VLDB Journal, 2, 455-488 (1993), Fred J. Maryanski, Editor
©VLDB

Understanding Semantic Relationships

Veda C. Storey

Received February 4, 1991; revised version received, February 2, 1992; accepted November 2, 1992.

Abstract. To develop sophisticated database management systems, there is a need to incorporate more understanding of the real world in the information that is stored in a database. Semantic data models have been developed to try to capture some of the meaning, as well as the structure, of data using abstractions such as inclusion, aggregation, and association. Besides these well-known relationships, a number of additional semantic relationships have been identified by researchers in other disciplines such as linguistics, logic, and cognitive psychology. This article explores some of the lesser-recognized semantic relationships and discusses both how they could be captured, either manually or by using an automated tool, and their impact on database design. To demonstrate the feasibility of this research, a prototype system for analyzing semantic relationships, called the Semantic Relationship Analyzer, is presented.

Key Words. Database design, entity-relationship model, relational model, semantic relationships, database design systems.

1. Introduction

One advance needed in database management systems (DBMSs) is the capture of some of the semantics of an application for which a database is developed. In particular, there is a need within the database community to extend the relational model to accommodate more real world knowledge (Reiter, 1984). Some initial steps have been taken through the incorporation of certain semantic relationships into DBMS designs, commonly referred to as data abstractions.

In general, an abstraction is a simplified description, or specification, of a system that emphasizes some of the system's details or properties while suppressing others (Shaw, 1984). Various researchers (Brodie, 1984; Ullman, 1986; Mattos, 1988) have pointed out the benefits of incorporating data abstractions into DBMSs to enhance the capabilities of these systems and to capture more of the semantics of an application. The most common abstractions are inclusion (subtype–supertype or

Veda C. Storey, Ph.D., is Assistant Professor, Computers and Information Systems, William E. Simon Graduate School of Business Administration, University of Rochester, Rochester, NY 14627.

is-a), aggregation (component), and association (membership) relationships (Smith and Smith, 1977; Blaha et al., 1988; Mattos, 1988; Potter and Kerschberg, 1988; Davis and Bonnell, 1989; Mattos and Michels, 1989; Goldstein and Storey, 1991, 1992).

1.1 Semantic Relationships

In addition to the common data abstractions mentioned above, research in linguistics, logic, and cognitive psychology has recognized many more semantic relations[1] (Winston et al., 1987). Landis et al. (1987) divided semantic relationships into antonym, synonym, class inclusion, part-whole, and case relationships. Chaffin and Herrmann (1988) provided a list of 31 semantic relationships that are broken into categories similar to the above. Winston et al. (1987) presented a taxonomy of relationships dealing with objects and their components, and other concepts related to the part-whole or *meronymic* relationship. Brachman (1983) analyzed different interpretations of the *is-a* relationship and Schubert et al. [1983] discussed recognition of *part-of, color,* and *time* relationships.

Figure 1 presents a taxonomy of the seven types of semantic relationships analyzed here (*inclusion, possession, attachment, attribution, antonym, synonym,* and *case*). The taxonomy is based on the work of Winston et al. (1987) and Chaffin et al. (1988), but is expanded to include the other classes of semantic relationships identified by Landis et al. (1987). An interesting feature of the taxonomy is that it places the well-known data abstractions within the context of a broader set of semantic relationships. In particular, inclusion as found in the database literature corresponds to class inclusion in the taxonomy; aggregation corresponds to component-object; and association to member-collection.

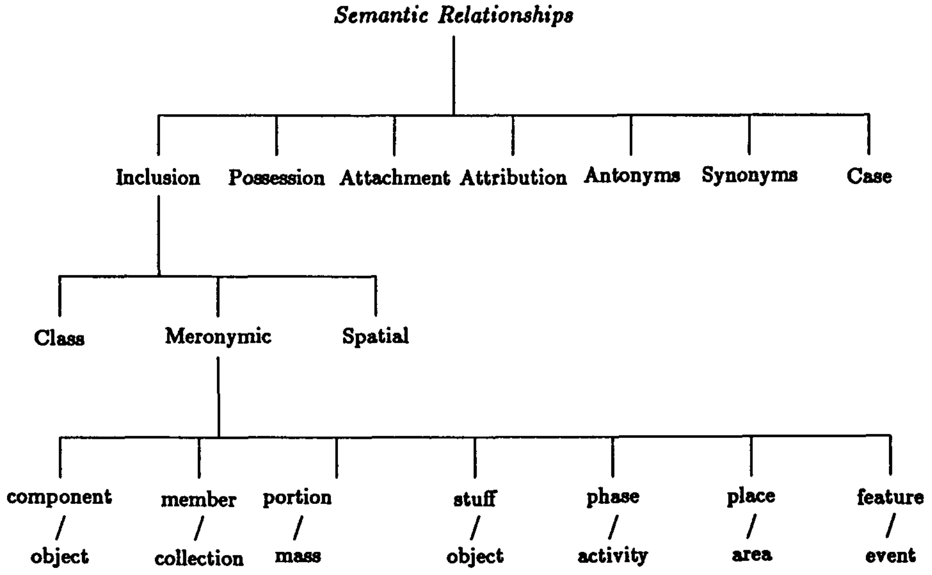
The overall objective of this research is to analyze some of the lesser-known semantic relationships to understand better how they can be employed to create a design that accurately reflects the semantics of an application. The guidelines provided for doing so are intended to be useful for both automated database design tools and human database designers. Specifically the objectives are to:

- Make human designers more aware of the existence and classification of various types of semantic relationships and their design implications.
- Provide heuristics that allow an interactive database design system to use what it "knows" about semantic relationships (from the analysis carried out in this article) to:

    1. capture more information about a design problem from a user;

1. Hereafter referred to as semantic relationships.

Figure 1. Semantic relationships



Based on Landis et al. (1987); Winston et al. (1987); Chaffin et al. (1988).

2. refine a user's input so that information is obtained that better reflects the semantics of the user's application.

To demonstrate the implications of this research, a prototype system for examining semantic relationships, called the Semantic Relationship Analyzer, is presented.

This article is divided into five sections. The remainder of this section discusses database design and automated database design tools. Seven categories of semantic relationships are presented in Section 2 and their impact on database design is analyzed. Have/has relationships, which are often used in data modeling, but which can be ambiguous in meaning, are discussed in Section 3. The Semantic Relationship Analyzer is presented in Section 4. Section 5 summarizes and concludes the article.

1.2 Database Design

Database design can be defined as the process of capturing the relevant information and processing requirements of an enterprise and mapping them onto an underlying DBMS (Dogac et al., 1989). This design process is often thought of in four stages. During the *requirements analysis* phase, an analysis is made of the information needs within an organization, resulting in a preliminary specification of the information needs of various user groups. *Conceptual design* models and represents the users' and applications' views of information and, possibly, a specification of the processing or use of the information in a manner that supports, for example, the Entity-

Relationship (E-R) (Chen, 1976) or some other semantic data model. During the *logical design phase,* a logical schema (or design) that corresponds to the data model of the chosen DBMS is produced; for example, a relational data model. Finally, *physical design* transforms the logical data model design into a form that is suitable for the given hardware and DBMS.

The focus of this research is on identifying semantic relationships in the conceptual design phase and determining how they should be transformed into a logical design. This is illustrated in terms of the E-R model, which is well-known as an effective conceptual modeling tool (Tauzovich, 1989), and its translation into a relational model which has become widely accepted as an implementation model.

The semantic relationships discussed in this article are binary ones. In a relationship, *A verb phrase B,* the min/max cardinalities (Tsichritzis and Lochovsky, 1982) represent the minimum and maximum number of occurrences of the entity type, *B,* that can exist for each occurrence of the entity type, *A,* and vice versa. Consider, for example:

Company employs	*Employees*
(1,*)	(1,1)

This relationship is interpreted to mean that each company employs between one and many (*) employees; each employee is employed by one and only one company.

In general, when an entity-relationship model is converted to a relational model, each entity becomes a separate *entity relation* (Teorey et al., 1986) that looks exactly the same as the entity (i.e., it has the same key and non-key attributes.) This entity relation may then be modified if a relationship is represented by adding another entity type's key to it as a foreign key. Teorey et al. (1986) refer to this modified entity relation as an *extended entity relation.* Alternatively, a relationship may be represented by creating a separate *relationship relation* whose key is the concatenation of the keys of the involved entity types and whose non-keys are the relationship's attributes. In general, when the min/max cardinalities of one entity type are (1,1) in a relationship, the key of the other entity type is added as a foreign key to the entity type with the (1,1) cardinalities. All other relationships are many-to-many and are represented as separate relations. Furthermore, it is the many-to-many relationships that can have relationship attributes.[2] Thus, the cardinalities are most important because they dictate how best to represent a (semantic) relationship in a relational data model.

The conceptual results of this research should also be applicable to the emerging object-oriented systems (Kim, 1990). This is because, given an entity-relationship model, an entity would become an object in an object-oriented system (with information provided on its behavior). A relationship could correspond to an aggregate

2. For further discussion on database design, see Elmasri and Navathe (1989), Storey (1988, 1991*a*), Storey and Goldstein (1988), and Teorey et al. (1986).

object (e.g., *Assignment* can be thought of as an aggregate of *Project* and *Employee*) or an attribute (e.g., *Employee has Name*).

Recently, a number of systems have been developed that attempt to automate the database design process, among which are SECSI (Systéme en Conception de Systéme d'Informations; Bouzeghoub et al., 1985); the View Creation System (Storey, 1988; Storey and Goldstein, 1988; 1990*a*; 1990*b*; AVIS (Automated View Integration System; Wagner, 1989); OICSI (Rolland and Proix, 1986; Cauvet et al., 1988; Proix and Rolland, 1988); and CARS (Computer Aided Requirements Synthesis; Demo and Tilli, 1986).[3] Some of these systems rely on the user (who may or may not be a database design expert) to identify and provide, as input, the relationships that one would find, for example, in an entity-relationship model of the user's application. Most of the systems deal with the input on a syntactical basis only; that is, they simply treat the names of the entity types and verbs that appear in relationships as strings of characters. They do not have sophisticated means for capturing much information about an application beyond that provided by the user.

There are various ways in which information about semantic relationships could aid an automated tool for database design. First, many database design tools are interactive where the user is either a database designer or, possibly, an end-user. If a user were to provide, as input, a (semantic) relationship of the form, *A verb phrase B,* a database design system could "understand" the design impact of the relationship and make inferences about it. This is easily demonstrated by the well-known class inclusion or *is-a* relationship. If a user indicates that *Managers are Employees,* for example, then a system could automatically delete from *Manager* any (non-key) attributes that are found in *Employee* because *Manager* could "inherit" them from *Employee.* This minimizes redundancy that might appear in the design. Similarly, if the user provides, as input, the three relationships, *Foreman is-a Manager, Manager is-an Employee,* and *Foreman is-an Employee,* then the system can ignore the latter relationship because it is redundant. These types of inferences are possible because the semantics of *is-a* relationships are well-understood.

Based on what a system "knows" about semantic relationships (from the analysis carried out in this article), it could make certain design decisions. For example, if a member–collection relationship (such as *Employee member-of Committee*) is identified, the system could infer that a priori attributes (e.g., "max#") and derived attributes (e.g., "average-age") are applicable and prompt the user to provide them.

Dahlgren (1988) distinguishes between things which are "typical" and "inherent" in the real world. Typical things usually take place (e.g., managers usually prepare budgets). Inherent properties are always true (e.g., employees have names and addresses). One would expect a system with knowledge about the real world to "know" these types of things. Some could be captured using case relationships

3. For an overview of these and others, see Lloyd-Williams (1991); Ram (1989); Storey and Goldstein, in press; Storey, 1993, in press).

(discussed later). As an extreme situation, suppose that the system had a dictionary of semantic relationships that included applications found to be most useful in previous design sessions. Then the system could suggest, for example, an *Employee* entity type through which various kinds of employees were related by *is-a* relationships (*Manager is-an Employee, Secretary is-an Employee,* etc.). Unfortunately, there are not, as yet, adequate capabilities for database design systems to capture such real world knowledge (Storey, 1992*a*). Some work, however, has been carried out. The SECSI system (Bouzeghoub, 1985), for example, allows a designer to add some domain-specific knowledge. Tauzovich (1989) suggested that learning mechanisms for the refinement and customization of the knowledge base of the Modeller system would be appropriate. A goal of the I^2S system (Kawaguchi et al., 1986) is to "learn" information about certain application domains as it is used over time.

Semantic relationships can be problematic to understand and distinguish among (Winston et al.,1987; Kuczora and Cosby, 1989). Some verb phrases often used in expressing relationships between entity types can have many different meanings. A database design system could assist a user in determining the best interpretation for the application. Some of the most common of these ambiguous relationships are discussed later in this article and it is shown how they can be modified to be more meaningful.

2. Analysis of Relationships

Each of the relationships of the taxonomy presented in Figure 1 is analyzed in this section and its database design implications are highlighted.

2.1 Inclusion

The inclusion relationship describes situations where one entity type comprises or contains other entity types. Winston et al. (1987) identified three different types of inclusion: class, meronymic, and spatial.

2.1.1 Class Inclusion. Class inclusion is the standard subtype/supertype relationship that frequently appears in data modeling (Hull and King, 1987; Peckham and Maryanski, 1988) and is often expressed as *A is-a B* where *A* is referred to as the specific entity type and *B* the generic entity type (e.g., *Manager is-an Employee.*) Class inclusion is hierarchical and may be expressed as: *X's are a type of Y; X's are Y's; X is a kind of Y;* and *X is-a Y* (Winston et al., 1987). Examples include *Cars are a type of Vehicle; Roses are Flowers; Theft is a kind of Crime;* and *Employee is-a Person.*

Within the database literature, different kinds of inclusion have been distinguished. These are: *classification*, which relates an entity occurrence to an entity type; *generalization*, in which an entity type is the union of non-overlapping subtypes; *specialization*, which is defined as the inverse of generalization (Codd, 1979; Tsichritzis and Lochovsky, 1982; Urban and Delcambre, 1986; Potter and Kerschberg, 1988) and *subset hierarchy* (Teorey et al., 1986), in which possibly overlapping subtypes exist. For example, *Ian Ross is-an Engineer* is a classification. A generalization hierarchy could be defined by the two relationships, *Part-Time Employee is-an Employee* and *Full-Time Employee is-an Employee,* because *Part-Time Employee* and *Full-Time Employee* partition the generic entity type *Employee. Part-Time Employee* is a specialization of *Employee.*[4] The three relationships *Full-Time Employee is-an Employee, Part-Time Employee is-an Employee,* and *Secretary is-an Employee* form a subset hierarchy because, obviously, there could be overlap among the specific entity types (*Part-Time Employee, Full-Time Employee,* and *Secretary*).

Database Design Implications: Class Inclusion.

- A class inclusion relationship always has min/max cardinalities:

  Manager is-an Employee
  (1,1) (0,1)

- As previously mentioned, the well-known inheritance principle of *is-a* relationships (Brachman, 1983) states that anything that is true about the generic entity type, *B,* must also be true of the specific entity type, *A.* Any attributes of *B,* therefore, are also attributable to *A* (but not necessarily vice versa). Similarly, in whichever relationships *B* can participate, *A* can participate also. Database designers, and therefore database design systems, can take advantage of this property by storing common attributes in the entity type at the highest level in an *is-a* hierarchy and allowing the entity types lower in the hierarchy to inherit them. Thus, any candidate key of a supertype entity can serve as a legitimate candidate key for a subtype entity.

- Relationships of the form *X is-a A* and *X is-a B* form an *is-a* lattice. Then, *X* can inherit attributes from both *A* and *B.* Inheritance conflicts can occur if two semantically different attributes of *A* and *B* are given the same name. Prefixing the attributes by their entity names is one way to help overcome such a problem. An *is-a* lattice can also highlight potential design errors. The correct interpretation of the above is that both *is-a* relationships hold simultaneously. If not, then modification to the design is required. Consider, for example, the relationships *Project-Participant is-an Employee* and *Project-Participant is-a Student.* If all *Project-Participants* are not both *Employees* and *Students,* then the common attributes of someone (employee or student) who participates in a project can be "abstracted

4. Note that specialization refers to one *is-a* relationship. For example *Publication* is a generalization of *Journal-Paper, Book,* and *Conference-Paper,* whereas *Book* is a specialization of *Publication* (Peckham and Maryanski, 1988, p.155).

out" and stored in the entity type *Project-Participant.* Two additional entity types, *Employee-Participant* and *Student-Participant* are needed. Then the *is-a* relationships become *Employee-Participant is-an Employee* and *Employee-Participant is-a Project-Participant.* Analogous relationships are needed for *Student-Participant* (Goldstein and Storey, 1991; 1992).

- Class inclusion is easily confused with member-collection relationships (as well as other meronymic relationships) because both involve membership of individuals in a larger set. Meronymic relationships, however, are determined on the basis of characteristics that are extrinsic to the individual members themselves. Class inclusion is determined by similarity to other members based on an intrinsic characteristic (Kuczora and Cosby, 1989).

- Winston et al. (1987) suggested that class inclusion may be distinguished from meronymic relationships when expressed as "kind of" versus "part-of." One is not inclined to say, for example, that *Robin is part-of a Bird* or *Wheel is kind-of a Car.* This, then, provides a simple mechanism for a database design system to determine if a relationship is best classified as *is-a* (class inclusion) or *part-of* (part-whole) without requiring that the user be aware of the distinction between the two.

- Chaffin et al. (1988) identified the following four kinds of class inclusion:

Type	Example
1. Natural Object–Kind	*Employee is-a Person*
2. Artifact–Kind	*Microcomputer is-a Computer*
3. State–Kind	*Single kind-of Marital Status*
4. Activity–Kind	*Consulting kind-of Work*

Both *natural object-kind* and *artifact-kind* relationships would be represented by two entity types connected by *is-a* relationships where all of the design implications of *is-a* relationships hold. The other two, *state-kind* and *activity-kind,* would be best represented by making the state or activity an attribute of an appropriate entity type. Then, the *kind* would be an instantiation of that attribute. For example, *"single"* would be a value of the (state) attribute *"marital-status"* and *"consulting"* a value of the (activity) attribute *"work."*

- Transitivity properties hold between *is-a* and *part-of* relationships (Evens, 1988); that is, if *X is-a Y* and *Z part-of Y,* then *Z part-of X.* If all three relationships were to appear in a database design, a system could infer that the relationship *Z part-of X,* is redundant and therefore should be deleted.

Personal Computer (X)	*is-a*	Computer (Y)
CPU (Z)	*part-of*	Computer (Y)
therefore:		
CPU (Z)	*part-of*	Personal Computer (X)

2.1.2 Meronymic Inclusion. Meronymic (from the Greek word "meros" for part) relationships occur between something and its parts (Winston et al., 1987). Seven different types of meronymic relationships, based on Storey (1991*b*), are discussed below.

1. **Component-Object.** The *component-object* relationship takes place between a component and the object of which it is a part (e.g., *Engine component-of Car*). This is the aggregation abstraction in data modeling. An *aggregation* is an abstraction in which a relationship between objects (e.g., an entity type) is considered a higher-level object (Smith and Smith, 1977). This makes it possible to focus attention on the aggregate while suppressing low-level detail. *Reservation,* for example, can be thought of as an aggregate of *Hotel, Person,* and *Date* (Smith and Smith, 1977). A component of an aggregate can be either: *relevant* (the component can exist, but is not necessary for the aggregate); *characteristic* (the component is required for the existence of the aggregate); or *identifying* (the component is both required and uniquely identifies the aggregate [Dos Santos et al., 1980]). Examples are:

   Relevant:
   Receptionist component-of Company
   (0/1,1) (0,∗)
   Characteristic:
   Employee component-of Company
   (0/1,1) (1,∗)
   Identifying:
   CEO component-of Company
   (0/1,1) (1,1)

   The minimum cardinality for the part entity type is denoted by 0/1 to indicate that, in many applications, the part can be considered separate from its attachment to the whole. If the part entity type can be considered independently of its attachment to the aggregate (Blaha et al., 1988), then the minimum cardinality is 0. If it is only considered with respect to the whole, then the minimum cardinality is 1.

   Database Design Implications: Component-Object. The cardinalities for the different types of aggregation are (Goldstein and Storey, 1991):

Relevant component-of Aggregate
(–,–) (0, > 0)
Characteristic component-of aggregate
(–,–) (≥ 1, $\geq$ min)
Identifying component-of aggregate
(0/1,1) (1, ≥ 1)

The cardinality expression $\geq$ *min* means that the maximum cardinality, as always, must be greater than or equal to the minimum cardinality. This ensures that nonsensical cardinalities, such as (2,1), are avoided for the aggregate. A system could ask the user appropriate questions to determine which kind of component exists and then verify that the cardinalities are appropriate for that type of component (or vice versa).

2. **Feature-Event.** The *Feature part-of Event* relationship is exemplified by *Trapeze Act part-of Circus* (Chaffin et al., 1988). Events resemble objects (in component-object relationships) in the sense that they have a predetermined structure or arrangement of parts. Events differ from objects in that an event may have parts (features) that occur at different moments in time, whereas the parts of an object typically occur at the same time.

   Demonstration part-of Presentation
   (0,$*$) (0,$*$)

   Database Design Implications: Feature-Event. Database management systems do not have a special means for representing events. A *feature–event* relationship, therefore, should be represented following the normal rules for relationship representation (i.e., by a foreign key or by constructing a separate, relationship relation).

3. **Member-Collection.** This relationship is the association abstraction (also referred to as *belongs-to* and *grouping* or *partitioning,* Brodie, 1984) where a set of members is considered an object in its own right (e.g., *Employee member-of Committee*). It differs from the component-object relationship in that it does not assume that the member performs a specific function in the collection.

   Database Design Implications: Member–Collection.

   - There are, in general, no restrictions on the cardinalities: *A (0,$*$) member-of B (0,$*$)*.
   - An association relationship can have both a priori and derived attributes (Brodie, 1981). For example, “max-number” is an a priori attribute

of *Committee* whereas "average-age" is a derived attribute. Therefore, having established that a relationship is member-collection, a system could prompt a user to identify such attributes.

- Chaffin and Herrmann (1988) identify three subcategories of member-collection: (1) *member-group* (e.g., brother-fraternity, cow-herd); (2) *member-collection* (e.g., tree-forest, ship-fleet, book-library); and (3) *unit-organization* (e.g., delegation-UN, battalion-army, registry-college). These do not appear to have any particular design implications, however, understanding that there are different categories could help the designer to recognize better when a member–collection relationship occurs.
- Hierarchies of this relationship are possible; e.g., *Performance member-of Concert member-of Concert-Series.*

4. **Portion-Mass.** In a *portion-mass* relationship, the part is similar to all other parts and to the whole, e.g., *Slice part-of Pie.* Every portion of a pie is "pie" and is similar to each other slice and to the whole pie (Winston et al., 1987). This relationship differs from component–object and member–collection relationships where the parts may be dissimilar to each other, and different from the wholes which they comprise (Kuczora and Cosby, 1989).

Database Design Implications: Portion-Mass.

- Many of the properties of the mass are inheritable by the portion. If *Slice part-of Pie,* for example, then "kind of pie," "texture," etc. would all be inheritable by *Slice;* "no. of slices," however, would be an attribute of pie only and (in this case) derived directly from its portions.
- The key of the portion should include the key of the mass. For example, *Slice* would be identified by the pie to which it belongs and the "slice#" within that pie. A system could prompt the user to provide both of these types of attributes.

5. **Phase-Activity.** The *Phase part-of Activity* relationship relates a phase to an activity (or process) and is similar to the feature–event relationship, except that the phases cannot be separated from the activity (whereas a feature can be separated from an event). Examples include: *Adolescence part-of Growing Up, Paying part-of Shopping,* and *Billing part-of Consulting.*

Database Design Implications: Phase-Activity. Databases do not have any special means for representing activities so this relationship would be represented following normal database design principles. For example:

```
Billing part-of Consulting
(1,1)           (1,*)
```

This implies that each *Billing* can be identified uniquely (e.g., by "invoice#"). *Consulting* corresponds to different types of consulting activities (e.g., "market research."

6. **Place-Area.** A *place-area* relationship takes place between an area and special places or locations within it; for example, *Everglades place in Florida* and *Reception-Area place in Office.* This is similar to the portion-mass relationship except that the parts cannot be separated from the whole. The part (place) is similar to the whole (area) as in the portion-mass relationship.

   Database Design Implications: Place-Area. This type of relationship would be most significant for geographic databases.[5] One would expect the place and area entity types to have attributes in common. For example, *Reception-Area* and *Office* should both have attributes "rented-by," "#-square-meters," etc. However, the values of the attributes might differ. One would certainly expect the "#-square-meters" of an *Office* to be larger than that of its *Reception-Area.*

7. **Stuff-Object.** A *stuff-object* relationship deals with a constituent of an object but differs from the component-object relationship in that the stuff cannot be physically separated from the object without altering its identity (as a component can). *Bicycle is-partly Aluminum* is a stuff-object relationship (Winston et al., 1987).

   Database Design Implications: Stuff-Object. The above relationship could be represented by an attribute "metal-made-of" for which a value of "aluminum" would appear.

2.1.3 Identification of Meronymic Relationships. Chaffin et al. (1988) and Winston et al. (1987) suggested that the following four dimensions may be used to distinguish among meronymic relationships.

- *Function*—whether the parts are in a specific position with respect to each other that supports their functional role with respect to the whole. For example, the handle of a cup can only be placed in a limited number of positions if it is to function as a handle (Winston et al., 1987).

- *Separable*—if the parts can be physically disconnected, in principle, from the whole to which they are connected (Winston et al., 1987).

- *Homogeneous*—if the parts are similar to each other and to the whole to which they belong (Winston et al., 1987).

5. Evens (1988, p.25) mentioned the transitivity of part-whole relationships in geographical applications.

- *Contemporaneous*—whether the whole has all or most of its parts occurring at the same time.

These dimensions are especially useful when it is difficult to determine the proper semantic interpretation of a relationship. Table 1 summarizes these dimensions based on Chaffin et al. (1988). The dimensions are given for the part with respect to the whole; that is, a part can have a function in the whole, be separable, be homogeneous, and/or be contemporaneous.

2.1.4 Spatial Inclusion. Spatial inclusions describe situations where one object is surrounded by another but is not part of the thing that surrounds it; for example:

Customer is-in Territory
(1,1) (0,*)

Database Design Implications: Spatial Inclusion. The significant database implication of spatial inclusion is that the included object can not be in more than one of the including objects at the same time. Therefore, the maximum cardinality on the left-hand side entity type must be 1.

2.2 Relationships Similar to Meronymic

Winston et al. (1987) identify other relationships that can easily be confused with meronymic relationships. These are possession, attachment, and attribution.

2.2.1 Possession. This is the ownership relationship. Two initial cases can be distinguished as illustrated by the following examples:

Case 1:
Person possesses Driver's License
(0,1) (1,1)
Case 2:
Driver possesses Driver's License
(1,1) (1,1)

In the first, more general, relationship, possession of a driver's license is optional for *Person* as reflected by the minimum cardinality of 0 for the *Person* entity type. In the second relationship it is, at least in principle, compulsory for a *Driver* to have a *Driver's License;* in fact, a driver's license is definitional for *Driver.* In both examples, the *Driver's License* uniquely identifies the possessor. In the first, optional relationship *Person* does not uniquely identify *License.* In the compulsory relationship, however, *Driver* does identify uniquely *Driver's License.* Because possession is a quasi-legal concept, the possessor can only be a person or other entity type with legal standing such as a corporation. It is also possible for something to be possessed by more than one person (or corporation, etc.). For example:

Table 1. Meronymic relationships

Relationship	Example	Function	Separable	Homogeneous	Contemporaneous
Component–Object	*Computer part-of Office* (0,1) (0,∗)	√	√	-	√
Feature–Event	*Demonstration part-of Presentation* (0,∗) (0,∗)	√	√	-	-
Member–Collection	*Employee part-of Committee* (0,∗) (1,∗)	-	√	-	√
Portion–Mass	*Module part-of Program* (0,∗) (1,∗)	-	√	√	√
Phase–Activity	*Billing part-of Consulting* (1,1) (1,∗)	√	-	-	-
Place–Area	*Reception Area part-of Office* (1,1) (0,1)	-	-	√	√
Stuff–Object	*Metal part-of Desk*	-	-	-	√

(Based on Chaffin et al., 1988.)

Case 3:
Person possesses House
(0,∗) (0,∗)

This example illustrates that, in general, one cannot automatically assign cardinalities to the possession relationship.

Database Design Implications: Possession.

- If the thing being possessed is definitional for the possessor, then the minimum cardinality of the possessor entity type must be 1.

- Everything must be possessed by at least one person or thing which would imply that the minimum value of the *B* entity type might be 1. This, however, can only be used as a design heuristic because, as illustrated above, a *House* might not be possessed by a person (although it could be by a corporation, government body, etc.). Only if the union of all the *A*'s that could possibly possess *B* was considered, could the minimum cardinality of the *B* entity type be 1; for example, *Employee (1,∗) possesses Benefit (1,∗).*

- In the *Person possesses Driver's License* example (Case 1), the relationship would be represented by making the key of *Person* a foreign key in *Driver's License.* In the second case, both cardinalities are (1,1) so the relationship could be represented by making the key of *Person* a foreign key in *Driver's License* or vice versa (or even both). The actual choice depends on the anticipated frequency of use in processing queries against the database. In the *Person possesses House* example (Case 3), the relationship should be represented by creating a separate relationship relation, because neither entity type has (1,1) min/max cardinalities. These three are shown below, illustrating the importance of understanding the semantics of a relationship.

Person: [SSN, name, address, date-of-birth, phone]
Driver's License: [LICENSE#, date-issued]
House: [LOCATION, description]

Representation in a Relational Database:

Case 1:
Driver's License: [LICENSE#, date-issued, ssn]

Case 2:
Driver's License: [LICENSE#, date-issued, ssn]
or
Person: [SSN, name, address, date-of-birth, phone, license#]

Case 3:
Person-possesses-House:
[SSN, HOUSE-LOCATION, date-acquired]

- Possession can easily be confused with attribution; for example, *Person possesses Name,* instead of "name" is an attribute of *Person.*

- The possession relationship is often expressed by the verb "has" (Pitha, 1972) in which case it may easily be confused with other relationships as discussed later in this article.

2.2.2 Attachment. In an attachment relationship, one entity type is connected or joined to another:

Handle attached to Mug
(1,1) (1,1)

It is possible to distinguish attachments between an entity type and one of its parts, such as *Fingers attached to Hand,* from attachments between independent entity types (Winston et al., 1987). An example of an attachment between independent entity types is *Modem (0,1) attached to Computer (0,∗).*

Database Design Implications: Attachment. When the attachment is between independent entity types, the minimum cardinality of both entity types must be 0 to reflect the fact that either can exist without the other. When one object is a part of another object to which it is attached, attachment is equivalent to component-object or aggregation; for example, an engine is attached to a car, but it is also part of a car. This, then, provides a very simple way for a system to determine, from a user, whether a relationship should be attachment or component-object.

2.2.3 Attribution. Attribution is the relationship between an object and its attribute, e.g., *Car has Color.* This is the standard attribute association that is found throughout the database literature.[6]

Database Design Implications: Attribution. An attribute can be associated with either an entity type or a relationship. In a relational data model, attributes appear as either key or non-key fields. Confusion may arise when a user provides, as an attribute, something that should be a component (Kuczora and Cosby, 1989) or some other kind of meronymic relationship (e.g., *Car has Color* [attribute] versus *Car has Engine* [component]). To test for this, a system should ask the user if color, for example, is a characteristic or property of all cars and if it does *not* have properties of its own.[7]

2.3 Case Relationships

Many events in the real world are structured according to scripts or plans (Schank and Abelson, 1977; Chaffin et al., 1988) that are used to account for knowledge of the everyday world. Such knowledge includes expectations that result from experience (Chaffin and Herrmann, 1987) and can be represented by case relationships. This type of semantic relationship describes things that are typically true in the real world; for example, things that agents use or activities they perform. Sometimes, the case relationship is referred to as a "use" relationship (Landis et al., 1987). Six different types of case relationships were identified by Chaffin and Herrmann (1984). Three involve agents, two involve actions, and one is concerned with invited attributes.

2.3.1 Case Relationships Involving Agents. The three types of relationships involving agents are *agent-action, agent-instrument,* and *agent-object.* Agent-action relationships take place between an agent and the action that it usually performs (e.g., *Consultant performs Consulting* and *Programmer performs Programming*). Agent-instrument relationships take place between an agent and the instrument that it uses

6. Users, however, may have difficulty distinguishing whether something should be classified as an entity, attribute, or relationship (Goldstein and Storey, 1990).

7. Teorey et al. (1986) compiled a set of guidelines for distinguishing whether something should be an entity or an attribute.

(e.g., *Programmer uses Computer* and *Janitor uses Mop*). Agent-object relationships take place between the agent and the object that the agent uses or makes (e.g., *Consultant uses Billing-Rates* and *Carpenter uses Lumber*).

Database Design Implications: Agent Case Relationships. The agent could be represented by an entity type and the action, instrument, and object as attributes of the entity type:

Consultant: [NAME, ..., action-performs, instrument-uses, object-uses]

There could also be a group of agents represented by one entity type (e.g., a group of employees):

Employee: [TYPE, name, action-performs, instrument-uses, object-uses]

2.3.2 Case Relationships Involving Actions. The two types of relationships involving actions are: (1) *action-recipient;* and (2) *action-instrument.* Action-recipient relationships take place between the action and the thing that receives that action (e.g., *Bill-Client* and *Supervise-Employee*). In an entity-relationship model, these could be *Company bills Client* and *Manager supervises Employee,* respectively (implying that more information is needed about the agent who performs the action). Action-instrument relationships take place between the action and the instrument that is used in that action (e.g., *Producing uses Machine, Analyzing uses Program,* and *Computing uses Computer*).

Database Design Implications: Action Case Relationships. These two types of relationships could be included in a database design by representing the action as an entity type and the recipient and instrument as its attributes:

Action: [TYPE, recipient, instrument]

2.3.3 Case Relationships Involving Invited Attributes. An invited attribution relationship takes place between a thing (an entity) and an attribute that is likely to be associated with it (e.g., *Employee is Skilled* and *Manager is Authoritative*).

Database Design Implications: Invited Attributes. Practically every entity type could have an "invited-attribute" that would reflect a typical property of that entity type. The attributes would be treated in the standard manner:

Employee: [EMP#, ..., type-of-skill]
Occurrences:
Employee: [13241, ..., computer-skills]
Employee: [44699, ..., supervision-of-others]

2.3.4 Discussion: Application of Case Relationships. Case relationships are used to represent things that typically occur in an application about agents, objects, and properties. In general, there are no special means in database design for capturing the case relationships of an application. As illustrated above, however, such relationships can easily be accommodated using a combination of entity types and attributes. In their work on the implementation of meronymic relationships, Kuczora and Cosby (1989) suggested that understanding semantic relationships is important for computer systems with some common-sense reasoning capabilities. If a database design system contained such knowledge in the form of case relationships, it might be possible for it to generate relationships automatically, possibly subject to verification by the user. Ideally, this would address the need to include more real-world knowledge in a relational database (Reiter, 1984). For example, a system might generate: *Employee uses Workstation* or *Programmer uses Computer.* A system might also be able to generate some invited (or common) attributes of an entity type. Examples include "profit-level" as an attribute of *Company* and "yrs-experience" as an attribute of *Employee.* These capabilities would be useful because they could help to shorten the development time.

2.4 Antonyms

Antonyms (opposites) could occur in attributes, entities, or relationships. For example, the attribute "work-status" can have only the values "full-time" or "part-time." Two entity types will be mutually exclusive if they are antonyms (e.g., *Male-Employee* and *Female-Employee*). Relationships of the form A *verb phrase*$_1$ B and A *verb phrase*$_2$ B could be expressing information that is opposite in meaning; e.g., *Borrowers borrow Books* and *Borrowers return Books.*

Database Design Implications: Antonyms.

- Identifying attribute antonyms involves examining the actual *values* in a database. Integrity constraints are required to ensure that the corresponding attribute values are mutually exclusive. These *domain* or *value* constraints could be identified during the database design process, simply by examining typical values of the attributes.

- An integrity constraint is required to ensure that occurrences of opposite entity types do not appear simultaneously in a database. For example: a single occurrence of *Employee* cannot be an occurrence of both a *Male-Employee* and a *Female-Employee.*

- An occurrence of an entity type should not appear in mutually exclusive relationships at the same time. Given the relationships, *Guest arrives-at Hotel* and *Guest departs Hotel,* an occurrence of *Guest* should not appear in both of these relationships (or their relational representations) simultaneously.

- It is possible that only one of two relationships that are antonyms–*Passenger starts Flight* and *Passenger ends Flight*–is required. As a heuristic, the design should be checked to determine whether one or both are necessary.

2.5 Synonyms

As with antonyms, synonyms (same or nearly the same) can occur among attributes, entities, or relationships. For example, the attributes "job" and "function" of *Employee* could be synonymous. The entity types *Employee* and *Worker* are synonyms. Finally, synonymous relationships can often be found by examining sets of relationships of the form *A verb phrase*$_1$ *B* and *A verb phrase*$_2$ *B* (or *B verb phrase*$_2$ *A*), as in *Professor teaches Course* and *Course taught-by Professor.*

Database Design Implications: Synonyms.

- For synonymous attributes, one of the attributes should be deleted from the entity type's definition. An automated tool could benefit from a dictionary of synonyms which is either built into the system initially, or developed over time as the system is applied to different database design problems.

- When entities are synonyms, one of the entity types should be retained and the other deleted. Any relationships in which the name of a synonymous entity type appears should be changed to the name of the entity type that was retained. Redundant relationships identified in this process should be deleted. For example:

Relationships:
Employee assigned-to Project
Worker assigned-to Project
Project requires Worker

Become (Employee chosen over Worker):
Employee assigned-to Project
Project requires Employee

- In Example 1 below, the two relationships are synonymous so one should be removed. In Example 2, however, the relationships could have different meanings so both should be retained.

Example 1:
Manager manages Employee
Manager supervises Employee

Example 2:
Employee assigned-to Project
Project requires Employee

- Relationships of the form *A verb phrase B* and *C verb phrase B* could suggest that an *is-a* relationship is missing. For example, given the relationships:
  Employee assigned-to Project
  Project-Leader assigned-to Project

the design needs to be checked to determine if there is a missing relationship; in this case, *Project-Leader is-an Employee* is also required. This heuristic stems from the fact that a specific entity type in an *is-a* relationship can inherit the relationships in which its corresponding generic entity type occurs.

2.6 Summary of Classes of Relationships

The various types of semantic relationships and ways in which they might be identified in a real world application are summarized in Table 2.

3. General Relationships

Some relationships are very general, leading to ambiguity in a design, particularly, *have/has* relationships of the form *A have B* or *A has B.* Although this relationship is often used in the entity-relationship model, it can easily be seen to have multiple interpretations (Storey, 1988). The verb phase *have/has* can be used to convey either an attribute, or a relationship in which the best interpretation could be almost any of the semantic relationships previously discussed. It would, therefore, be beneficial to use the interpretation that best reflects the semantics of the application and reserve the general form of this relationship for situations where it is not possible to clarify the semantics further. Various interpretations and types of difficulties associated with the *have/has* relationship are outlined below. These also illustrate why it is important to understand the semantics of an application and the sometimes subtle differences among the relationships.

Part-Whole. *Have/has* relationships can be used to represent part-whole relationships: "*X* is a meronym (part-whole) of *Y* if and only if sentences of the form *A Y has X's/an X* and *An X is a part of a Y* are normal when the noun phrases *an X, a Y* are interpreted generically" (Cruse, 1986). Thus, it is reasonable to expect that a user might express *part-whole* relationships as *have/has* relationships: *Office has Desk* versus *Desk part-of/component-of Office. Have/has* can be used to express other types of part-whole relationships. *Has* can also easily confuse attribution with part-whole relationships. For example, the relationships *Car has Color,* and *Car has Engine,* although of the same format, express an attribute and a component-object relationship, respectively (Kuczora and Cosby, 1989).

Possession and Attribution. The relationships most often associated with *have/has* are those of possession and attribution (Pitha, 1971; 1972). When the correct interpretation is attribution, the attribute should be included in the definition of

Table 2. Modeling with semantic relationships

Real world situation	Semantic relationship	Database design implications
Classes of things	Class inclusion (*is-a* relationships, hierarchies, lattices)	Inheritance from supertype to subtype
Something along with its components, parts	Meronymic or part-whole relationships	Partial inheritance for some relationships; distinguished by: function, separable, homogeneous, contemporary
One thing included or surrounded by another	Spatial inclusion	Maximum cardinality of included entity type is 1
One thing possesses or owns another	*Possessor owns possesses/has) Possession*	If possession definitional for possessor, then minimum cardinality of possessor is 1
One thing connected to another	Attachment	When thing attached is also a part, this is equivalent to aggregation or component/ object relationship
A property or characteristic of something	Attribute	Both entity types and relationships can have attributes
Typical, opposite, and similar relationships	Case, antonym, and synonym relationships	Case relationships capture real-world situations; synonyms should be removed; antonym sets should be checked to determine if both are needed

the corresponding entity type. For the possession interpretation, the relationship should be renamed; for example, *Company has Assets* would be more accurately modeled as *Company possesses Assets.*

Reversal of Have/Has. *Have* can often be reversed (Zoeppritz, 1981):

If: *x has daughter y;*
then: *y has parent x.*

The database design implication is that a relationship should be examined carefully to determine which would be a "more natural" way to express something:

If: *Company has employee Manager;*
then: *Manager has employment from Company.*

This could also be captured in an entity-relationship diagram by including the roles that the entity types play in a relationship.

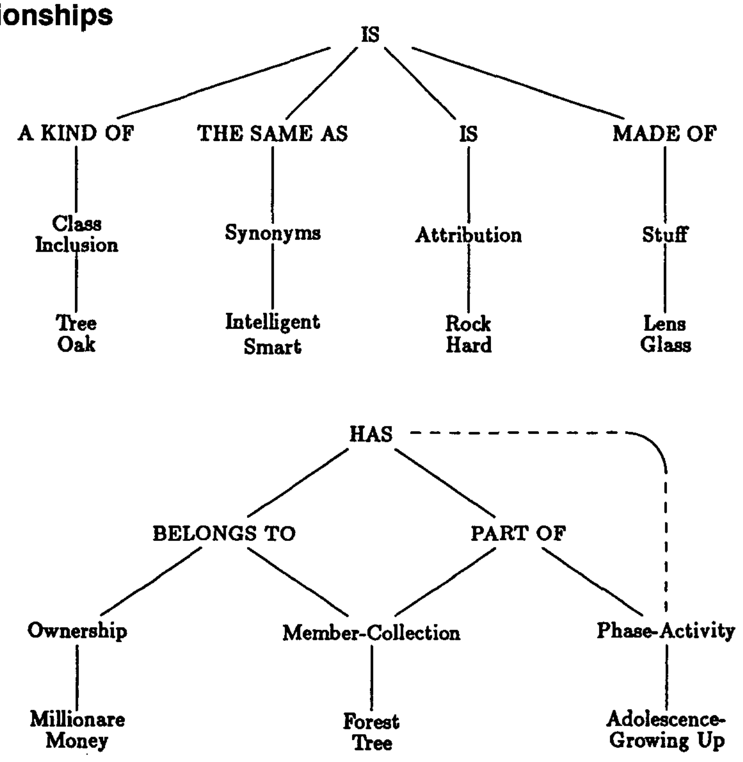
Have/Has versus Is-A. *Has* can be confused with *is-a* (Chaffin and Herrmann, 1988). *Is* can be used to express class inclusion, synonym, attribution, and stuff relationships. These relationships are organized in a hierarchical fashion. *Has* can be used to express *belongs-to* and some *part-whole* relationships. The *has* relationship, however, is not organized in a strictly hierarchical fashion. For example, a phase-activity relationship can be expressed by *part-of,* but not by *has.* A collection-member relationship can be expressed by both *belongs-to* and *part-of* Being aware of this hierarchical distinction can help to determine which interpretation is appropriate. *Is* and *have* relationships are compared in Figure 2 (Chaffin and Herrmann, 1988), where the non-hierarchical feature is indicated by a dashed line.

Others. *Have/has* can be used to express a variety of other (semantic) relationships including belongs-to (Chaffin and Herrmann, 1988), inherited, introduced, and contains (Maschtera, 1987):

Interpretation	Example
Inherited	*Part-Time Employee has (inherits) Employee-Benefits* (Part-time employees are employees; they inherit (some-of) the benefits of employees.)
Introduced	*Store has/introduces Fashion-Line*
Belongs-to	*Committee-Member has (belongs-to) Committee* (Member–Collection)
Contains	*Store has (contains) Merchandise*

There are a number of other interpretations of *have/has*: hold; retain; possess; keep; and maintain (Table 3). Most of the interpretations resemble the possession

Figure 2. Two examples of quasi-hierarchical organization of relationships



relationship in the sense that if all of the entity types of the thing that is obligated, obtained, shown, etc., are considered, then the minimum cardinality of that entity type should be 1. For example, something is not deserved, unless it is deserved by at least one person or thing; something is not demonstrated or shown, unless it is demonstrated or shown by someone or something. This observation only holds true, however, if all of the entity types that could do the deserving, holding, demonstrating, etc., are considered. Hence, this can only serve as a design heuristic.

3.1 Discussion

For database design purposes, the most common interpretations of *have/has* relationships appear to be: possession, attribution, and some part-whole. Obtaining the most appropriate interpretation is necessary to determine the correct min/max cardinalities and, hence, the relationship's proper representation in a relational model. At the very minimum, the verb phrase of the relationship should be changed to one that reflects most accurately the application. Using a more refined interpretation should result in a database that is easier to use.

4. The Semantic Relationship Analyzer

A prototype system, the Semantic Relationship Analyzer, has been developed that implements the preceding analysis.[8] The system elicits relationships of the form *A verb phrase B* from a user who is either a database designer or an end-user. For each relationship, the Semantic Relationship Analyzer obtains the min/max cardinalities and then tries to capture the best interpretation for the relationship. For example, because a *have/has* relationship is ambiguous, the system determines whether such a relationship would be better expressed by *part-of* or one of the other relationships that can easily be confused with *have/has.* While doing so, it checks whether the min/max cardinalities, as provided by the user, are correct, and, modifies them according to what it "knows" about the semantics of the relationship. The system makes inferences about attributes, keys, integrity constraints, and missing relationships and then determines how to represent the relationship in a relational model. Synonymous relationships are identified and redundant relationships deleted. The final output is a relational model of the user's application. A sample session using the prototype system was described by Storey (1992*b*).

4.1 Have/Has and Meronymic Relationships

If the user provides a *have/has* relationship, the Semantic Relationship Analyzer explains that the verb phrase *have/has* is subject to multiple interpretations and starts to refine the relationship by asking the user if it can be expressed using *part-of.* If so, the system proceeds to find the best interpretation of *part-of.* It may also check if *have/has* has been confused with *is-a* (by asking the user).

System:	Enter next relationship.
User:	office has supplies ...
System:	Is it correct to say that: "supplies" is part of "office"?
User:	yes

If the user does not believe that the relationship can be expressed using *part-of,* the system simply displays all of the possible interpretations of *have/has* (possession, attribution, different kinds of, part-of, etc.) and asks the user to identify which one is a suitable interpretation. If the user indicates that none of the interpretations is appropriate, the relationship remains unchanged.

A *part-of* relationship is either provided directly by the user or obtained as a result of refining a *have/has* relationship. The Semantic Relationship Analyzer tries to determine the most appropriate interpretation of part-of by leading the user through a series of questions that are designed to be easy for the user to understand

8. The system is written in Arity Prolog and runs on a PC.

Table 3. Have/Has Interpretations

Interpretation	Example
Obligation	Buyer has-to Pay (0,*) (1,1) Seller has-to Deliver (0,*) (1,*)
Obtain, receive	Company has Permit (0,*) (1,*)
Feature	Product has Features (0,*) (1,*)
Show/Demonstrate	Employeee has Skill (1,*) (1,*)
Deserve	Student has Degree (0,*) (1,1)
Hold	Employee has Shares (0,*) (0,1)
Retain/Keep	Company has Experts (0,*) (1,1)
Maintain	Company has Records (1,*) (1,1) Employee has Contacts (0,*) (1,*)

and respond to. Relationships that can easily be confused with *part-of* are also included in this process.

System: "supplies part-of office"
Is "supplies" something that an office possesses? ...

The order in which the system asks the user questions about the various possible interpretations of a relationship was carefully chosen to minimize the amount of confusion that might exist for the user. The system does so by trying to deal with the interpretations that could be the greatest source of confusion first.

If the system cannot classify a *part-of* relationship, it asks the user specific questions to determine the relationship's characteristics. It then compares them to those given in Table 1 and, from this, suggests what the most appropriate interpretation might be.

System: "reception-area part-of office"
Does the "reception-area" need to be
in a particular place (location) or
occur at a special time in order to support
a particular function that it plays in the office?

User: no . . .
(The system determines that a reception-area is homogeneous
and contemporary with the office and, therefore, this is a
place–area relationship)

4.2 Min/Max Cardinalities

When a relationship is identified by a user, the system obtains an initial set of min/max cardinalities by asking the user to select appropriate values for the number of occurrences of one entity type that could appear for each occurrence of the other. This is done by displaying a menu of possibilities and does not require the user to understand the concept of a cardinality.

System: "employee part-of committee"
How many occurrences of "committee"
are there for each occurrence of "employee"?

a. Exactly one.
b. At most one.
c. At least one.
d. Any number.
e. Do not know.

User: d.
(From this the system infers that the min/max
cardinalities for employee are (0,*) and asks
a similar question for committee.)

Then, when the system is attempting to determine the correct interpretation of a relationship, it tries to ascertain whether the min/max cardinalities that it obtained initially, were, indeed, the most appropriate ones (based on what it "knows" about the semantic relationship). If not, the system changes the cardinalities to correctly reflect the application. To illustrate, if the system establishes that a relationship is best interpreted as *possession,* it asks the user if the thing being possessed is definitional for the possessor. If, in the relationship *Employee possesses Skills,* it is assumed that possession of a skill is required to be an employee, then the minimum cardinality of Employee must be 1. If it is not, the system informs the user of the error and corrects it.

4.3 Inferences

The Semantic Relationship Analyzer tries to make inferences about attributes, keys, integrity constraints, and missing relationships. It does so to ensure that the design is complete and to detect errors in the user's input. For example, when a member–collection, or association, relationship is identified, the system elicits derived and a priori attributes and includes them in the final design. For *Employee member-of Committee,* the system might determine that an a priori attribute is "max-number" whereas derived attributes are "average-age" and "no-managers."

The system obtains key attributes for *portion-mass* and *is-a* relationships because it can make certain inferences about keys for these types of relationships. For portion-mass relationships, the system "knows" that the key of the portion (e.g., *Module*) should include the key of its mass (e.g., *Program.* The key of module is [PROGRAM#, MODULE#]). For *is-a* relationships, the key of the specific entity type can adopt the key of the generic entity type (e.g., because *Manager is-an Employee,* [EMP#] can serve as a key for both.)

The system includes, in its output, integrity constraints from spatial inclusion relationships, and relevant, characteristic, and identifying components of component-object relationships. For example, an integrity constraint from a spatial inclusion relationship could be: There is at most one "territory" for each "customer." It would be possible, of course, for the system to list all of the restrictions imposed by the min/max cardinalities as integrity constraints.

There are several ways in which the system checks for missing relationships. When a *component–object* relationship is identified, the system suggests that the user might wish to provide relevant, characteristic, and identifying relationships. For example, given that an *Office-manager* is a relevant component of an *Office,* the user is prompted to provide *Secretary* and *CEO* as characteristic and identifying components, respectively. When an attachment relationship, such as *Modem attached-to Computer* is identified, the system checks if *Modem* and *Computer* might be components of something else; in this case, *Workstation.* When checking for synonymous relationships, the system also checks for missing *is-a* relationships. For example, given the relationships *Manager assigned-to Projects* and *Employee assigned-to Projects,* the system checks whether *Manager* and *Employee* are synonyms, whether one is a subset of the other, or whether they are related in some other way. In this case, *Manager is-an Employee* would be added.

After all of the relationships have been identified, the Semantic Relationship Analyzer checks for patterns of possible synonymous relationships (e.g., *Employee assigned-to Project* and *Employee allocated-to Project*) and deletes redundant relationships. If the system knew all of the synonyms of each verb phrase, it would not need to query the user about them. Because it does not, it starts to build up a table of synonyms which, ideally, could be drawn on during future design sessions.

4.4 Output

The output from the Semantic Relationship Analyzer is a set of relations that represents the design of a relational model for the user's application. The relations consist of the three types: entity, extended entity, and relationship relations. Any attributes that have been identified during the session are included in the output; for example, those obtained from attribution relationships such as *Employee has Name.* The name of an entity type concatenated with the suffix "-KEY" is used as a surrogate for the key of that entity type (e.g., OFFICE-KEY). Semantic integrity constraints that are identified are also listed (e.g., *"There must be a minimum of one Secretary for each Company"*).

4.5 Extensions

There are, obviously, a large number of extensions that could be made to the Semantic Relationship Analyzer. First, this system is only a prototype. A better user interface is needed; for example, one that employs a menu-driven dialogue. The wording of the system's questions is meant to be easy for the user to understand; however, because there are subtle differences among some of the relationships, further refinement of the system's questions might be useful. For example, should the relationship *Computer has Modem* be interpreted as *Modem attached-to Computer* or *Modem component-of Computer*? This distinction may be difficult to make. The system assumes that the user is able to answer its questions correctly. Heuristics could be added that would enable the system to identify where the user either makes mistakes or does not answer the system's questions in the most appropriate manner. Such heuristics would identify places of possible confusion on the part of the user.

The system does not elicit either entity or relationship attributes. It is assumed that this would be done in the main part of any database design system; however, this capability could easily be added to the Semantic Relationship Analyzer. The system obtains the min/max cardinalities and then uses what it "knows" about certain semantic relationships to verify that these cardinalities are correct. Alternatively, the system could have been developed so that it carries out whatever analysis it can and then uses the results to generate the min/max cardinalities. The two approaches could be tested to determine if one is better (e.g., easier for the user) than the other.

One of the uses of knowledge-based systems is that they are able to provide some justification for their responses to the user. More capabilities could be provided that would allow the user to ask "why" the system is asking a question. This would better enable the user to judge what his or her answer should be before responding to a query from the system.

Only binary relationships are analyzed. Non-binary relationships (for example, ternary) should be included, too. Also, case relationships are not incorporated, but, as previously discussed, would be useful for a system with some domain-specific knowledge. Similarly, antonyms are not currently handled by the system. Domain-specific knowledge could be added that would enable the system to generate automatically some relationships that should be true in an application (possibly, subject to verification by the user). The various types of relationships discussed in this article should capture much of the semantics of an application. Therefore, one possibility would be to modify the system so that it retains information about these relationships as it moves from one application to another. Such information could be built up over time and applied to new design problems. It would, obviously, require a great effort to build such a capability into the system.

Finally, the system could be extended to include the various kinds of *is-a* relationships previously discussed: generalization and its inverse, specialization, and subset hierarchies.

5. Summary and Conclusions

Although certain well-known data abstractions are often used in data modeling, research in linguistics, logic, and cognitive psychology has identified many more semantic relationships. The objective of this article has been to analyze a number of these lesser-known relationships in terms of their design implications and ways they could be employed by database design systems to capture some of the semantics of an application.

Various classes of semantic relationships are examined: inclusion, possession, attachment, attribution, antonyms, synonyms, and case. *Have/has* relationships, which can easily be confused with these, are also discussed. The guidelines that result from the analysis include ways to recognize when certain classes of semantic relationships exist and how relationships from these classes should be represented in a relational data model. The importance of obtaining the correct min/max cardinalities, which can only be done by having a clear understanding of the application (and the relationships used to model that application), is highlighted.

It is shown that the verb phrase, *has,* can be ambiguous and should be changed to one that is more appropriate for the application. In some situations this might not have a direct impact on the structure of the final design, but changing the name of a relationship to reflect more of the application's semantics should make the database easier for the end-user to work with.

A prototype system, the Semantic Relationship Analyzer, is presented. It implements the analysis carried out in this research. The system engages a user in a dialogue that elicits the relationships of the user's application. These relationships are then analyzed for their most appropriate semantic interpretations. The system's capabilities include: correcting erroneous min/max cardinalities; detecting missing relationships; and inferring attributes and integrity constraints. A relational database design is obtained as the system's output, which is comprised of a set of relations, along with some attributes and semantic integrity constraints.

It appears that the only way certain semantic relationships (e.g., case relationships) could be really useful is if they are included in a database design system that contains some real world or domain-specific knowledge. Current systems, unfortunately, do not have such capabilities, suggesting that further research is required in this area. Future research could also examine other semantic relationships from research in linguistics, cognitive psychology, and software engineering to determine if there are more relationships that could be identified and incorporated into an automated tool for database design that has some understanding of real world semantics.

It is hoped that the Semantic Relationship Analyzer, or its knowledge base, could eventually be incorporated into a database design system to assist the user in providing the best input. Ideally, this would result in more of the semantics of the real world being captured in the final design.

Acknowledgments

This research was supported by the William E. Simon Graduate School of Business Administration, University of Rochester. The author wishes to thank the anonymous reviewers for their insightful suggestions on an earlier version of this article. The author also wishes to thank Robert C. Goldstein, and Yair Wand, University of British Columbia, for useful discussions on this research. The assistance of Lucie Tepla, Olin Fellow, William E. Simon Graduate School of Business Administration, is gratefully acknowledged.

References

Blaha, M.R., Premerlani, W.J., and Rambaugh, J.E. Relational database design using an object-oriented methodology. *Communications of the ACM,* 31(4):414-427, 1988.

Bouzeghoub, M., Gardarin, G., and Metais, E. Database design tools: An expert system approach. *Proceedings of the 11th International Conference on Very Large Databases,* Stockholm, Sweden, 1985.

Brachman, R.J. What IS-A is and isn't: An analysis of taxonomic links in semantic networks. *Computer,* 16(10):30-36, 1983.

Brodie, M. Association: A database abstraction. In: Chen, P., ed. *Approach to Information Modeling and Analysis.* Amsterdam: North-Holland, 1981, pp. 583-608.

Brodie, M. On the development of data models. In: Brodie, M.L., Mylopoulos, J., and Schmidt, J.W., eds. *On Conceptual Modelling.* Berlin: Springer-Verlag, 1984, pp. 19-47.

Cauvet, C., Proix, C., and Rolland C. Information systems design: An expert system approach. In: Schmidt, J.W., Ceri, S., and Missikoff, M., eds. *Advances in Database Technology: Proceedings of the International Conference on Extending Database Technology,* In: Goss, G. and Hartmanis, J., eds. *Lecture Notes in Computer Science,* Berlin: Springer-Verlag, 1988, pp.113-133.

Chaffin, R., and Herrmann, D.J. The nature of semantic relations: A comparison of two approaches. In: Evens, M., ed. *Relational Models of the Lexicon: Representing Knowledge in Semantic Networks.* New York: Cambridge University Press, 1988, pp. 289-334.

Chaffin, R., and Herrmann, D.J. Relation elementy theory: A new account of the representation and processing of semantic relations. In: Gorfein, D. and Hoffman, R., eds. *Memory and Learning: The Ebbinghaus Centennial Conference.* Hillsdale, N.J.: Lawrence Erlbaum Associates Inc., 1987. pp. 221-245.

Chaffin, R., and Herrmann, D.J. The similarity and diversity of semantic relations. *Memory and Cognition,* 13(2):134-141, 1984.

Chaffin, R., Herrmann, D.J., and Winston, M. An empirical taxonomy of part-whole relations: Effects of part-whole type on relation identification. *Language and Cognitive Processes,* 3(1):17-48, 1988.

Chen, P.P. The entity-relationship model: Toward a unified view of data. *ACM Transactions On Database Systems,* 1(1):9-36, 1976.

Codd, E.F. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems,* 4(4):397-434, 1979.

Cruse, D.A. *Lexical Semantics,* New York: Cambridge University Press, 1986.

Dahlgren, K. *Naive Semantics for Natural Language Understanding.* Boston, MA: Kluwer Academic Publishers, 1988.

Davis, J.P. and Bonnell, R.D. Modeling semantics with concept abstraction in the EARL data model, *Proceedings of the Eighth International Conference on Entity-Relationship Approach,* Toronto, Canada, 1989.

Demo, B. and Tilli, M. Expert system functionalities for database design tools. In: Sriram, D. and Adey, R., eds. *Applications of Artificial Intelligence in Engineering Problems: Proceedings of the 1st International Conference, Southampton University.* Berlin: Springer-Verlag, 1986, pp. 1073-1082.

Dogac, A., Yuruten, B., and Spaccapietra, S. A generalized expert system for database design. *IEEE Transactions on Software Engineeering,* 15(4):479-491, 1989.

Dos Santos, C.S., Neuhold, E.J., and Furtado, A.L. A data-type approach to the entity- relationship model. In: Chen, P.P., ed. *Proceedings of the Entity-Relationship Approach to Systems Analysis and Design.* Amsterdam: North-Holland, 1980, pp. 103-119.

Elmasri, R. and Navathe, S. *Fundamentals of Database Systems.* City?: Benjamin/ Cummings Publishing Co., Inc., 1989.

Evens, M. Introduction. In: Evens, M., ed. *Relational Models of the Lexicon: Representing Knowledge in Semantic Networks.* New York: Cambridge University Press, 1988, pp. 1-37.

Goldstein, R.C. and Storey, V.C. Unravelling Is-A structures. *Information Systems Research,* 3(2):99-126, 1992.

Goldstein, R.C. and Storey, V.C. Data Abstractions. Working Paper, University of Rochester, 1991.

Goldstein, R.C. and Storey, V.C. Some findings on the intuitiveness of entity-relationship constructs. In: Lochovsky, F., ed. *Entity-Relationship Approach to Database Design and Querying: Proceeding of the 8th International Conference,* Toronto, Canada. City?: North Holland, 1990, pp. 9-23.

Hull, R. and King, R. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys,* 19(3):201-260, 1987.

Kawaguchi, A., Taoka, N., Mizoguchi, R., Yamaguchi, T., and Kakusho, O. An intelligent interview system for conceptual design of database. *Proceedings of the European Conference on Artificial Intelligence,* 1986.

Kim, W. Object-oriented databases: Definition and research directions. *IEEE Transactions on Knowledge and Data Engineering,* 2(3):327-341, 1990.

Kuczora, P.W., and Cosby, S.J. Implementation of meronymic (part-whole) inheritance for semantic networks. *Knowledge-Based Systems,* 2(4):219-227, 1989.

Landis, T.Y., Herrmann, D.J., and Chaffin, R. Development differences in the comprehension of semantic relations. *Zeitschrift für Psychologie,* 195(2):129-139, 1987.

Lloyd-Williams, M. Expert systems for database design. Working Paper, Department of Computer Studies, Polytechnic of Wales, February 1991.

Maschtera, U. Modeling support environments. In: Sol, H.G., et al., eds. *Expert Systems and Artificial Intelligence in Decision Support Systems,* 1987, pp.151-173.

Mattos, N.M. Abstraction concepts: The basis for data and knowledge modelling. *Proceedings of the E-R Conference,* 1988.

Mattos, N.M., and Michels, M. Modeling with KRISYS: the design process of DB applications reviewed. *Proceedings of the Eighth International Conference on Entity-Relationship Approach,* Toronto, Canada, 1989.

Peckham, J. and Maryanski, F. Semantic data models. *ACM Computing Surveys,* 20(3): 153-189, 1988.

Pitha, P. Remarks on possessivity II. *Prague Bulletin of Mathematical Linguistics,* 17:25-36, 1972.

Pitha, P. Remarks on possessivity. *Prague Bulletin of Mathematical Linguistics,* 16:33-46, 1971.

Potter, W.D. and Kerschberg, L. A unified approach to modeling knowledge and data. In: Meersman, R.A. and Sernadas, A.C., eds. *Data and Knowledge.* Amsterdam: Elsevier Science Publishers, 1988, pp. 265-291.

Potter, W.D. and Trueblood, R.P. Traditional, semantic, and hyper-semantic approaches to data modeling. *IEEE Computer,* 1988, pp. 53-62.

Proix, C. and Rolland C. A knowledge base for information system design. In: Meersman, R.A. and Sernadas, A.C., eds. *Data and Knowledge.* Amsterdam: Elsevier Science Publishers, 1988, pp. 293-306.

Ram, S. Automated tools for database design: State of the art. Working Paper, Department of Management Information Systems, College of Business and Public Administration, University of Arizona, 1989.

Reiter, R. Towards a logical reconstruction of relational database theory. In: Brodie, M.L., Mylopoulos, J., and Schmidt, J.W., eds. *On Conceptual Modelling.* New York: Springer-Verlag, 1984, pp. 191-233.

Rolland, C. and Proix, C. An expert system approach to information system design. In: Kugler, H.J. *Information Processing 86,* City: Elsevier Science Publishers B.V. (North-Holland), 1986, pp. 241-250.

Schank, R.C. and Abelson, R.P. *Scripts, Plans, Goals, and Understanding.* Hillsdale, N.J.: Lawrence Erlbaum Associates Inc., 1977.

Schubert, L.K., Papalaskaris, M. A., and Taugher, J. Determining type, part, color, and time relationships. *IEEE Computer,* pp. 53-60, 1983.

Shaw, M. The impact of modelling and abstraction concerns on modern programming languages. In: Brodie, M.L., Mylopoulos, J., and Schmidt, J.W., eds. *On Conceptual Modelling.* New York: Springer-Verlag, 1984, pp. 49-83.

Smith, J.M. and Smith, C.P. Database abstractions: Aggregation and generalization. *ACM Transactions on Database Management Systems,* 2(2):105-133, 1977.

Storey, V.C. View creation: An expert system for database Design. Ph.D. Dissertation, Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, Canada, Washington, D.C.: ICIT Press, 1988.

Storey, V.C. Relational database design based on the entity-relationship model. *Data and Knowledge Engineering,* North-Holland, 7(1):47-83, 1991*a*.

Storey, V.C. Meronymic relationships. *Journal of Database Administration,* 2(3)22-35, 1991*b*.

Storey, V.C. Real world knowledge for databases. *Journal of Database Administration,* 3(1):1-19, 1992*a*.

Storey, V.C. Understanding semantic relationships: Theory and implementation. Technical Report, University of Rochester, 1992*b*.

Storey, V.C. The use of artificial intelligence methods for database design systems. *Data and Knowledge Engineering,* 1993, in press.

Storey, V.C. and Goldstein, R.C. A methodology for creating user views in database design. *ACM Transactions on Database Systems,* 13(3):305-338, 1988.

Storey, V.C. and R.C. Goldstein. Design and development of an expert database design system. *International Journal of Expert Systems: Research and Applications,* 3(1):31-63, 1990*a*.

Storey, V.C. and R.C. Goldstein. An expert view creation system for database design. *Expert Systems Review,* 2(3):19-45, 1990*b*.

Storey, V.C. and Goldstein, R.C. Knowledge-based approaches to database design. *Management Information Systems Quarterly,* 17(1):25-46, 1993.

Tauzovich, B. An expert system for conceptual data modelling. *Proceedings of the Eighth International Conference on Entity-Relationship Approach,* Toronto, Canada, 1989.

Teorey, T.L., Yang, D., and Fry, J.P. A logical design methodology for relational databases using the extended entity-relationship model. *Computing Surveys,* 18(2), 1986.

Tsichritzis, D. and Lochovsky, F. *Data Models.* New York: Prentice-Hall, 1982.

Ullman, J. Discussion. In: Brodie, M.L. and Mylopoulos, J., eds. *On Knowledge Base Management Systems,* Berlin: Springer-Verlag, 1986, pp. 197-198.

Urban, S.D. and Delcambre, L.M.L. An analysis of the structural, dynamic, and temporal aspects of semantic data models. *Proceedings of the International Conference on Data Engineering,* 1986.

Wagner, C. View integration in database design, Ph.D. Dissertation, Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, Canada, 1989.

Winston, M.E., Chaffin, R., and Herrmann, D. A taxonomy of part-whole relations. *Cognitive Science,* 11:417-444, 1987.

Zoeppritz, M. The meaning of OF and HAVE in the USL system. *American Journal of Computational Linguistics,* 7(2):109-119, 1981.