause the other to change. Changes to things are manifested as changes to properties, which will be modeled via changes in the values of attribute functions–namely, changes of state.

Definition 8.* Thing X *acts on* thing Y if and only if the states that Y traverses for a given subset of M when X is present are different from the states that Y would traverse if the thing X did not exist. Things X and Y *interact* if at least one acts on the other.

For example, consider specific instances of an inventory item and a customer. Assume the current state of the item manifests that it is in stock. If the customer decides to purchase the item, its availability (and hence its state) will change as a function of time. According to our definition, the customer and the inventory item interact because the states that the inventory item traverses through time are affected by the existence of the customer.

A close link exists between the notion of interaction and the notion of a mutual property. In this regard, consider two types of things, T_1 and T_2, such that a thing of type T_1 can interact with a thing of type T_2. The existence of interaction is a mutual property of things in T_1 and in T_2. Hence, it can be described in terms of an attribute predicate:

$$A_I : T_1 \times T_2 \rightarrow \text{Statement that an interaction exists}$$

If t_1 and t_2 interact, this form implies that the predicate $A_I(t_1, t_2)$ is "true."

Conversely, the existence of a mutual property can indicate an interaction. Consider an example from Table I. The (mutual) property that a person is employed by a company implies that the existence of the company affects the state of the person (and vice versa). For example, if the company ceases to exist, the person is no longer employed. Similarly, if the employee quits, the (set-valued) attribute of the company that shows its list of employees will change in value. A mutual property that reflects interaction is termed a *binding* mutual property [Bunge 1977, p. 102]. A binding property implies that some changes in one thing are related to (precede, are accompanied by, or are followed by) changes in the other thing. This is true in the example "a person works-for a company." A property that does not imply interaction is termed *nonbinding*. For example, "object a is behind object b."

2.7 Composition of Things

Composition is a fundamental ontological concept which addresses the notion that a thing is *made-of* other things; that is, it has *parts*. Each of these things, in turn, is a *component-of* or *part-of* the composite thing:

POSTULATE 3*. Two things may associate to form another thing.

Based on this postulate, a thing is a *composite* if and only if at least two concrete things combine to form it; otherwise, it is *simple*. Ontology also postulates that simple things exist.

The reason we assemble things into composite things is that we are interested in some property of the composite. In other words, the composite has at least one *emergent* property that interests us; that is, a property not possessed by any of its component parts:

*Definition 9**. A property of a composite thing is *inherited* if and only if it is a property of any of its components; otherwise, it is *emergent*.

To illustrate the distinction between an inherited and an emergent property, consider a computer. A computer is a composite thing because it is made up of a central processor, main memory, etc. The memory size of the computer is an inherited property because it is a property of the memory component. In contrast, the *processing power* of a computer is an emergent property because it cannot be attached to any individual component. Processing power can be evaluated only for the whole system, even though it depends on the properties of individual components.

Ontology postulates that humans view an aggregation of things as a composite thing only if they are interested in at least one emergent (holistic) characteristic of the composite:

POSTULATE 4*. Every composite thing possesses emergent properties.

For example, every composite will have emergent properties like its number of components and, in the case of systems, the structure of its components and its overall history (which will not be equivalent to the histories of its components). Often, also, it is the relationship of the whole (rather than the individual components) to some other thing that is of interest to stakeholders–for example, the relationship of a university (a system of interacting components) with its funding agencies.

In ontology, the part-of construct is a type of binding mutual relationship. The history of the composite (as manifested through one or more of its emergent properties) depends on one or more of its components. For example, the death of a family member (a component of the family) may affect its taxation status (an emergent property of the family composite). The death of a family member also affects other emergent properties of the composite such as the number of family members and the average age of family members.

A number of ontological researchers (e.g., Simons [1987] ; Varzi [1994]) have attempted to clearly distinguish between *connection* relationships

(e.g., a husband and wife interact) and *part-of* relationships (e.g., the husband and wife are parts of a family). The former relationships are defined as *topological* relationships; the latter as *mereological* relationships. In ontology, however, both topological and mereological relationships are types of mutual properties. Moreover, in conceptual modeling, we are usually interested in those types of topological and mereological relationships that are *binding* mutual properties.

Note that our notion of aggregation is *not* one of "assembling" attribute "objects" to form a "composite" object (see, e.g., Elmasri and Navathe [1994, pp. 636–637]). For a start, in our ontology, attributes are *not* objects (see Rule 1 in Section 4). Rather, they are proxies for (unknowable) properties. We model objects/things via a functional schema, which is the set of attributes we choose to describe the object/thing. In our ontology, aggregation pertains only to building composite things using the part-of (mereological) relationship.

Finally, the notion of composite might be extended further by indicating mutual properties among its components. In particular, if components have a mutual attribute that indicates some precedence among them, the components can be viewed as forming a sequence.

3. FORMALIZATION OF CONCEPTUAL MODELING CONSTRUCTS

In this section we propose mappings from the set of ontological constructs discussed in Section 2 into well-known conceptual modeling constructs. Our goal is threefold. First, we seek to show that ontological constructs often have a representation in familiar modeling constructs. Second, the mapping enables us to assign a precise meaning to these constructs. Third, by providing an ontological meaning to well-known modeling constructs, we seek to lay the foundation for resolving the difficulties we described earlier pertaining to the modeling of relationships.

Table II summarizes our proposed mappings from the ontological constructs discussed in Section 2 to commonly used constructs in *data-oriented* and *OO* conceptual modeling. Table II also includes proposed generic terms for conceptual modeling constructs. Note that we are *not* implying that users of conceptual modeling constructs always comply with our proposed mappings. Indeed, the conceptual modeling literature is replete with examples where the mappings are violated; a matter we take up further in Sections 4 and 5. We argue that these violations are the primary source of semantic ambiguity in conceptual modeling work. They reflect that a singular ontological meaning of a conceptual modeling construct has not been preserved, and thus the interpretation of the construct is no longer clear.

3.1 Things

We begin by positing that the fundamental construct needed to model a domain is a representation of a thing. Identifying the things in a domain is a prerequisite to our eliciting the remaining semantics–for example, the

Table II. Mapping Ontological Constructs to Conceptual Modeling Constructs

Ontological Construct	Commonly Used Construct	Proposed Generic Conceptual Modeling Construct
Thing	Entity, Object	Instance
Property	No direct representation	No direct representation
Attribute representing an intrinsic property	Attribute (of an entity or an object)	Attribute (type: intrinsic)
Attribute representing a mutual property	Relationship (binary or n-ary) Reference attribute	Attribute (type: mutual)
Interaction attribute representing a binding mutual property	Relationship Reference attribute Message connection Service request	Attribute (type: mutual, binding)
Class	Entity type, object class	Class
Kind	Entity type, object class	Class
Natural kind	Object type	Class
Simple thing	Entity, object	Instance (type: simple)
Composite thing	Aggregate entity or object	Instance (type: composite)
Connection attribute representing a binding mutual property	Relationship	Attribute (type: mutual, binding, topological)
Component-of attribute representing a binding mutual property	Relationship Part-of	Attribute (type: mutual, binding, mereological)

attributes that are important, the relationships that exist among things, and the classes used to generate a view of the domain.

The *generic* conceptual modeling construct used to represent a thing is an *instance*. Depending upon the modeling approach used, an instance in turn is represented as an *entity* or *object*:

Premise 1. The primary focus when modeling a domain is the things that exist in the domain.

Definition 10. An instance in a conceptual model is a representation of a thing in the ontological model.

The notion of using objects or entities as *modeling constructs* to represent instances of things is widely held. For example, Coad and Yourdon [1991, p. 53] argue that an object is "An abstraction of something in the problem domain." Similarly, Chen [1976, p. 10] argues that "An *entity* is a 'thing' which can be distinctly identified."

3.2 Properties and Attributes

In the ontological model, recall that humans are deemed to be aware of properties of things only via attributes. Thus, conceptual models *never*

represent properties directly. Rather, they represent only attributes. Because attributes represent either intrinsic or mutual (relational) properties, however, we distinguish between two situations in conceptual models:

Definition 11. An *attribute* in a conceptual model is a representation of an *attribute* in the ontological model, which in turn represents an *intrinsic property* (one or more) of a thing in the real world. A *relationship* in a conceptual model is a representation of an *attribute* in the ontological model, which in turn represents a *mutual property* (one or more) of a thing in the real world.

In *data-oriented* conceptual models, relationships are shown in two ways: with no special symbol, but simply by a line connecting two entities; or explicitly by a special symbol (e.g., a diamond) to which the related entities are connected. In OO conceptual models, these representations are also used. In addition, relationships in OO conceptual models are sometimes shown via *message connections* (with or without the use of a special construct) [Rumbaugh et al. 1991; Coad and Yourdon 1991]. In all cases, the *degree* of a relationship can be determined by the number of lines connecting the things or objects that participate in the relationship.

At first glance, message connections in OO models may appear to be a different construct from the relationships shown in data-oriented conceptual models. The ontological model indicates, however, that they represent the same construct. From an ontological perspective, a message connection implies that one object can affect the state of the other; that is, the two objects interact. The attributes of the objects that change directly as a result of the message connection are mutual attributes. In ontology, however, recall that interaction is manifested as a *binding mutual property*. Because a relationship represents any mutual property, it is a generalization that represents either a binding or a nonbinding mutual property. In OO conceptual models, therefore, use of the message construct conveys additional information about the nature of the mutual property, namely, that it is binding.

3.3 Class and Kind

We assume that all kinds are defined by a finite set of properties. Hence, they can be viewed as classes. A class is defined via a set of properties (a compound property) that a set of things possess in common. In the ontological model, properties in turn are represented by attribute functions in the functional schema. Hence, a functional schema can be conceived as a model of things that are alike in some way; that is, a model of a class (or kind):

Definition 12. A class (or type) in a conceptual model is a representation of a class or kind in the ontological model.

In the ontological model, recall that attribute functions in a functional schema represent either intrinsic properties or mutual properties. In a

conceptual model, therefore, a class is defined in terms of both attributes and relationships (and message connections for OO models). Note that the *generic* conceptual modeling construct is a class. This generic construct in turn is represented in specific conceptual modeling methodologies as either entity types or object classes and types.

In conceptual modeling, subclasses are often defined via the process of *specialization* (the IS-A concept). Similarly, superclasses are often defined via the process of *generalization*. In the context of the ontological model, subclasses and superclasses are simply classes or kinds. Moreover, given our property-based view of classes, specialization and generalization can be conceived as processes that lead to the addition of properties to or subtraction of properties from the set of properties possessed by other classes (see Parsons and Wand [1997]).

Definition 13. Specialization is the process of forming one class from another class by adding properties to the set of properties possessed by the latter class. Conversely, generalization is the process of forming one class from two or more other classes by identifying properties that are common to these latter classes (subtracting properties that are not common to them).

3.4 Laws and Natural Kinds

Recall that laws in the ontological model constrain the states that can be assumed by things. The generic conceptual modeling construct used to represent a law is a constraint:

Definition 14. A constraint in a conceptual model is a representation of a law in the ontological model.

In data-oriented conceptual models, constraints are represented by *integrity* constraints. In OO models, *methods* that control object states are also a representation of constraints. In the ontological model, when the instances of a kind are subject to laws, the kind is termed a natural kind. Hence, class definitions that include integrity constraints and object-type definitions that include methods (or operations) can be viewed as representing *natural kinds*.

While entities and objects represent individual things, note that classes, kinds, and types represent *views* of things, and usually apply to many things. Because things exist independently of the views that are assigned to them, we argue that entities and objects also should be conceived independently of classes or types. In this light, the term "object" should not be used as a descriptor for a class, kind, or type [Shlaer and Mellor 1988, p. 11]).

3.5 Composites and the Part-of Relationship

Ontologically, both a composite and its components are things. The former is a complex thing; the latter are simple or complex things. Recall that the composite and its components are related via a binding mutual property. In

conceptual models, a relationship construct is often used to show that the composite and its components are related (e.g., Elmasri and Navathe [1994, p. 67]). In ontology, the part-of (mereological) relationship is a type of binding mutual attribute. Recall, it is distinguished from a connection (topological) relationship, which is also a type of binding mutual attribute.

4. ONTOLOGY-BASED CONCEPTUAL MODELING RULES

In this section we propose a set of conceptual modeling rules based on Bunge's [1977; 1979] ontological theory and our proposed mappings from his ontological constructs to conceptual modeling constructs. These rules are not tied to a specific conceptual modeling approach. Rather, they are applicable to any approach that uses the generic constructs listed in the right-hand column of Table II—for example, ER modeling, object-role modeling (ORM, or its precursor, NIAM), and OO modeling. We underscore again that our focus is on conceptual or domain modeling, and not data modeling (in the sense of database design). Also, the mappings we propose include more concepts than are usually available within any one modeling method.

The motivation for our rules is that we seek to provide a precise meaning for some important conceptual modeling constructs. In particular, we seek to reduce *semantic ambiguity* by avoiding (a) *construct overload*, whereby one modeling construct stands for two or more ontological constructs, and (b) *construct redundancy*, whereby more than one modeling construct can be used to represent a single ontological construct (see Wand and Weber [1993]). Whenever instances of construct overload or construct redundancy occur, we argue that a *prima facie* case exists for semantic ambiguity to arise. Stakeholders' understanding of the domain being modeled is likely to be undermined because a one-one mapping does not exist between ontological constructs and modeling constructs.

In this regard, several of our rules provide *theoretical* support for some existing modeling practices by showing why semantic ambiguity should not arise. These rules are likely to be accepted, therefore, at face value. Other rules are counterintuitive, however, and contravene widespread conceptual modeling practices. We see this outcome as a strength of our approach because it motivates us to be circumspect about some existing, widely accepted conceptual modeling practices. In some cases we now have research outcomes (which we reference below) to show that our rules do indeed reduce semantic ambiguity in conceptual modeling. In other cases, however, the research needed to evaluate the merits of our rules has yet to be conducted. Accordingly, we see our rules as *propositions* to be tested empirically.

Based on our ontological theory, then, we propose *seven* rules as foundations for conceptual modeling practice:

RULE 1. Things are represented only as instances. Instances should represent only things.

Rationale: Because things are fundamental constructs, they should be represented by special constructs that do not represent any other ontological concept.

Practical implications: The notion of an entity or object should be used *only to represent things*. In the ontological model, things can be linked only by mutual properties (usually reflecting interactions), *not* by other things. Because mutual properties are not things, they should *not* be represented by constructs that represent things. Thus, Rule 1 *proscribes* the practice of using entities or objects to connect other entities or objects. Interestingly, widely-read literature on conceptual modeling often recommends that entities or objects be used to model constructs other than things; for example, the use of "event remembered" to connect two object classes where the relationships is of cardinality $m : n$ [Coad and Yourdon 1991, p. 131] and the notion of "objectified relationships," which involve "treating a relationship between objects as an object itself" [Halpin 1995, p. 65].

Similarly, in ontology, an event is a change of state of a thing, not a thing. Therefore, rule 1 also proscribes modeling events as entities or objects. Again, widely-read literature on conceptual modeling often shows events modeled as entities or objects. For example, the authors of OSA [Embley et al. 1992, p. 68] indicate "we may model any event as an object." Again, we note that this implication applies to conceptual (domain) modeling, not to database design. From a database design viewpoint, an event can be represented as a record modeled as an entity in an ER diagram or a data object in an OO diagram.

Rule 1 also proscribes modeling properties (attributes) as things (entities, objects). Thus, this rule runs counter to modeling approaches like NIAM that are founded on the contention that a distinction should not be made between properties and things (NIAM models both attributes and entities as objects) [Halpin 1995]. Based on the ontological principles we have articulated here, however, Weber [1996] provides psychological theory and compelling evidence in support of humans making a distinction between things and their properties and the need to maintain this distinction in conceptual models that are built to describe a domain.

Nor does Rule 1 allow "values" to be modeled as things or objects (e.g., see Elmasri and Navathe [1994, pp. 667–671]). Values are elements of the codomains of attribute functions. They cannot exist independently in the world. Instead, they must be conceived in terms of things that have properties that in turn are represented as values of attribute functions.

RULE 2. Both simple and composite things should be represented using the same construct (entity, object).

Rationale: This rule is a corollary of Rule 1: both simple things and composite things are things. Hence, they should be modeled in the same way.

Practical implications: Given Rule 2, whether we use an entity/object or a relationship to represent a composite thing should no longer be an issue in

conceptual modeling. Rule 2 proscribes the use of relationships to represent composites. Note, however, that we are not proscribing the use of different *symbols* to distinguish simple things from composite things. For example, to avoid construct overload, we may want to annotate the symbol we use for things, to differentiate between simple things and composite things. Alternatively, we might use the same symbol to represent a simple thing and a composite thing, and rely on the existence of a part-of relationship symbol to signal when a thing is simple or composite.

RULE 3. A class or a kind of thing is defined in terms of a given set of attributes *and* relationships; that is, intrinsic attributes and mutual attributes.

Rationale: A class is represented by a functional schema. A functional schema is defined in terms of a set of intrinsic attributes and mutual attributes that are used to represent both intrinsic and mutual properties.

Practical implications:

(1) A class/type definition should include relationships or message connections, as appropriate.

(2) The same thing can be viewed in terms of *different* functional schemata (sets of intrinsic and mutual attributes/relationships). Each view should be represented as a class or type, defined in terms of attributes and relationships. Thus, at any time, a specific thing can be an *instance of several classes* that do not have to be related by inheritance.

(3) Assume a thing has "acquired" a property (for example, a person has become an employee of a company). If this property is relevant for modeling purposes, Rule 3 requires that the thing now be modeled via a *new* functional schema. This schema is represented by another entity type of object class. Thus, the "acquisition" (or loss) of a property signals that the thing has become (or ceased to be) an instance of a class.

(4) Because a class is defined via a set of attributes and relationships (mutual attributes), a class definition cannot include optional properties.

RULE 4. An aggregate type/class must have properties in addition to those of its component types/classes.

Rationale: A composite thing must possess emergent properties; that is, properties not possessed by any of its components.

Practical implications: A composite thing must be shown with attributes or relationships not possessed by any of its components. Otherwise, it is not a composite thing or else the model is missing important information. Recall that relationships represent mutual properties, and mutual properties may represent interactions. It is sufficient, therefore, for a composite thing to be involved in a relationship in which none of its components partake. For example, an organization may be entitled to government

support, but none of the individuals in the organization may be entitled to the support.

RULE 5. All attributes and relationships in a class represent properties of things in the class.

Rationale: Class definitions in ontology are based upon properties of individual things; that is, substantial properties.

Practical implications: Identification attributes are *not* part of a conceptual model. At first glance, this implication is counterintuitive, as "name" and "key" are usually the first attributes of a thing to be identified. Rule 5 does not preclude the use of identifying attributes in system design. Rather, it indicates that they have no significance in modeling the world. This observation is congruent with Rumbaugh et al.'s [1991, p. 2] argument: "In the real world an object simply exists, but within a programming language each object has a unique *handle* by which it can be uniquely referenced." In essence, use of identification attributes reflects the ontological premise that every thing is unique (no two things possess the same set of properties).

RULE 6. "Null" attributes have no meaning.

Rationale: Not having a property is *not* a property.

Practical implications: Rule 6 requires that all attributes included in a model, whether representing intrinsic or mutual properties, have values for all possible instances. In particular, it implies that a relationship should be instantiated for all instances of a class. In short, intrinsic attributes and relationships should *not* be optional but mandatory (see also Weber and Zhang [1996]). Rule 6 negates the common practice of showing attributes and relationships as either optional or mandatory (e.g., Teorey et al. [1986]). In other words, "null" values must not exist when we model a domain. Again, these implications apply to conceptual modeling rather than to database design. The same database design may result, irrespective of whether we use optional attributes, optional relationships, or subclassification in our conceptual model. We contend, however, that the approach we use in our conceptual model will impact the clarity of the meaning we convey to stakeholders, including database designers.

RULE 7. The same construct should be used to represent a binary relationship and a higher-order relationship.

Rationale: Both binary and higher-order relationships reflect attributes representing mutual properties.

Practical implications: Rule 7 proscribes the practice of using different constructs to represent binary and higher-order relationships (e.g., Embley et al. [1992]; Rumbaugh et al. [1991]). Again, note that we are not proscribing the use of different *symbols* to distinguish binary relationships from higher-order relationships. Instead, we only require that designers recognize that binary and higher-order relationships are types of the same construct–namely, a mutual property/attribute. In terms of a conceptual

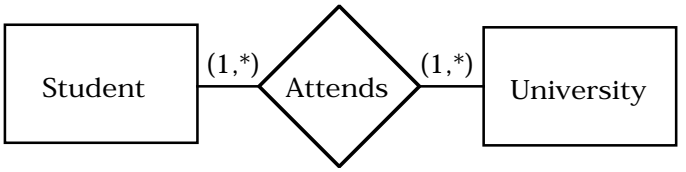

Fig. 1. Student attends a university.

modeling diagram, the same symbol can often be used to represent binary and higher-order relationships, since the number of lines linking the relationship symbol with the entity symbols, representing the entities involved in the relationship, indicates the degree of the relationship.

5. APPLYING THE RULES TO MODEL RELATIONSHIP TYPES IN ER MODELING

In this section we use the ontological foundations and conceptual modeling rules we have articulated above to derive additional rules for modeling relationship types in ER modeling. Our purpose is to illustrate the power of the rules and to show how they can be used to identify the strengths and weaknesses of practices employed to represent relationships in a widely used conceptual modeling approach.

5.1 Relationship Types With Attributes

Consider the following two statements about a university domain:

A student attends a university.
A student attends a university from a given date.

How should these two statements be represented in the ER model?

Figure 1 shows how we believe the first statement will be represented. This representation complies with our rules. It shows that things of class student possess a mutual property with things of class university–namely, an "attends" mutual property. The possible states of student things and the possible states of university things are not independent of each other. Hence, the existence of a mutual property.

Figure 2 shows how we believe the second statement will be represented. But this representation is problematical because its interpretation is ambiguous. Specifically, *start date* is a mutual property of the *student* and *university* things. In the ER model, therefore, it should be shown using a *relationship* symbol if construct overload is to be avoided. Instead, it is shown using the *intrinsic attribute* symbol. How then should the ER diagram be interpreted? [Weber 1997, pp. 112–114]. Is *start date* an attribute of the mutual property *attends*? Under our ontological model, recall that properties themselves cannot have properties. Thus the representation violates our ontology. Moreover, if mutual attributes are to be given attributes, then consistency dictates we should also be able to attach attributes to *intrinsic* attributes—a representation we believe that many, if not most, conceptual modelers who employ the ER model would reject.

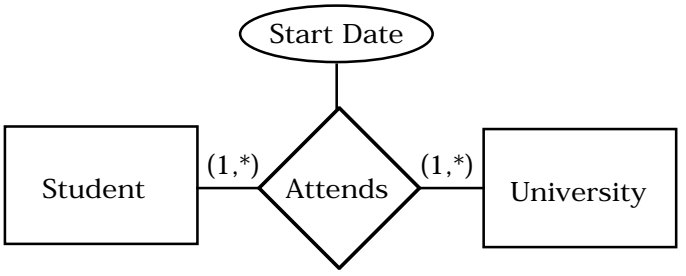

Fig. 2. Student attends a university from a given date.

Figure 3 shows how the statement should be represented in the ER model if our rules are applied. Relative to the representation in Figure 2, at first glance the representation in Figure 3 seems to have two disadvantages. First, visually it appears to be more complex. Second, the cardinalities that should be attached to the *start date* relationship type are unclear if one tries to think of the relationship type as a "linking" construct—like, say, the relationship type *attends* in Figure 1, where the values of this relationship type are typically conceived of as tuples whose elements are some kind of identifiers for the student and university entities.

We argue, however, that conceiving of relationship types as a *link* between two entities reflects a *design and implementation view* (often that of relations and normalization) rather than a *conceptual modeling view* of a domain. Recall from our ontology that mutual properties do not exist independently of the things that possess them. From the perspective of conceptual modeling, therefore, *they must be conceived of as properties of the things that possess them*. From the perspective of the university entity, therefore, the *start date* mutual property is a multivalued property (presumably a student can start at multiple times during a year). From the perspective of the student entity, the *start date* mutual property is also a multivalued property (presumably a student may have multiple start dates if she or he has attended multiple universities). The cardinalities shown in Figure 3 for the *start date* relationship type, therefore, see this relationship as a multivalued property of two things.

We believe that the design and implementation perspective of relationships as links between two entities (which can be implemented as connecting relations) has been the primary reason for modeling relationship types in the ER model, as in Figure 2. From a normalization perspective, for example, the start date mutual property poses problems because its values are not the identifiers of the *student* and *university* entities. The "easy" solution, therefore, is to see *start date* as an attribute of a relationship type that does have the identifiers of the *student* and *university* entities as values, namely, the *attends* relationship type. The normalized relation derived from this relationship type then comprises the triple (*student ID*, *university ID*, and *start date*). The links between relations that are the implementations of entity types are thereby attained. From a conceptual modeling perspective, however, our ontological model predicts that this

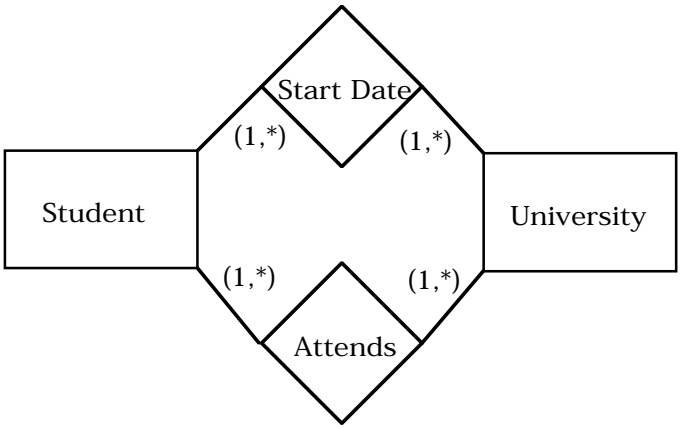

Fig. 3. Alternative representation of student attends a university from a given date.

representation might result in a loss of information about the modeled domain. While this prediction may seem counterintuitive, Burton-Jones and Weber [1999] have experimental evidence to show that users of an ER conceptual model who received the representation of mutual attributes, shown in Figure 3, were better able to undertake problem-solving in a domain than users who received the representation of mutual attributes shown in Figure 2.

5.2 Binary and Higher-Order Relationship Types

In the ER model, the relationship type construct is usually shown using one of two approaches: (a) a line connecting related entity types, or (b) a special symbol (e.g., a diamond) that is connected to related entity types via lines. Note that both approaches to representation should not be used in the same ER diagram if construct redundancy, and therefore semantic ambiguity, is to be avoided.

For two reasons, our ontology leads us to proscribe the representation of relationship types via a line with no special mediating symbol between related entity types. First, mutual attributes whose values are not identifiers for the related entities cannot usually be represented in this way. For instance, the *start date* mutual attribute of our example in Section 5.1 cannot be shown with just a line connecting the *student* and *university* entities. In practice, one approach to try to overcome this difficulty is to employ a "connecting entity type" that can possess attributes (similar to using a relationship type with attributes, as in Figure 2). Our ontological analysis proscribes this practice, however, as entity types should only be used to represent classes of things.

Second, higher-order (ternary and above) mutual attributes cannot be represented directly. In practice, one approach to overcome this difficulty is to employ several binary constructs to represent a higher-order mutual attribute [Rob and Coronel 1997, p. 205]. Again, our ontological analysis proscribes this practice. A mutual attribute can be understood only in

terms of *all* involved things. Thus, it is *not ontologically equivalent* to a set of binary attributes, each depending on two things only. Indeed, using binary attributes to represent higher-order mutual attributes may result in spurious high-order links; that is, links that do not exist. In other words, the information about which links truly exist is lost (this phenomenon is known as a *connection trap* [Elmasri and Navathe 1989, p. 48]).

Our ontology highlights still another way in which some current practices for modeling relationships in the ER model are deficient. Specifically, binary mutual attributes are sometimes represented by making one entity a "property" of other entities. For example, *employer* might be included as an attribute of *employee*. Because being employed is a mutual attribute and not an intrinsic attribute, it should be represented by a relationship construct and not an intrinsic attribute construct. We believe that representing mutual attributes via intrinsic attributes is a carryover from database design considerations. Specifically, mutual attributes are often represented as foreign keys in each of the relations that represent the entity types that possess the mutual property. Design considerations, however, should not affect conceptual models. Indeed, Embley et al. [1995] argue that considering design features in models used for analysis "can corrupt the analysis process." Recall that keys have no meaning in our ontological model (Rule 5), and representing a mutual property via a foreign key attribute in a conceptual model is thus vacuous in our model. In short, *all* mutual attributes should be represented as relationships in the ER model, not just some.

5.3 Mandatory and Optional Relationships

Things of certain types *may* interact. For example:

A customer might buy a certain product.
A student might take a given course.

To reflect this type of situation, ER models often employ *optional* relationship types to indicate that instances of a class *may* form a relationship with instances of another class or the same class. Our rules proscribe this practice, however (Rule 6). To see why, consider first the case where every thing from one class interacts with a thing from another class. For example:

An employee works for a company.
A university student is enrolled in a tertiary institution.

In these examples, a person cannot be an employee without working for a company, and a person cannot be a university student without enrolling in a tertiary institution. Accordingly, the functional schema of an employee type will contain an attribute representing the mutual property of being employed. Similarly, the functional schema of a student type will contain an attribute representing the mutual property of being enrolled in a university. Our ontology requires that these mutual properties be represented as relationship types (mutual attributes).

Consider, however, the notion of *possible* interaction. If things do not interact with other things, they have no mutual property that manifests an interaction. In ontology, recall that not having a property is not a property (Rule 6). Thus, a thing that interacts with another thing will have a *different functional schema* from a thing that does not interact with the other thing.

This analysis leads to several outcomes. First, using an optional relationship type amounts to "folding" two functional schemas into one entity type. This approach violates our rule that requires each class or type to be represented by its own functional schema (Rule 3). Second, when a thing acquires/loses an interaction that is important for modeling purposes, it should be represented by a new functional schema. According to our rules, therefore, it should be modeled as a new entity type (sub/super class or sub/super type, Rule 3). Third, when acquiring or losing a mutual property, the thing may also have acquired and lost several other properties. For example, assume a person who is a full-time employee becomes a student. As a result, the person ceases being a full-time employee and loses his/ her eligibility for the company's medical insurance. Thus, significant changes could occur in the relevant properties modeled.

Accordingly, we provide the following rules:

(1) Optional relationships should be avoided.

(2) The acquisition and loss of a mutual property should be modeled as a change in type/class.

(3) The ability of instances of a class to acquire an interaction (mutual property) without losing properties should be modeled as subclassification.

Note that point (2) above refers to an instance, and point (3) refers to the classification structure.

In ER modeling, we therefore advocate the use of an "IS-A" construct instead of optional relationships. Use of the IS-A construct allows all entity types to be modeled with mandatory (intrinsic and mutual) attributes only. Optional attributes, whether instrinsic or mutual, are not required.

To illustrate some implications of our rules, consider the example of university students who can borrow books from a library (Figure 4). When a student borrows a book ("book" refers to a specific volume), the book becomes unavailable in the library. Moreover, the student becomes responsible for returning the book. The states of the student and the book have changed. Thus, students and books *may* interact, which is often represented via an optional relationship (note the zero cardinality number in Figure 5). When a student actually borrows a book, however, the student and the book acquire a mutual property. According to our rules, both the student and the book should now be described by new functional schemas, represented as new entity types (Figure 6). The student becomes a *borrower* (for simplicity, assume only students can borrow books), and the book

Fig. 4. Student and book—no interaction.

becomes a *borrowed-book*. In Figure 6, note that cardinality numbers are still used, but our rules proscribe using a cardinality number of zero.

Several advantages accrue from using subtyping rather than optional relationship types. (1) Optional relationship types do not model a situation satisfactorily when "acquisition" of a relationship signals a fundamental change in how we wish to model a thing. For example, consider a paper submitted to a journal. If the paper is accepted, in due course it becomes a published article with a substantial change to its properties. For example, it is published in a certain issue of a journal and will be referenced in citation indexes. Reclassification accommodates such changes more easily than optional relationships. Moreover, it clarifies the semantics associated with the paper.

(2) Using optional relationship types may lead to semantic ambiguities and inconsistencies because a single functional schema has been used to describe two classes of things, one for things with and one for things without the mutual attribute. Each class calls for another name, yet only one name can be used in the model. In the case of the journal paper, for example, it is unclear whether the word "paper" relates to a submitted paper or a published paper.

(3) The integrity constraints (rules/laws) governing things in the domain are clearer when subtyping is used. With subtyping, the integrity constraints apply to *all* things in the class. Moreover, *all* integrity constraints that apply to superclasses are inherited "naturally" by their subclasses. With optional properties, however, integrity constraints have to be qualified. For example, if unpublished papers and published papers are covered by a single entity type that has optional attributes, some integrity constraints will depend on the outcome of a test to determine whether a paper is published or unpublished. Moreover, when subclasses are then created for a class with optional attributes, the inheritance of integrity constraints is no longer straightforward. Some integrity constraints may be inherited by the subclass; others may not.

(4) Semantic ambiguities may arise because the chosen relationship type name is inconsistent with the chosen class name. For example, modeling the fact that "An instructor teaches a course" with an optional relationship implies that not every instructor teaches a course. In a university, however, the title "instructor" implies the person *does teach* a course. Therefore, the class name implies different semantics from those implied by the relationship name.

(5) Semantic ambiguities may arise because a relationship name itself may imply that it *must* be possessed by a thing whereas a structural constraint indicates it is optional. For example, the fact stated as "department controls project" implies that every department controls at least one

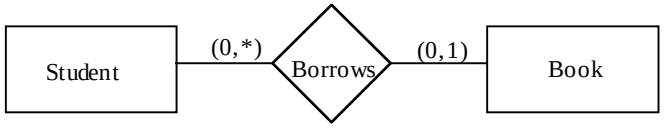

Fig. 5. Student may borrow—the use of an optional relationship.

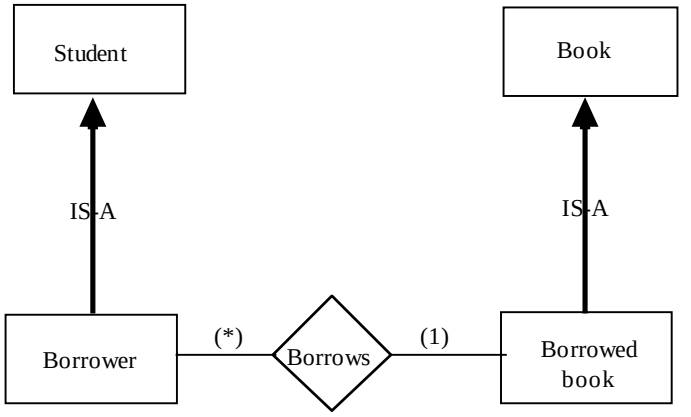

Fig. 6. Student and book acquire a mutual property—borrows.

project. Yet notation indicating the relationship is optional implies otherwise (Figure 7). Which interpretation, therefore, is correct?

In practice, such ambiguities are often resolved using our experience and commonsense. For example, we might know that controling a project is not a salient characteristic of a department and interpret *controls* as *may control*. In some cases, however, we might be modeling an organization where departments can only exist if they have projects under their control. Should we then follow the semantics implied by the words (and our experience/commonsense), or the structural constraint specified in the model? Admittedly, the distinction might not be significant for the practical design of a database. If a conceptual model is intended to convey information about an enterprise and to facilitate problem solving, however, such ambiguities should be avoided.

In our analysis above, note that we assume that optional attributes stand for *real* properties. For example, an analyst might want to indicate that a person lives in either an apartment or a house. The analyst might draw a conceptual schema diagram, therefore, with two optional relationships: one connects a person entity type to a house entity type; and the other connects the person entity type to an apartment entity type. Because the analyst has made a distinction between houses and apartments, we assume this distinction indicates the analyst is seeking to distinguish between two real properties. For example, the real properties might pertain to the types of insurance needs that house dwellers are likely to have versus those that apartment dwellers are likely to have. According to our rule, therefore, the

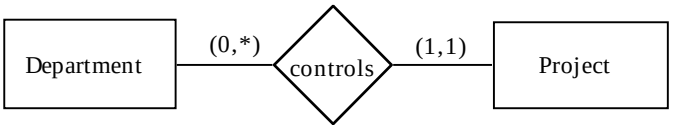

Fig. 7. Department controls a project (adapted from Elmasri and Navathe [1994, p. 58]).

analyst should replace the optional properties with two subtypes: one for house dwellers and the other for apartment dwellers.

In some cases, however, optional properties are simply alternative *proxies* for the same underlying real property. In our example above, the real property of concern to the analyst might simply be a person's place of residence. At the *conceptual modeling level*, therefore, the analyst does not need to distinguish between different types of residences because the domain stakeholders care only that a person has a residence of some sort. At the *implementation modeling level*, however, designers might want to provide alternative representations of this underlying real property; for example, address-line fields/domains that are best suited to houses versus address-line fields/domains that are best suited to apartments. Hence, a schema diagram prepared to assist design and implementation of the database might show optional relationships to accommodate these two alternative types of addresses. We stress that these are implementation concerns, however, and not conceptual modeling concerns.

In short, our rule in relation to optional (intrinsic and mutual) attributes is straightforward. If, in the context of some domain, an optional attribute in a conceptual schema diagram represents an alternative real property, a subtype must be created. If an optional attribute in a conceptual schema diagram simply represents an alternative "label" for some underlying real property, however, then implementation modeling has been confused with conceptual modeling. At the implementation level, use of optional relationships (and attributes) will perhaps lead to a satisfactory database design. We believe this issue needs further research because we suspect that optionality undermines understanding, even when it pertains to attributes that stand for alternative labels of a common real property rather than alternative real properties themselves.

We recognize that the downside of our rule is that the size of a conceptual model might sometimes increase markedly if subtypes with mandatory properties are used in place of optional relationships. Nonetheless, the primary purpose of a conceptual model is to provide clear semantics about a domain to its users. A concise conceptual model achieves little, therefore, if its semantics are ambiguous. In this regard, Gemino [1998; 1999] and Bodart et al. [1998], during experimental work, found that users who employ a conceptual model where optional properties are proscribed undertake better problem solving in relation to the domain described than users who employ a conceptual model where optional properties are permitted.

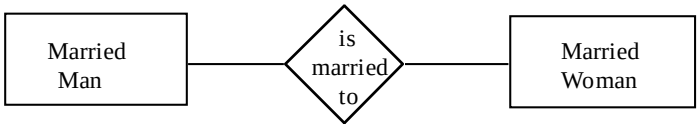

Fig. 8. Marriage as a relationship (adapted from Hansen and Hansen [1992, p. 85]).

5.4 Composite Things

In the ER model, the composition construct is represented in various ways. For example, use of a "part-of" relationship to connect a component and its composite is common in the extended entity-relationship approach (e.g., Fahrner and Vosson [1995]). Sometimes, however, a relationship symbol rather than an entity symbol is used to represent a composite thing (Figure 8). Also, composition is sometimes shown by physically enclosing the entity symbols representing the components within the entity symbol representing the composite (Figure 9).

Our ontological model, however, allows us to determine clear-cut rules for modeling composite things:

(1) The functional schema of a composite thing and the functional schemas of its components should be represented as entity types/classes.

(2) Each component should be linked with the composite entity via a (directional) *part-of* relationship type construct (representing a binding mutual property).

(3) The emergent properties of the composite should be modeled as attributes and relationship types.

These rules have several implications. First, because a composite is a thing, it has its own properties. Moreover, recall that the ontological model requires that a composite *must* possess at least one emergent property. This property can be either intrinsic (shown as an attribute) or mutual (shown as a relationship type). If we model an aggregate without additional properties, the model is incomplete.

Second, we need to recognize that things that are part-of another thing may play different *roles*. For example, two people can be a part of a team. One may be the team leader, however, and the other may be a regular member. Thus, we sometimes need to embellish our statements about the part-of relationship to indicate the role to be played:

A is part-of B in the role of . . .

The implication is that the specific *role* that a part plays means we must form subclasses of a class of things to reflect different roles. For example, we need to classify individuals who make up a team into team members and team leaders. Otherwise, as we show below, we will be forced to violate one of our earlier rules and to use optional properties to indicate that

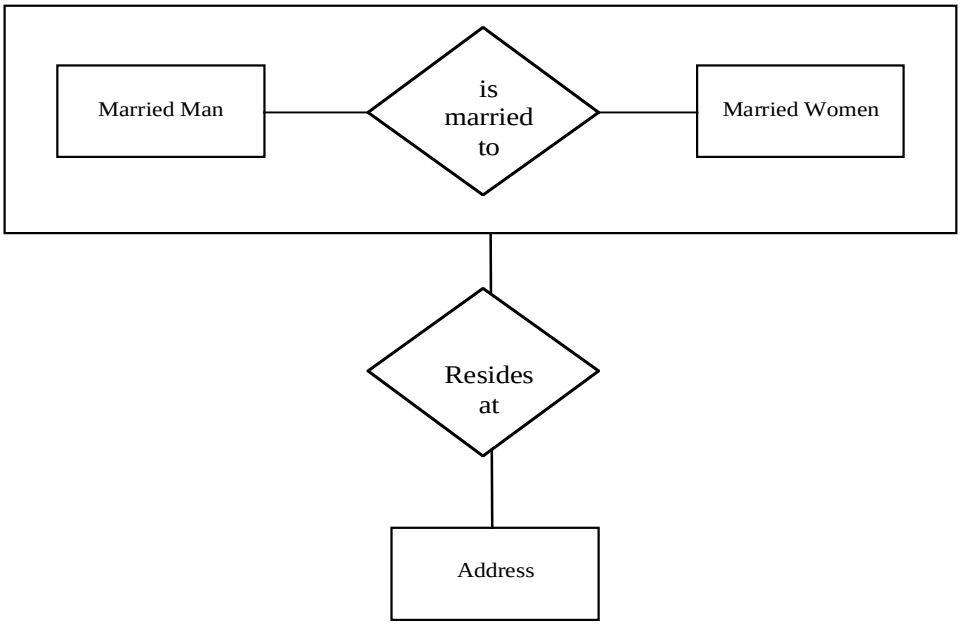

Fig. 9. Marriage as an aggregate (adapted from Hansen and Hansen [1992, p. 85]).

individuals may sometimes be team members and sometimes be team leaders.

Third, showing both the composite and the components provides richer meaning to the part-of relationship-type construct. In particular, the model can accommodate the minimum and maximum number of components of each type. Moreover, because each component's role can be represented, different constraints can be associated with each role.

To illustrate the advantages of modeling a composite according to these rules, consider an ER model representation of a *team* made up of members and a team leader, all of whom are *employees* (Figure 10). Assume that the team is represented by an optional relationship construct (contrary to our rules pertaining to optional properties).

In this representation it is difficult to show that the team must have at least one member or that it has exactly one leader, as the structural constraints relate to *employees*, not to the team.

Figure 11 shows how we model *team* according to our rules for composites. First, both *team-member* and *team-leader* are subclasses of *employee*. Note that the union of these two subclasses might not equal the employee class. In other words, some employees may be neither team members nor team leaders. Moreover, the team-member and team-leader subclasses might not be disjoint. For example, we might consider all team leaders to also be team members. Alternatively, even if we do not categorize team leaders as team members, some employees may be leaders of one team but members of another team. To avoid construct overload, we need to annotate our class/subclass/entity type symbols to show whether we are dealing with

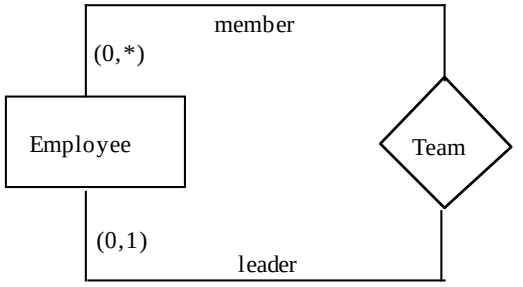

Fig. 10. TEAM as a relationship.

subsets or proper subsets or whether we are dealing with partially overlapping, totally overlapping, or disjoint sets (see, e.g., Elmasri and Navathe [1994, pp. 618–619]). Alternatively, we might use different symbols to represent the different circumstances.

Note that our choice of names for the entity types/subclasses indicates the *role* they play in the composition relationship; that is, team leader or team member. The use of subclasses indicates that instances of the subclass have an additional property, namely, they are either team leaders or team members.

Note that *team member* and *team leader*, which are the components, and *team*, which is the composite, have all been modeled as entity types. They are all *things* that have certain properties we wish to model. Indeed, recall that *team* must have at least one emergent property (either intrinsic or mutual) that we wish to model.

The composition relationship is shown via a *part-of* (relationship-type) symbol. We have also added an arrow to the part-of symbol to indicate the direction from the component to the composite. Overall, the symbols make clear to the user of the diagram that we are seeking to represent a *mereological* relationship rather than a *topological* relationship between things.

Our model contains no optional relationships. The cardinality constraints indicate that:

(a) A team member *must be* a member of at least one team and *can be* a member of two or more teams.

(b) A team leader *must be* a team leader of at least one team and *can be* a team leader of two or more teams.

(c) A team *must have* at least one team member and *must have* only one team leader.

Representing the composite as an entity type enables it to be a *part-of* another entity type. In this way, a hierarchy of composite things can be represented where a thing in the hierarchy can be *part-of* a "higher-level" composite. For example, a team can be part-of a department.

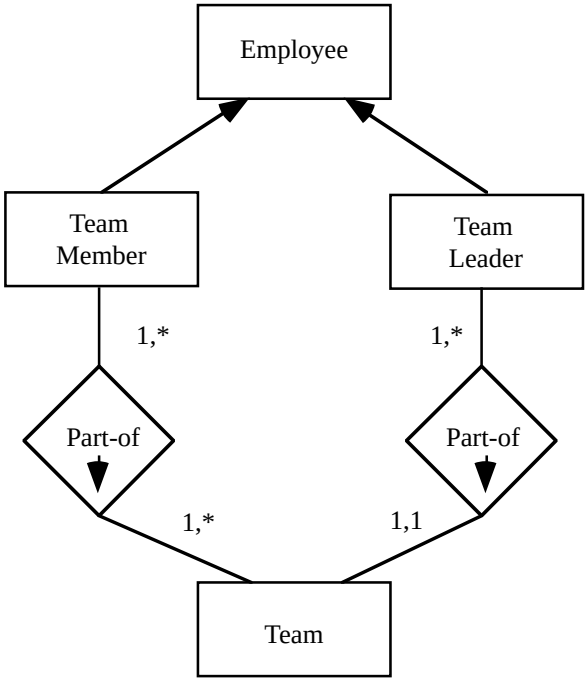

Fig. 11. Using subtyping with mandatory properties.

Finally, because the composite is represented as a class of things, it can be a subclass itself (e.g., of a unit) or a superclass (e.g., of a project team).

To further illustrate the application of our rules to a domain that is sometimes awkward to model, consider a bill-of-materials application; that is, one where components are parts of a higher-level assembly that itself may be a component of a higher-level assembly. Figure 12 shows one way to model a bill-of-materials application using our rules. Note that we have two entity types/classes, namely, higher-level assembly and lower-level assembly–linked by a part-of relationship type. The intermediate components (components that are composites themselves and a component of another composite) are represented by the intersection of the two classes. The highest-level components (components that are not part of another component) can be determined by subtracting the lower-level class (set) from the higher-level class (set). Similarly, the lowest-level components (components that are not made up of other components) can be determined by subtracting the higher-level class (set) from the lower-level class (set). The entity-type symbols representing the two classes overlap to show they are not disjoint. Note that there are no optional part-of relationship types. Nonetheless, Figure 12 still shows the basic recursive nature of the relationship.

To summarize, we argue the following rules apply to modeling composites:

(1) The class of composite things should be represented as a class (entity type).

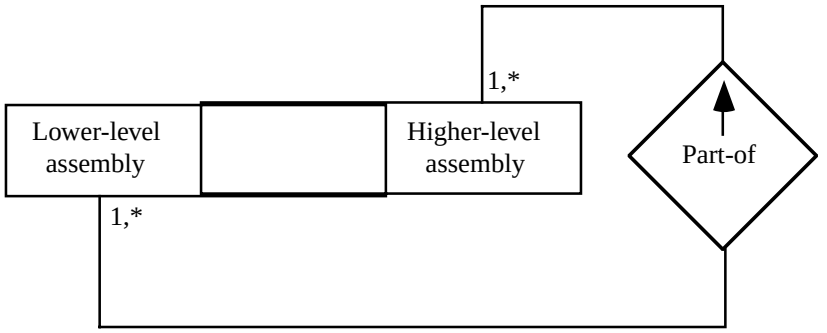

Fig. 12. Bill-of-materials application.

(2) Instances of the class of composite things should have attributes representing their *emergent* properties.

(3) Each component type should be represented as a class (entity type).

(4) Each component type should be linked to the composite via a (directional) *part-of* relationship type.

(5) The *role* of a component should be described using meaningful names for the subclass of which the component is a member.

Structural constraints can be used to indicate the following information for the *composite*:

(1) the minimum number of components in a given role (must be at least one, but may be greater);

(2) the maximum number of components in a given role;

(3) if several components of the same type are allowed to have the same *part-of* role, they represent interchangeable things.

Structural constraints can be used to indicate the following information for a *component*:

(1) the minimum number of composites (must be one or more) in which the component plays some part-of role (note that if a thing is not always a component of another thing, two subclasses must be formed: one to describe the independent thing and the other to describe the component thing);

(2) the maximum cardinality, which represents a "load" measure that indicates the maximum number of composite entities *of a given type* in which the component can be a part *in a given role*.

6. CONCLUSIONS

In this paper we sought to show how a theory of ontology can be used to clarify the meaning of certain constructs that are widely used to undertake

conceptual modeling. In particular, we focused on the relationship construct, which is central to many conceptual modeling methodologies, but seemingly difficult to use in a clear and unambiguous way. We have shown how our ontological theory can be used to generate simple, prescriptive rules that we believe will enhance the meaning we seek to communicate via conceptual models in general and the relationship construct in particular. We have then shown how the relationship construct must be modeled in ER modeling if conceptual modelers are to comply with our rules, and indicated how current ER modeling practices are often at odds with our prescriptions. While these latter practices may suffice to design a database, they may lead to a loss of meaning when modeling a domain.

The merits of our theoretical analyses and prescriptions should now be assessed in four ways. (1) They can be evaluated for face validity. Specifically, they can be scrutinized to determine whether they are consistent with the ontological theory we employ as the foundation for our work. (2) They can be shown to experienced conceptual modelers to determine whether they have intuitive appeal. Our goal is to impact not only the theory, but also the practice, of conceptual modeling. Thus, the extent to which our rules are accepted by practitioners is an important test of their merits. (3) They can be applied to other conceptual modeling approaches to tease out their implications for these approaches. For brevity, in this paper we focus on the implications of our rules for ER modeling only. In various places throughout the paper, however, we foreshadow their implications for approaches such as OO modeling and ORM. As with ER modeling, it quickly becomes clear that our rules proscribe some of the current practices followed with OO modeling and ORM. (4) Formal empirical tests of our predictions must be undertaken. For example, experiments can be designed to test our predictions about whether certain conceptual modeling practices pertaining to relationships undermine the communication of meaning about an application domain and whether practices that follow our rules mitigate problems. As we indicated at various places in the paper, we have already begun this work with some of our colleagues, and the initial results are supportive of our rules. Much more work needs to be done, however, to evaluate the worth of the rules we have articulated in this paper.

ACKNOWLEDGMENTS

This research was supported in part by the Natural Sciences and Social Sciences Research Councils of Canada, the J. Mack Robinson School of Business at Georgia State University, the Australian Research Council, and GWA Limited. We thank the Associate Editor and three reviewers for comments on an earlier version of this paper.

REFERENCES

AGASSI, J. 1990. Ontology and its discontent. In *Studies on Mario Bunge's Treatise*, P. Weingartner and G. J. W. Dorn, Eds. Rodopi, Amsterdam, The Netherlands, 105–122.

ANGELES, P. A. 1981. *Dictionary of Philosophy*. Harper Perennial, New York, NY.

ASHENHURST, R. L. 1996. Ontological aspects of information modeling. *Minds and Machines 6*, 287–394.

BATINI, C., CERI, S., NAVATHE, S., BATINI, C., CERI, S., AND NAVATHE, S. B., Eds, 1992. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin-Cummings Publ. Co., Inc., Redwood City, CA.

BATRA, D., HOFFLER, J. A., AND BOSTROM, R. P. 1990. Comparing representations with relational and EER models. *Commun. ACM 33*, 2 (Feb. 1990), 126–139.

BODART, F., PATEL, A., SIM, M., AND WEBER, R. 1998. Should the optional property construct be used in conceptual modeling?: A theory and three empirical tests. Unpublished working paper. University of Queensland, Australia.

BRODIE, M. L. 1984. On the development of data models. In *On Conceptual Modeling*, M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, Eds. Springer-Verlag, New York, NY, 19–47.

BUNGE, M. 1977. *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World*. D. Reidel Publishing Co., Inc., New York, NY.

BUNGE, M. 1979. *Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems*. D. Reidel Publishing Co., Inc., New York, NY.

BURTON-JONES, A. AND WEBER, R. 1999. Understanding relationships with attributes in entity-relationship diagrams. Unpublished working paper. University of Queensland, Australia.

CHEN, P. P. 1976. The entity-relationship model: Toward a unified view of data. *ACM Trans. Database Syst. 1*, 1, 9–36.

COAD, P. AND YOURDON, E. 1991. *Object-Oriented Analysis*. 2nd ed. Yourdon Press Computing Series. Yourdon Press, Upper Saddle River, NJ.

DATE, C. J. 1995. *An Introduction to Database Systems*. 6th ed. Addison-Wesley Longman Publ. Co., Inc., Reading, MA.

ELMASRI, R. AND NAVATHE, S. B. 1989. *Fundamentals of Database Systems*. Benjamin-Cummings Publ. Co., Inc., Redwood City, CA.

ELMASRI, R. AND NAVATHE, S. B. 1994. *Fundamentals of Database Systems*. 2nd ed. Benjamin-Cummings Publ. Co., Inc., Redwood City, CA.

EMBLEY, D. W., KURTZ, B. D., AND WOODFIELD, S. N. 1992. *Object-Oriented Systems Analysis: A Model-Driven Approach*. Yourdon Press Computing Series. Yourdon Press, Upper Saddle River, NJ.

EMBLEY, D. W., JACKSON, R. B., AND WOODFIELD, S. N. 1995. OO systems analysis: Is it or isn't it?. *IEEE Software 12*, 1 (Jan.), 19–33.

FAHRNER, C. AND VOSSEN, G. 1995. A survey of database design transformations based on the entity-relationship model. *Data Knowl. Eng. 15*, 3 (June), 213–250.

GEMINO, A. 1998. To be or maybe to be: An empirical comparison of mandatory and optional properties in conceptual modelling. In *Proceedings of the Annual Conference on Administrative Sciences Association of Canada* (May 30-June 2), Administrative Sciences Association of Canada, Saskatoon, Sask., Canada, 33–44.

GEMINO, A. 1999. Empirical methods for comparing system analysis modelling techniques. Ph.D. Dissertation. University of British Columbia, Vancouver, Canada.

GOLDSTEIN, R. 1985. *Database: Technology and Management*. John Wiley and Sons, Inc., New York, NY.

GOLDSTEIN, R. AND STOREY, V. 1989. Some findings on the intuitiveness of entity-relationship constructs. In *Proceeding of the 8th International Conference on Entity-Relationship Approach to Database Design and Querying* (Toronto, Canada), 9–23.

GUARINO, N. 1995. Formal ontology, conceptual analysis and knowledge representation. *Int. J. Hum.-Comput. Stud. 42*, 6 (June 1995), 625–640.

HALPIN, T. 1996. *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*. 2nd. ed Prentice-Hall, Inc., Upper Saddle River, NJ.

HANSEN, G. W. AND HANSEN, J. V. 1992. *Database Management and Design*. Prentice-Hall, Inc., Upper Saddle River, NJ.

KENT, W. 1978. *Data and Reality: Basic Assumptions in Data Processing Reconsidered*. North-Holland Publishing Co., Amsterdam, The Netherlands.

MCFADDEN, F. R. AND HOFFER, J. A. 1985. *Data Base Management*. Benjamin-Cummings Publ. Co., Inc., Redwood City, CA.

PARSONS, J. AND WAND, Y. 1997. Choosing classes in conceptual modeling. *Commun. ACM 40*, 6, 63–69.

PAULSON, D. AND WAND, Y. 1992. An automated approach to information systems decomposition. *IEEE Trans. Softw. Eng. 18*, 3 (Mar. 1992), 174–189.

PRIETULA, M. J. AND MARCH, S. T. 1991. Form and substance in physical database design: An empirical study. *Inf. Syst. Res. 2*, 4 (Dec.), 287–314.

ROB, P. AND CORONEL, C. 1997. *Database Systems: Course Technology*. 3rd ed. International Thomson Computer Press, Boston, MA.

RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F., AND LORENSEN, W. 1991. *Object-Oriented Modeling and Design*. Prentice-Hall, Inc., Upper Saddle River, NJ.

SHLAER, S. AND MELLOR, S. J. 1988. *Object-Oriented Systems Analysis: Modeling the World in Data*. Yourdon Press Computing Series. Yourdon Press, Upper Saddle River, NJ.

SIMONS, P. 1987. *Parts: A Study in Ontology*. Clarendon Press, New York, NY.

SMITH, J. M. AND SMITH, D. C. P. 1977. Database abstractions: Aggregation. *Commun. ACM 20*, 6 (June), 405–413.

TAKAGAKI, K. AND WAND, Y. 1991. An object-oriented information systems model based on ontology. In *Proceedings of the IFIP Working Group 8.1 Conference on Object Oriented Approach to Information Systems* (Quebec, Oct.), IFIP.

TEOREY, T. J., YANG, D., AND FRY, J. P. 1986. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv. 18*, 2 (June 1986), 197–222.

VARZI, A. 1994. On the boundary between mereology and topology. In *Philosophy and the Cognitive Sciences*, R. Casati, B. Smith, and G. White, Eds. Hoelder-Pichler-Tempsky, Vienna, Austria.

WAND, Y. 1989. A proposal for a formal model of objects. In *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F. H. Lochovsky, Eds. ACM Press Frontier Series. ACM Press, New York, NY, 537–559.

WAND, Y. AND WANG, R. Y. 1996. Anchoring data quality dimensions in ontological foundations. *Commun. ACM 39*, 11, 86–95.

WAND, Y. AND WEBER, R. 1989. An ontological evaluation of systems analysis and design methods. In *Proceedings of the IFIP WG 8.1 Working Conference on Information Systems Concepts: An In-Depth Analysis* (Oct., Namur, Belgium),

WAND, Y. AND WEBER, R. 1990. An ontological model of an information system. *IEEE Trans. Softw. Eng. 16*, 11 (Nov. 1990), 1282–1292.

WAND, Y. AND WEBER, R. 1993. On the ontological expressiveness of information systems analysis and design grammars. *Eur. J. Inf. Syst. 3*, 217–237.

WAND, Y. AND WEBER, R. 1995. On the deep structure of information systems. *Eur. J. Inf. Syst. 5*, 203–223.

WEBER, R. 1996. Are attributes entities?: A study of database designers' memory structures. *Inf. Syst. Res. 7*, 2 (June), 137–162.

WEBER, R. 1997. *Ontological Foundations of Information Systems*. Coopers and Lybrand, Melbourne, Australia.

WEBER, R. AND ZHANG, Y. 1996. An ontological evaluation of NIAM's grammar for conceptual schema diagrams. *Eur. J. Inf. Syst. 6*, 2 (Apr.), 147–170.

WIMMER, K. 1992. Conceptual modelling based on ontological principles. *Knowl. Acquis. 4*, 4 (Dec. 1992), 387–406.

Received: April 1998; revised: March 1999; accepted: June 1999