Semantic Database Modeling: Survey, Applications, and Research Issues

RICHARD HULL

Computer Science Department, University of Southern California, Los Angeles, California 90089-0782

ROGER KING

Computer Science Department, University of Colorado, Boulder, Colorado 80309

Most common database management systems represent information in a simple record-based format. Semantic modeling provides richer data structuring capabilities for database applications. In particular, research in this area has articulated a number of constructs that provide mechanisms for representing structurally complex interrelations among data typically arising in commercial applications. In general terms, semantic modeling complements work on knowledge representation (in artificial intelligence) and on the new generation of database models based on the object-oriented paradigm of programming languages.

This paper presents an in-depth discussion of semantic data modeling. It reviews the philosophical motivations of semantic models, including the need for high-level modeling abstractions and the reduction of semantic overloading of data type constructors. It then provides a tutorial introduction to the primary components of semantic models, which are the explicit representation of objects, attributes of and relationships among objects, type constructors for building complex types, ISA relationships, and derived schema components. Next, a survey of the prominent semantic models in the literature is presented. Further, since a broad area of research has developed around semantic modeling, a number of related topics based on these models are discussed, including data languages, graphical interfaces, theoretical investigations, and physical implementation strategies.

Categories and Subject Descriptors: H.0 [**Information Systems**] General; H.2.1 [**Database Management**] Logical Design—*data models*; H.2.2 [**Database Management**] Physical Design—*access methods*; H.2.3 [**Database Management**] Languages—*data description languages (DDL); data manipulation languages (DML); query languages*

General Terms: Design, Languages

Additional Key Words and Phrases: Conceptual database design, entity–relationship model, functional data model, knowledge representation, semantic database model

INTRODUCTION

Commercial database management systems have been available for two decades, originally in the form of the hierarchical and network models. Two opposing research directions in databases were initiated in the early 1970s, namely, the introduction of the relational model and the development of semantic database models. The relational model revolutionized the field by separating logical data

CONTENTS

representation from physical implementation. Significantly, the inherent simplicity in the model permitted the development of powerful, nonprocedural query languages and a variety of useful theoretical results.

The history of semantic modeling research is quite different. Semantic models were introduced primarily as schema design tools: A schema could first be designed in a high-level semantic model and then translated into one of the traditional models for ultimate implementation. The emphasis of the initial semantic models was to accurately model data relationships that arise frequently in typical database applications. Consequently, semantic models are more complex than the relational model and encourage a more navigational view of data relationships. The field of semantic models is continuing to evolve. There has been increasing interest in using these models as the bases for full-fledged database management systems or at least as complete front ends to existing systems.

The first published semantic model appeared in 1974 [Abriel 1974]. The area matured during the subsequent decade, with the development of several prominent models and a large body of related research efforts. The central result of semantic modeling research has been the development of powerful mechanisms for representing the structural aspects of business data. In recent years, database researchers have turned their attention toward incorporating the behavioral (or dynamic) aspects of data into modeling formalisms; this work is being heavily influenced by the object-oriented paradigm from programming languages.

This paper provides both a survey and a tutorial on semantic modeling and related research. In keeping with the historical emphasis of the field, the primary focus is on the structural aspects of semantic models; a secondary emphasis is given to their behavioral aspects. We begin by giving a broad overview of the fundamental components and the philosophical roots of semantic modeling (Section 1). We also discuss the relationship of semantic modeling to other research areas of computer science. In particular, we discuss important differences between the constructs found in semantic models and in object-oriented programming languages. In Section 2 we use a Generic Semantic Model to provide a detailed, comprehensive tutorial that describes, compares, and contrasts the various semantic constructs found in the literature. In Section 3, we survey a number of published models. We conclude with an overview of ongoing research directions that have grown out of semantic modeling (Section 4); these include database systems and graphical interfaces based on semantic models and theoretical investigations of semantic modeling.

Semantic data models and related issues are described in the earlier survey article by Kerschberg et al. [1976] by Tsichritzis and Lochovsky [1982], and the collection of articles that comprise Brodie et al. [1984]. Also, Afsarmanesh and McLeod [1984], King and McLeod [1985b], and

Maryanski and Peckham [1986] present taxonomies of the more prominent models, and Urban and Delcambre [1986] survey several semantic models, with an emphasis on features in support of temporal information. The dynamic aspects of semantic modeling are emphasized in Borgida [1985]. The overall focus of the present paper is somewhat different from these other surveys in that here we discuss both the prominent semantic models and the research directions they have spawned.

1. PHILOSOPHICAL CONSIDERATIONS

There is an analogy between the motivations behind semantic models and those behind high-level programming languages. The ALGOL-like languages were developed in an attempt to provide richer, more convenient programming abstractions; they buffer the user from low-level machine considerations. Similarly, semantic models attempt to provide more powerful abstractions for the specification of database schemas than are supported by the relational, hierarchical, and network models. Of course, more complex abstraction mechanisms introduce implementation issues. The construction of efficient semantic databases is an interesting problem—and largely an open research area.

In this section we focus on the major motivations and advantages of semantic database modeling as described in the literature. These were originally proposed in, for example, Hammer and McLeod [1981], Kent [1978], Kent [1979], and Smith and Smith [1977] and have since been echoed and extended in works such as Abiteboul and Hull [1987], Brodie [1984], King and McLeod [1985b], and Tsichritzis and Lochovsky [1982].

Historically, semantic database models were first developed to facilitate the design of database schemas [Chen 1976; Hammer and McLeod 1981; Smith and Smith 1977]. In the 1970s, the traditional models (relational, hierarchical, and network) were gaining wide acceptance as efficient data management tools. The data structures used in these models are relatively close to those used for the physical representation of data in computers, ultimately viewing data as collections of records with printable or pointer field values. Indeed, these models are often referred to as being *record based.* Semantic models were developed to provide a higher level of abstraction for modeling data, allowing database designers to think of data in ways that correlate more directly to how data arise in the world. Unlike the traditional models, the constructs of most semantic models naturally support a top-down, modular view of the schema, thus simplifying both schema design and database usage. Indeed, although the semantic models were first introduced as design tools, there is increasing interest and research directed toward developing them into full-fledged database management systems.

To present the philosophy and advantages of semantic database models in more detail, we begin by introducing a simple example using a generic semantic data model, along with a corresponding third normal form (3NF) relational schema. The example is used for several purposes. First, we present the fundamental differences between semantic models and the object-oriented paradigm from programming languages. Next, we illustrate the primary advantages often cited in the literature of semantic data models over the record-oriented models. We then show how these advantages relate to the process of schema design. We conclude by comparing semantic models with the related field of knowledge representation in AI.

1.1 An Example

The sample schema shown in Figure 1 is used to provide an informal introduction to many of the fundamental components of semantic data models. This schema is based on a generic model, called the Generic Semantic Model (GSM), which was developed for this survey and is presented in detail in Section 2.

The primary components of semantic models are the explicit representation of objects, attributes of and relationships among objects, type constructors for building complex types, ISA relationships, and

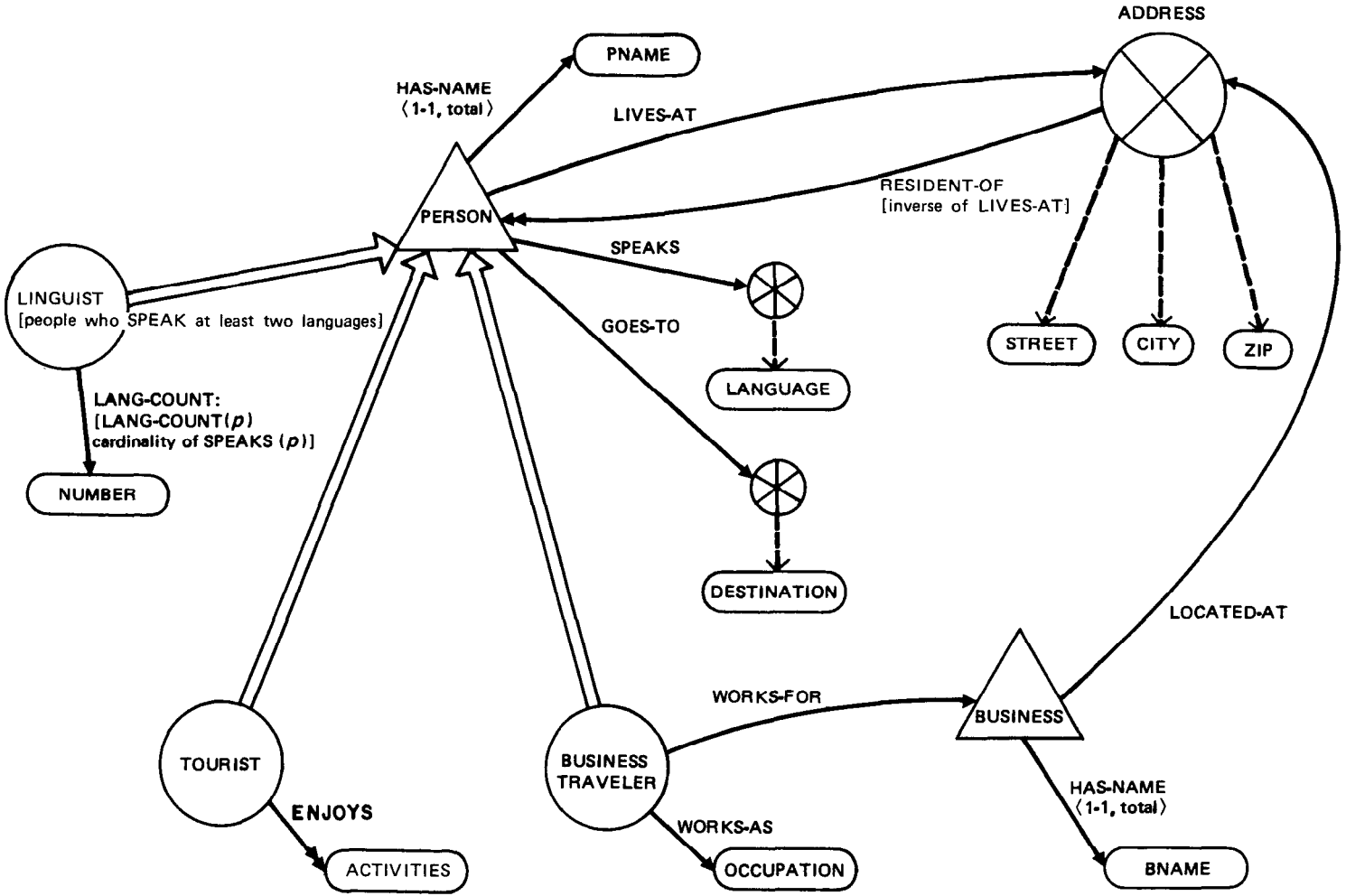


Figure 1. Schema of World Traveler database.

derived schema components. The example schema provides a brief introduction to each of these. The schema corresponds to a mythical database, called the World Traveler Database, which contains information about both business and pleasure travelers. It is necessarily simplistic but highlights the primary features common to the prominent semantic database models.

The World Traveler schema represents two fundamental object or entity types, corresponding to the types PERSON and BUSINESS. These are depicted using triangle nodes, indicating that they correspond to *abstract* data types in the world. Speaking conceptually, in an *instance* of this schema, a set of objects of type PERSON is associated with the PERSON node. In typical implementations of semantic data models [Atkinson and Kulkarni 1983; King 1984; Smith et al. 1981] (see Section 4.1), these abstract objects are referenced using internal identifiers that are not visible to the user. A primary reason for this is that objects in a semantic data model may not be uniquely identifiable using printable attributes that are directly associated with them. In contrast with abstract types, *printable* types such as PNAME (personname) are depicted using ovals. (In the work by Verheijen and Bekkum [1982], which considers the design of information systems, printable types are called lexical object types (LOT) and abstract types are called nonlexical object types (NOLOT).

The schema also represents three subtypes of the type PERSON, namely, TOURIST, BUSINESS-TRAVELER, and LINGUIST. Such subtype/supertype relationships are also called *ISA* relationships; for example, each tourist "is-a" person. In the schema, the three subtypes are depicted using circular nodes (indicating that their underlying type is given elsewhere in the schema), along with double-shafted ISA arrows indicating the ISA relationships. In an instance of this schema, subsets of the set of persons (i.e., the set of internal identifiers associated with PERSON node) would be associated with each of the three subtype nodes. Note that in the absence of any restrictions, the sets corresponding to these subtypes may overlap.

The sample schema illustrates two fundamental uses of subtyping in semantic models, these being to form *user-specified* and *derived* subtypes. For example, the subtypes TOURIST and BUSINESS-TRAVELER are viewed here as being user specified because a person will take on either (or both) of these roles only if this is specified by a database operation. In contrast, we assume here (again simplistically) that a person is a LINGUIST if that person can speak at least two languages. (The attribute SPEAKS that is defined on PERSON is discussed shortly.) Thus, the contents of the subtype LINGUIST can be *derived* from data stored elsewhere in the schema, along with the defining predicate (in pseudo-English) "LINGUIST := PERSONs who SPEAK at least two LANGUAGEs". This example illustrates one type of derived schema component typical of semantic models.

The sample schema also illustrates how *constructed types* can be built from atomic types in a semantic data model. One example of a constructed type is ADDRESS, which is an *aggregation* (i.e., Cartesian product) of three printable types STREET, CITY, and ZIP. This is depicted in the schema with an ×-node that has three children corresponding to the three coordinates of the aggregation. Aggregation is one form of *abstraction* offered by most semantic data models. For example, here it allows users to focus on the abstract notion of ADDRESS while ignoring its component parts. As we shall see, this aggregate object will be referenced by two different parts of the schema. A second prominent type constructor in many semantic models is called *grouping*, or *association* (i.e., finitary powerset) and is used to build sets of elements of an existing type. In the schema, grouping is depicted by a *-node and is used to form, for example, sets of LANGUAGEs and DESTINATIONs.

As illustrated above, object types can be modeled in a semantic schema as being abstract, printable, or constructed and can be defined using an ISA relationship. Through this flexibility the schema designer may choose a construct appropriate to the significance of the object type in the

particular application environment. For example, in a situation in which cities play a more prominent role (e.g., if CITY had associated attributes such as language or climate information), the type of city could be modeled as an abstract type instead of as a printable. As discussed below, different combinations of other semantic modeling constructs provide further flexibility.

So far, we have focused on how object types and subtypes can be represented in semantic data models. Another fundamental component of most semantic models consists of mechanisms for representing *attributes* (i.e., functions) associated with these types and subtypes. It should be noted that unlike the functions typically found in programming languages, many attributes arising in semantic database schemas are not computed but instead are specified explicitly by the user to correspond to facts in the world. In the World Traveler Database, attributes are represented using (single-shafted) arrows originating at the domain of the attribute and terminating at its range. For example, the type PERSON has four attributes: HAS-NAME, which maps to the printable type PNAME; LIVES-AT, which maps to objects of type ADDRESS; SPEAKS, which maps each person to the set of languages that person speaks; and GOES-TO, which maps each person to the set of destinations that person frequents. In the schema the HAS-NAME attribute is constrained to be a 1:1, total function. The attribute SPEAKS is set valued in the sense that the attribute associates a *set* of languages (indicated by the *-node) to each person. RESIDENT-OF is similar in that it associates a set of people with an address; however, this property is represented with a *multivalued* attribute. ENJOYS of TOURIST is also multivalued. The distinction between set valued and multivalued attributes is discussed in Section 2. In several models it is typical to depict both an attribute and its inverse. For example, in the sample schema, the inverse of the LIVES-AT attribute from PERSON to ADDRESS is a set-valued attribute RESIDENT-OF.

As shown in the schema, the subtype BUSINESS-TRAVELER has two attributes: WORKS-FOR and WORKS-AS. Because business travelers are people, the members of this subtype also *inherit* the four attributes of the type PERSON. Similarly, the other two subtypes of PERSON inherit these attributes of type PERSON.

The schema also illustrates how attributes can serve as derived schema components. One example is the attribute RESIDENT-OF; another is the attribute LANG-COUNT of the (derived) subtype LINGUIST, which is specified completely by the predicate "LANG-COUNT is cardinality of SPEAKS" and other parts of the schema.

To conclude this section, Figure 2 shows a 3NF [Ullman 1982] relational schema corresponding to the World Traveler schema. In order to capture most of the semantics of the original schema, key and inclusion dependencies are included in the relational schema. (Briefly, a *key dependency* states that the value of one (or several) field(s) of a tuple determines the remaining field values of that tuple; an *inclusion dependency* states that all of the values occurring in one (or more) column(s) of one relation also occur in some column(s) of another relation.) For example, PNAME is the key of PERSON, indicating that each person has only one address; and the PNAME column of TOURIST is contained in the PNAME column of PERSON, indicating that each tourist is a person. In this schema one or more relations is used for each of the object types in the semantic schema. For example, even ignoring the subtypes of the type PERSON, information about persons is stored in the three relations PERSON, PERSPEAKS, and PERGOES. (In principle, a single relation could be used for this information, but in the presence of set-valued attributes such as SPEAKS and GOES-TO, such relations will not be in 3NF.)

1.2 Semantic Models versus Object-Oriented Programming Languages

Now that we have briefly introduced the essentials of semantic modeling, we are in a position to describe the fundamental distinctions between semantic models and

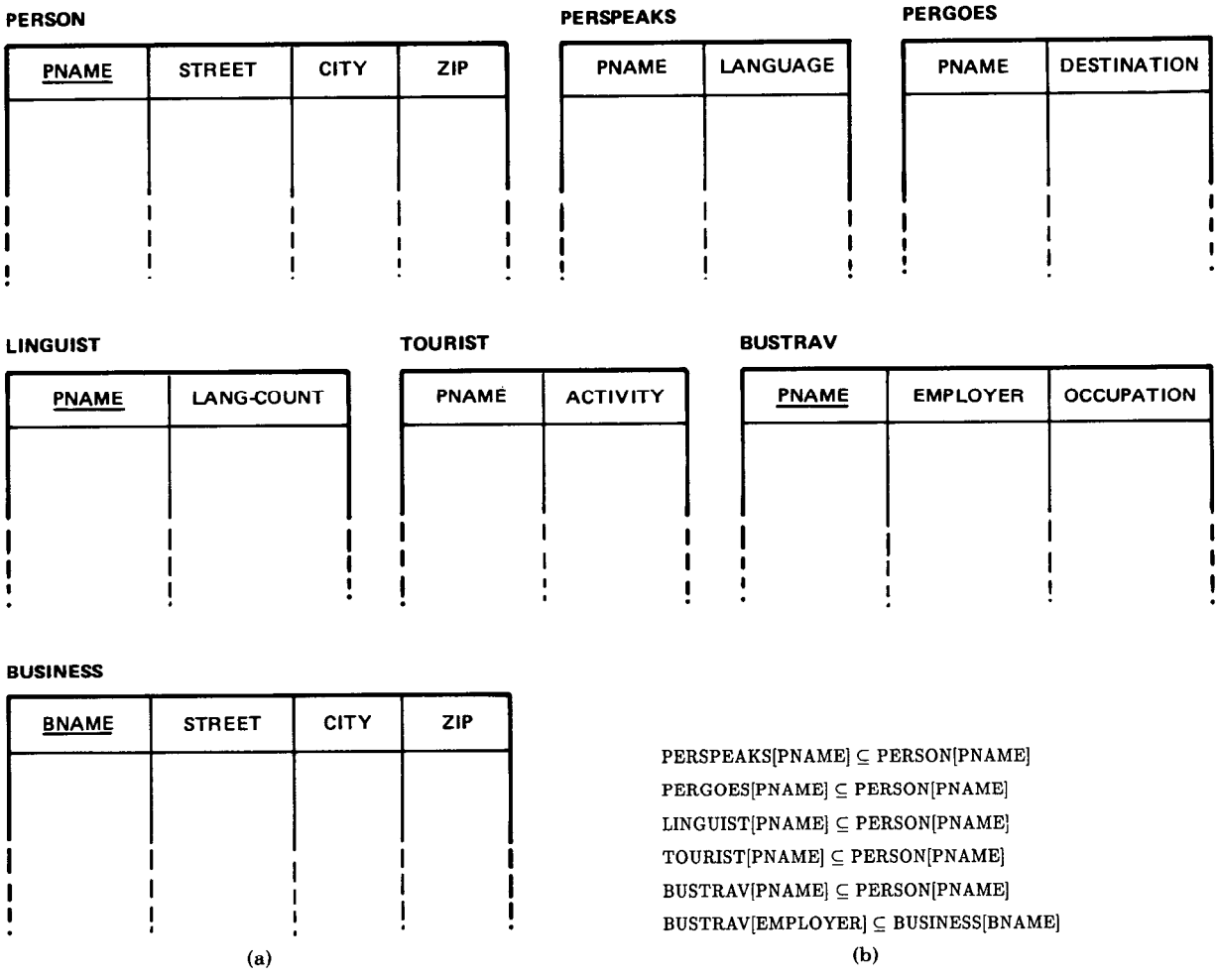


Figure 2. 3NF relational schema corresponding to the World Traveler schema. (a) Relations. (b) Inclusion dependencies.

object-oriented programming [Bobrow et al. 1986; Goldberg and Robson 1983; Moon 1986]. This is crucial in light of current database research thrusts.

Essentially, semantic models encapsulate structural aspects of objects, whereas object-oriented languages encapsulate behavioral aspects of objects. Historically, object-oriented languages stem from research on abstract data types [Guttag 1977; Liskov et al. 1977]. There are three principle features of object-oriented languages. The first is the explicit representation of object classes (or types). Objects are identified by surrogates rather than by their values. The second feature is the encapsulation of "methods" or operations within objects. For example, the object type GEOMETRIC_OBJECT may have the method "display_self". Users are free to ignore the implementation details of methods. The final feature of object-oriented languages is the inheritance of methods from one class to another.

There are two central distinctions between this approach and that of semantic models. First, object-oriented models do not typically embody the rich type constructors of semantic models. From the structural point of view, object-oriented models support only the ability to define single- and multivalued attributes. Second, the inheritance of methods is strictly different from the inheritance of attributes (as in semantic models). In a semantic model, the inheritance of attributes is only between types where one is a subset of the other. The inheritance of a method, since it is a behavioral—and not a structural—property, can be between seemingly unlike types. Thus, the object type TEXT might be able to inherit the "display_self" method of GEOMETRIC_OBJECT.

1.3 Advantages of Semantic Data Models

In this section we summarize the motivations often cited in the literature in support of semantic data models over the traditional data models. We noted above that semantic data models were first introduced primarily as schema design tools and embody the fundamental kinds of relationships arising in typical database applications. As a result of this philosphical foundation, semantically based data models and systems provide the following advantages over traditional, record-oriented systems:

(1) increased separation of conceptual and physical components,
(2) decreased semantic overloading of relationship types,
(3) availability of convenient abstraction mechanisms.

Abstraction mechanisms are the means by which the first two advantages of semantic models are obtained. We discuss abstraction separately because of the significant effort researchers have put into developing these mechanisms. Each of the three advantages is discussed below.

1.3.1 Increased Separation of Logical and Physical Components

In record-oriented models the access paths available to end users tend to mimic the logical structure of the database schema directly [Chen 1976; Hammer and McLeod 1981; Kent 1979; Kerschberg and Pacheco 1979; Shipman 1981; Smith and Smith 1977]. This phenomenom exhibits itself in different ways in the relational and the hierarchical/network models. In the relational model a user must simulate pointers by comparing identifiers in order to traverse from one relation to another (typically using the join operator). In contrast, the attributes of semantic models may be used as direct conceptual pointers. Thus, users must consciously traverse through an extra level of indirection imposed by the relational model, making it more difficult to form complex objects out of simpler ones. For this reason, the relational model has been referred to as being *value oriented* [Khoshafian and Copeland 1986; Ullman 1987] as opposed to *object oriented.*

In the hierarchical and network models a similar situation occurs. Users must navigate through the database, constructing larger objects out of flat record structures by associating records of different types. In contrast, semantic models allow users to

focus their attention directly on abstract objects. Thus, in a hierarchical/network model, the access paths correspond directly to the low-level physical links between records and not to the conceptual relationships modeled in a semantic schema.

To illustrate this point using the relational model, suppose that in the World Traveler database Mary is a business traveler. Using attributes, the city of Mary's employer can be obtained with the simple query:

```
print LOCATED-AT (WORKS-
  FOR('Mary')).CITY
```

This query operates as follows: Mary's employer is obtained by WORKS-FOR('Mary'); applying LOCATED-AT yields the address of that employer, and the '.CITY' construct isolates the second coordinate of the address. (We assume as syntactic sugar that because HAS-NAME is 1:1, the string 'Mary' can be used to denote the person Mary; if not, in the above query, 'Mary' would have to be replaced by HAS-NAME^{-1}('Mary').) Thus, the semantic model permits users to refer to an object (in this case using a printable surrogate identifier) and to "navigate" through the schema by applying attributes directly to that object. In the relational model, on the other hand, users must navigate through the schema within the provided record structure using joins. In the SEQUEL language, for example, the analogous query directed at the schema of Figure 2 would be

```
select CITY
from BUSINESS
where BNAME in
  select EMPLOYER
  from BUSTRAV
  where PNAME = 'Mary'
```

In essence, the user first obtains the name of Mary's employer by selecting the record about Mary in the relation BUSTRAV and retrieving the EMPLOYER attribute, then finds the record in the relation BUSINESS that has that value in its BNAME field, and finally reads the CITY attribute of that record. Thus, the linkage between the BUSTRAV and BUSINESS relations is obtained by explicitly comparing business identifiers (the EMPLOYER coordinate of BUSTRAV and the BNAME coordinate of BUSINESS).

1.3.2 Semantic Overloading

The second fundamental advantage cited for the semantic models focuses on the fact that the record-oriented models provide only two or three constructs for representing data interrelationships, whereas semantic models typically provide several such constructs. As a result, constructs in record-oriented models are *semantically overloaded* in the sense that several different types of relationships must be represented using the same constructs [Hammer and McLeod 1981; Kent 1978, 1979; Smith and Smith 1977; Su 1983]. In the relational model, for example, there are only two ways of representing relationships between objects: (1) within a relation and (2) by using the same values in two or more relations.

To illustrate this point, we briefly compare the relational and semantic schemas of the World Traveler database. In the relational schema, at least three different types of relationships are represented structurally within individual relations:

(1) the functional relationship between PNAME and STREET;
(2) the many–many association between PNAMEs and LANGUAGEs;
(3) the clustering of STREET, CITY, and ZIP values as addresses.

At least three other types of relationships are represented by pairs of relations:

(a) the type/subtype relationship between PERSON and TOURIST;
(b) the fact that PERSON, PERSPEAKS, and PERGOES all describe the same set of objects;
(c) the fact that the employers of BUSTRAVs are described in the BUSINESS relation.

In contrast, each of these types of relationship has a different representation in the semantic schema.

As indicated above, in the absence of integrity constraints the data structuring

primitives of the relational model (and the other record-oriented models) are not sufficient to model the different types of commonly arising data relationships accurately. This is one reason that integrity constraints such as key and inclusion dependencies are commonly used in conjunction with the relational model. Although these do provide a more accurate representation of the data, they are typically expressed in a text-based language; it is therefore difficult to comprehend their combined significance. A primary objective of many semantic models has been to provide a coherent family of constructs for representing in a structural manner the kinds of information that the relational model can represent only through constraints. Indeed, semantic modeling can be viewed as having shifted a substantial amount of schema information from the constraint side to the structure side.

1.3.3 Abstraction Mechanisms

Semantic models provide a variety of convenient mechanisms for viewing and accessing the schema at different levels of abstraction [Hammer and McLeod 1981; King and McLeod 1985a; Smith and Smith 1977; Su 1983; Tsichritzis and Lochovsky 1982]. One dimension of abstraction provided by these models concerns the level of detail at which portions of a schema can be viewed. On the most abstract level, only object types and ISA relationships are considered. At this level the structure of objects is ignored; for example, the ×-node ADDRESS would be shown without its children. A more detailed view includes the structure of complex objects; the further detail includes attributes and the rules governing derived schema components.

A second dimension of the abstraction provided by semantic models is the degree of modularity they provide. It is easy to isolate information about a given type, its subtypes, and its attributes. Furthermore, it is easy to follow semantic connections (e.g., attribute and ISA relationships) to find closely associated object types. Both of the above dimensions of abstraction are very useful in schema design and for schema *browsing*, that is, the ad hoc perusal of a schema to determine what and how things are modeled. Interactive graphics-based systems that use these properties of semantic models have been developed (see Section 4.3); comparable systems for the record-oriented models have not been developed.

An interesting question is why the central components of semantic models—objects, attributes, ISA relationships—are necessarily the best mechanisms to use to enrich a data model. Although, of course, there can be no clearcut choice of modeling constructs, there are two reasons to support the selection of these particular primitives. First, practice has shown that schemas constructed with traditional record-oriented models tend to simulate objects and attributes by interrelating records of different types with logical and physical pointers. The second point is that computer science researchers in AI and programming languages have selected similar constructs to enhance the usability of other software tools. It is thus interesting that researchers with somewhat different goals have found semantic model-like mechanisms useful. This latter point is discussed in more detail later in this section.

A third dimension of abstraction is provided by derived schema components that are supported by a few semantic models [Hammer and McLeod 1981; King and McLeod 1985a; Shipman 1981] and also by some relational implementations [Stonebraker et al. 1976]. These schema components allow users to define new portions of a schema in terms of existing portions of a schema. Derived schema components permit the user to identify a specific subset of the data, possibly perform computations on it, and then structure it in a new format. The "new" data are then given a name and can subsequently be used while ignoring the details of the computation and reformatting. In the relational model, derived schema components must be either new relations or new columns in existing relations. Semantic models provide a much richer framework for defining derived schema components. For example, a derived subtype specifies both a new type and

an ISA relationship; similarly, a derived single-valued attribute specifies both a piece of data and a constraint on it. Therefore, semantic models give the user considerably more power for abstracting data in this way.

Derived data are closely related to the notion of a *user view* (or external schema) [Chamberlain et al. 1975; Tsichritzis and Klug 1977], except that derived data are incorporated directly into the original schema rather than used to form a separate new schema. Another difference is that a view may contain raw or underived components, as well as derived information.

1.4 Database Design with a Semantic Model

In general, the advantages of semantic models, as described in the literature, are oriented toward the support of database design and evolution [Brodie and Ridjanovic 1984; Chen 1976; King and McLeod 1985a; Smith and Smith 1977]. At the present time the practical use of semantic models has been generally limited to the design of record-oriented schemas. Designers often find it easier to express the high-level structure of an application in a semantic model and then map the semantic schema into a lower level model. One prominent semantic model, the Entity–Relationship Model, has been used to design relational and network schemas for over a decade [Teorey et al. 1986]. Interestingly, relational schemas designed using the ER Model are typically in 3NF, an indication of the naturalness of using a semantic model as a design tool for traditional DBMSs.

A number of features of semantic models contribute to their use in both the design and the eventual evolution of database schemas. They provide constructs that closely parallel the kinds of relationships typically arising in database application areas; this makes the design process easier and lessens the likelihood of design errors. This is in contrast to record-oriented models, which force the designer to concentrate on many low-level details. Semantic models also provide a variety of abstraction mechanisms that researchers have used to develop structured design methodologies. A detailed and fairly comprehensive design methodology appears in Rosussopoulos and Yeh [1984]. After requirements analysis is performed, the authors advise the use of a semantic model as a means of integrating and formalizing the requirements. A semantic model serves nicely as a buffer between the form of requirements collected from noncomputer specialists and the low-level computer-oriented form of record-oriented models. Several methodologies have also addressed the issue of integrating schema and transaction design in order to simplify the collection and formalization of database dynamic requirements; see Brodie and Ridjanovic [1984] and King and McLeod [1985a] for examples.

Semantic models are a convenient mechanism for allowing database specifications to evolve incrementally in a natural, controlled fashion [Brodie and Ridjanovic 1984; Chen 1976; King and McLeod 1985a; Teorey 1986]. This is because semantic models provide a framework for top-down schema design, beginning with the specification of the major object types arising in the application environment, then specifying subsidiary object types. Referring to the World Traveler schema, the design might begin with the specification of the PERSON and BUSINESS nodes; the LINGUIST, TOURIST, and BUSINESS-TRAVELER nodes would follow; and finally the various attributes would be defined. The constructed type ADDRESS might be introduced when it is realized that both PERSON and BUSINESS share the identical attributes STREET, CITY, and ZIP.

In conclusion, significant research has been directed at applying specific semantic models to the design of either semantic or traditional database schemas. However, little work has been directed at providing methodological support for selecting an appropriate semantic model or for integrating the various modeling capabilities found in semantic models. Rather, methodological approaches are typically tied to one model and to one prescriptive approach to producing a semantic schema.

1.5 Related Work in Artificial Intelligence

We now consider the relationship between semantic data modeling and research on knowledge representation in artificial intelligence. Although they have different goals, these two areas have developed similar conceptual tools.

Early research on knowledge representation focused on *semantic networks* [Findler 1979; Israel and Brachman 1984; Mylopoulos 1980] and *frames* [Brachman and Schmolze 1985; Fikes and Kehler 1985; Minsky 1984]. In a semantic network, real-world knowledge is represented as a graph formed of data items connected by edges. The graph edges can be used to construct complex items recursively and to place items in categories according to similar properties. The important relationship types of *ISA, is-instance-of,* and *is-part-of* (which is closely related to aggregation) are naturally modeled in this context. Unlike semantic data models, semantic networks mix schema and data in the sense that they do not typically provide convenient ways of abstracting the *structure* of data from the data itself. As a consequence, each object modeled in a semantic network is represented using a node of the semantic network; these networks can be quite large if many objects are modeled. One of the earliest semantic database models, the Semantic Binary Data Model [Abrial 1974], is closely related to semantic networks; schemas from this model are essentially semantic networks that focus exclusively on object classes.

Frame-based approaches provide a much more structured representation for object classes and relationships between them. Indeed, there are several rough parallels between the frame-based approach and semantic data models. The frame-based analog of the abstract object types is called a *frame.* A frame generally consists of a list of *properties* of objects in the type (e.g., elephants have four legs) and a tuple of *slots,* which are essentially equivalent to the attributes of semantic data models. Frames are typically organized using ISA relationships, and slots are inherited along ISA paths in a manner similar to the semantic data models. In general, properties of a type are inherited by a subtype, but exceptions to this inheritance can also be expressed within the framework (e.g., three-legged elephants are elephants, but have only three legs). Exception-handling mechanisms may also be provided for the inheritance of slot values. For example, referring to the World Traveler Database, in a frame-based approach the HAS-NAME attribute of a given person might be different in the role of PERSON and the role of TOURIST (e.g., a nick-name). (Although the terminology used by the KL-ONE model [Brachman and Schmolze 1985] differs from that just given, essentially the same concepts are incorporated there.)

In general, frame-based approaches do not permit explicit mechanisms, such as aggregation and grouping for object construction. In recent research and commercial systems [Aikens 1985; Kehler and Clemenson 1983; Stefik et al. 1983], frames have been extended so that slots can hold methods in the sense of object-oriented programming languages; this development parallels current research in object-oriented databases, which is briefly discussed in Section 5.

Because frame-based systems are generally in-memory tools, the sorts of research efforts that have been directed at implementing semantic databases have not been applied to them. For example, considerable research effort has focused on the efficient implementation of semantic schemas and derived schema components [Chan et al. 1982; Farmer et al. 1985; Hudson and King 1986, 1987; Smith et al. 1981].

2. TUTORIAL

This section provides an in-depth discussion of the fundamental features and components common to most semantic database models. The various building blocks used in semantic models are described and illustrated, and subtle and not-so-subtle differences between similar components are highlighted. Philosophical implications of the overall approaches to modeling taken by different models are also considered.

To provide a basis for our discussion, we use the Generic Semantic Model (GSM). The model was developed expressly for this survey and is based largely on three of the most prominent models found in the literature: the Entity–Relationship (ER) Model, the Functional Data Model (FDM), and the Semantic Data Model (SDM). The GSM is derived in large part from the IFO Model [Abiteboul and Hull 1987], which itself was developed as a theoretical framework for studying the prominent semantic models [Abrial 1974; Brodie and Ridjanovic 1984; Hammer and McLeod 1981; Kerschberg and Pacheco 1976; King and McLeod 1985a; Shipman 1981; Sibley and Kerschberg 1977]. Although the GSM incorporates many of the constructs and features of these models, it cannot be a true integration of all semantic models because of the very different approaches they take. Specifically, the approach taken by GSM is closest to the FDM. Because the primary purpose of GSM has been to serve as a tool for exposition, it is not completely specified in this paper.

In some cases the literature taken as a whole uses a given term ambiguously. Perhaps the most common example of this is the term "aggregation." At a philosophical level, this term is used universally to indicate object types that are formed by combining a group of other objects; for example, ADDRESS might be modeled as an aggregation of STREET, CITY, and ZIP. At a more technical level, some models support this using a construction based on Cartesian product, whereas others use a construction based on attributes. In this section we adopt specific, somewhat technical definitions for various terms. For example, we use aggregation to refer to Cartesian-product-based constructions. These more restrictive definitions will permit a clear articulation of the different concepts arising in the literature.

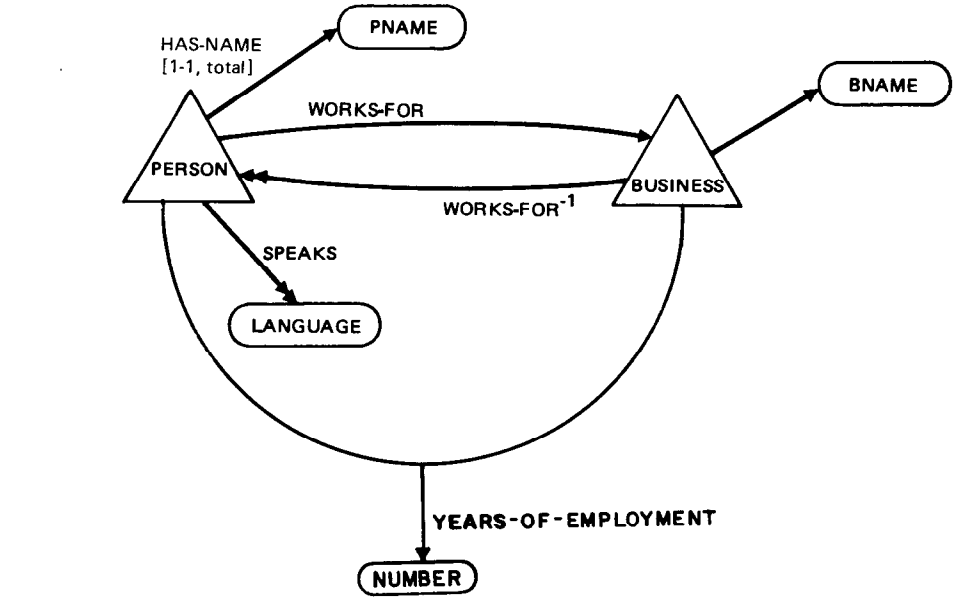
This section has four major parts. The first briefly compares two broad philosophical approaches that many models choose between, providing a useful perspective before delving into a detailed discussion of the different building blocks of semantic models. The second part defines the specific constructs used for describing the *structure* of data in semantic models and presents examples that highlight similarities and differences between them. The third considers how these constructs are combined and augmented to form database schemas in semantic models. The fourth discusses languages for accessing and manipulating data, and for specifying semantic schemas.

2.1 Two Philosophical Approaches

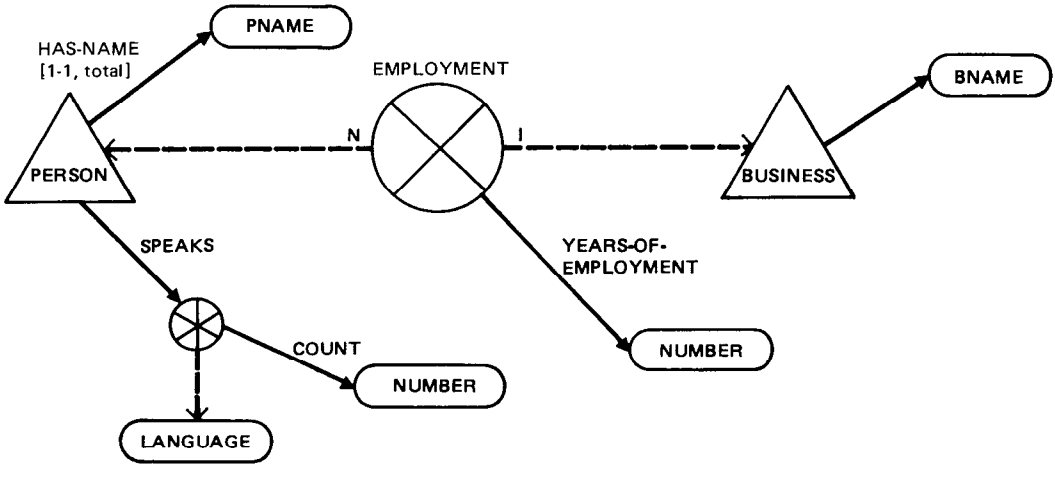
The GSM is meant to be representative of a wide class of semantic models; as a result of being somewhat eclectic, it blurs an important philosophical distinction arising in semantic modeling literature. Historically, there have been two general approaches taken in constructing semantic models. The distinction between them is not black and white, but models have had a tendency to adopt one approach or the other. Essentially, various models place different emphasis on the various constructs for interrelating object classes. One approach stresses the use of attributes to interrelate objects; the other places an emphasis on explicit type constructors. As a result, different data models may yield dramatically different schemas for the same underlying application.

To illustrate this point, for the same underlying data we compare two schemas that give very different prominence to attributes and type constructors. The comparison is particularly salient because the schemas reflect the underlying philosophies of two early influential semantic models, namely, the FDM and the ER Models, respectively.

Figure 3 shows the two GSM schemas, both representing the same data underlying a portion of the World Traveler Database application. The schema in Figure 3a loosely follows the FDM and emphasizes the use of attributes for relating abstract object types with other abstract object types. The schema in Figure 3b loosely follows the philosophy of the ER Model in that it emphasizes the use of type constructor aggregation (called *relationship* in the ER Model) and grouping for relating



(a)



(b) Emphasis on constructed types

(b)

Figure 3. Two schemas for the same underlying data. (a) Schema emphasizing attributes. (b) Schema emphasizing type constructors.

abstract object types. In both schemas an instance includes a set of PERSONs and a set of BUSINESSes (both considered sets of abstract objects), along with attributes specifying person and business names and the languages spoken by PERSONs.

Interestingly, in an instance of the first schema the relationship of people and their business is represented by the attribute (i.e., function) WORKS-FOR and its inverse WORKS-FOR^{-1}; in the second, the aggregation EMPLOYMENT (which is a

set of ordered pairs) is used. Both schemas represent the constraint that many people work for the same business, but not the reverse: In the first schema this is accomplished using a single-valued and a multivalued attribute, and in the second by the $N:1$ constraint. Further, in the first schema, a multivalued attribute is used to represent the languages spoken by a person, whereas in the second, a grouping construct is used.

The choice of emphasis—attribute based or type constructor based—affects the language mechanisms that seem natural for manipulating semantic databases. Consider Figure 3a. If a user wanted to know the business of a particular person, the attribute WORKS-FOR may be used to reference the business directly. In Figure 3b, the type constructor representing ordered pairs of PERSONs and BUSINESSes must be manipulated in order to obtain the desired data. On the other hand, the type constructor approach gives the user the flexibility of directly referencing, by name, ordered pairs in EMPLOYMENT.

The use of type constructors also allows information to be associated directly with schema abstractions. As one illustration, the bottom subschema includes an attribute on EMPLOYMENT that describes the length of time an individual has been employed at a particular company. (Essentially the same information is represented in the first schema with the two-argument attribute YEARS-OF-EMPLOYMENT, although the relationship EMPLOYMENT and this attribute are not linked together.) Analogously, in the second schema, the grouping construct for LANGUAGES is augmented by an attribute giving the cardinality of each set of languages. (No analog for this exists in the attribute-based approach.) In a model that stresses type constructors, relationships between types are essentially viewed as types in their own right; thus it makes perfect sense to allow these types to have attributes that further describe them.

2.2 Local Constructs

This section presents detailed descriptions of the building blocks that semantic models use to represent the structure of data. The discussion is broken into three parts, which focus on types, attributes, and ISA relationships, respectively. Importantly, in the section on attributes we compare the notions of attributes and aggregations.

2.2.1 Atomic and Constructed Types

A fundamental aspect of all semantic models is the direct representation of object types, distinct from their attributes and sub- or supertypes. Most models provide mechanisms to represent atomic or nonconstructed object types, and many models also provide type constructors. In the discussion below we focus on the use of object types in semantic models and on the two most prominent type constructors, namely, aggregation and grouping.

A semantic model typically provides the ability to specify a number of *atomic types*. Intuitively, each of these types corresponds to a class of nonaggregate objects in the world, such as PERSONs or ZIP-codes. (Of course, the type PERSON has many attributes.) Many semantic models distinguish between atomic types that are *abstract* and those that are *printable* (or *representable*). The abstract types are typically used for physical objects in the world, such as PERSONs, and for conceptual (or legal) objects, such as BUSINESSes. Atomic printable types are typically alphanumeric strings, but in some graphics-based systems they might include icons as well. It is often convenient to articulate subclasses of these, such as ZIP-codes, Person-NAMEs, or Business-NAMEs, and most models associate operators, such as addition for numbers, with them. As shown in the World Traveler schema, in the GSM abstract types are depicted with triangles, atomic printable types are depicted with flattened ovals, and subtypes are depicted with circles.

In instances of a semantic schema, abstract objects are viewed conceptually to correspond directly to physical or conceptual objects in the world and in some implementations of semantic models, they are represented using internal identifiers that are not directly accessible to the user. This corresponds to the intuition that such

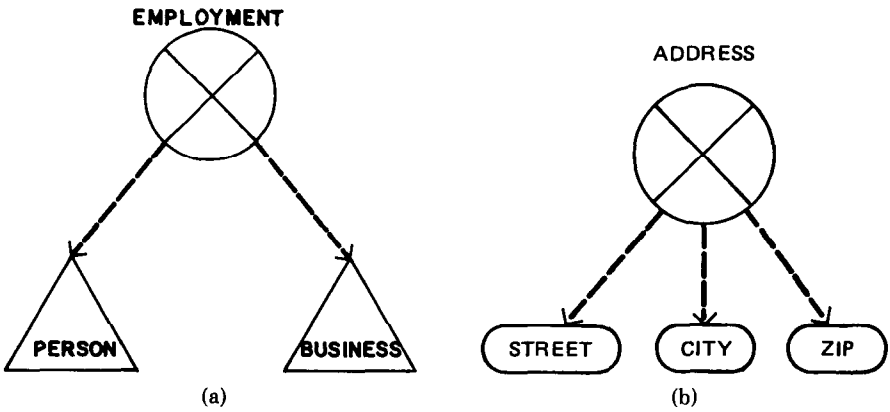


Figure 4. Object types constructed with aggregation. (a) EMPLOYMENT = PERSON × BUSINESS. (b) ADDRESS = STREET × CITY × ZIP.

objects cannot be "printed" or "displayed" on paper or on a monitor.

When defining an instance of a semantic schema, an *active domain* is associated with each node of the schema. The active domain of an atomic type holds all objects of that type that are currently in the database. This notion of active domain is extended to type constructor nodes below.

We now turn to *type constructors*. The most prominent of these in the semantic literature are *aggregation* (called *relationship* in the ER Model) and *grouping* (also known as *association* [Brodie and Ridjanovic 1984]). An aggregation is a composite object constructed from other objects in the database. For example, each object associated with the aggregation type EMPLOYMENT in Figure 4a is an ordered pair of PERSON and BUSINESS values. Mathematically, an aggregation is an ordered n-tuple. In an instance, the active domain of an aggregation type will be a *subset* of the Cartesian product of the active domains assigned to the underlying nodes. For example, the active domain of EMPLOYMENT will be the set of pairs corresponding to the set of employee–employer relationships currently true in the database application. According to our definition, the identity of an aggregation object is completely determined by its component values. Figure 4b highlights the use of aggregation for encapsulating information.

Before continuing, we reiterate that the definition of aggregation used here is deliberately narrow and differs from the usage of that term in some models, including SDM and TAXIS. The representation of aggregations in those models is generally based on attributes and is discussed in the next section. It should also be noted that some models, including FDM, emphasize the use of attributes, as well as support the use of aggregations in attribute domains.

The grouping construct is used to represent sets of objects of the same type. Figure 5a shows the GSM depiction of the grouping construct to form a type whose objects are sets of languages. Mathematically, a grouping is a finite set. In an instance, the active domain of a grouping type will hold a set of objects, each of which is a finite subset of the active domain of the underlying node. In a constructed object, a *-node will always have exactly one child.

As defined here, a grouping object is a set of objects. Technically, then, the *identity* of a grouping object is determined completely by that set. To emphasize the significance of this, we consider how committees might be modeled in a semantic schema. One approach is to define the type COMMITTEE as a grouping of PERSON because each committee is basically a set of people. This is probably not accurate in most cases because the identity of a

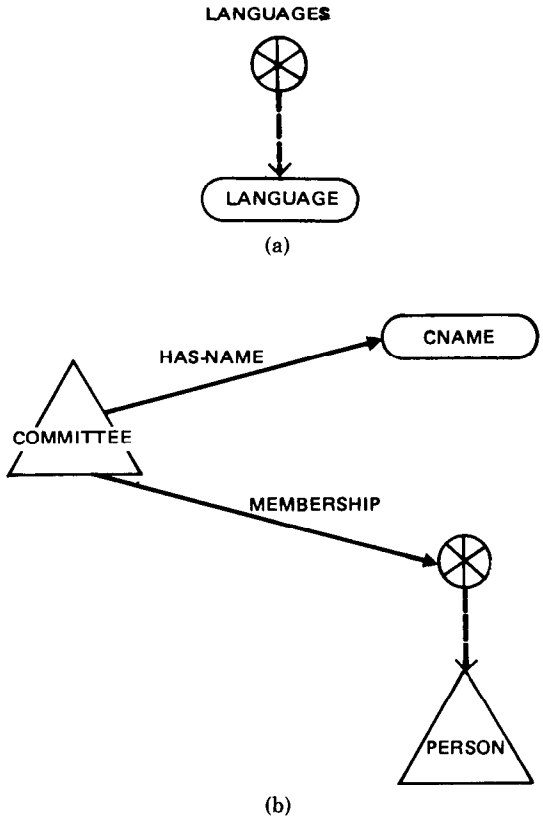


Figure 5. Object types constructed with grouping. (a) LANGUAGES = * LANGUAGE.

committee is separate from its membership at a particular time. Figure 5b shows a more appropriate approach. COMMITTEE is modeled as an abstract type and has an attribute MEMBERSHIP whose range is a grouping type.

As illustrated in Figure 6, the type constructors can be applied recursively. In this example, we view a VISIT as a triple consisting of a TOURIST-TRAP, a GUIDE (viewed as a subtype of PERSON), and a set of TOURISTs (also a subtype of person). As indicated in the figure, edges originating from an aggregation node can be labeled by a role; this is important if more than one child of an aggregation is of the same type. In the GSM and most semantic models supporting aggregation and grouping, there can be no (directed or undirected) cycle of type constructor edges. The Logical Data Model [Kuper and Vardi 1984, 1985] provides an alternative formalism in which cycles are permitted.

We close this section by mentioning other kinds of type constructors found in the literature. The TAXIS and Galileo models support *metatypes*; that is, types whose elements are themselves types. For example, in the World Traveler example, a metatype TYPE-OF-PERSON might contain the types PERSON, LINGUIST, TOURIST, and BUSINESS-TRAVELER. This metatype could have attributes such as SIZE or AVERAGE-AGE, which describe characteristics of the populations of the underlying types. A comparison of metatypes with both subtypes and the grouping construct is presented in Section 2.3.2.

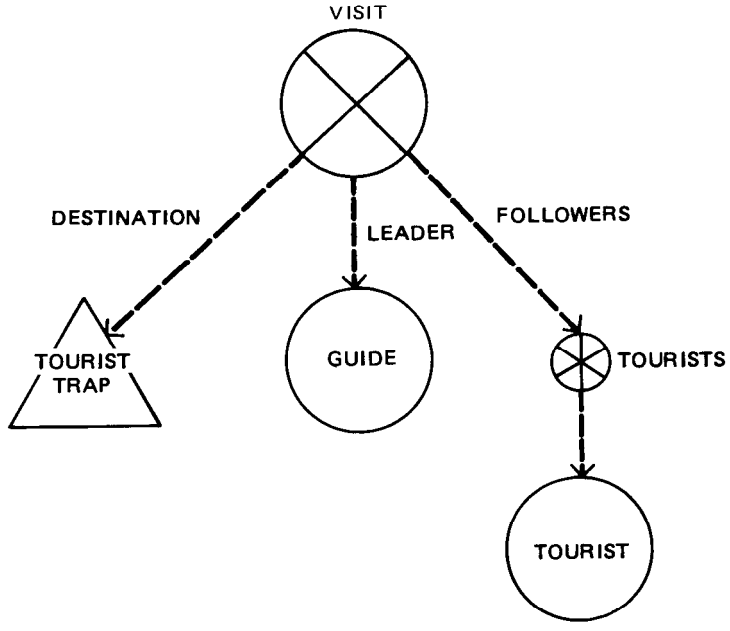
In principle, a data model can support essentially any type constructor in much the same way in which some programming languages do. Historically, almost all semantic models have focused almost exclusively on aggregation and grouping. Notable exceptions include SAM* (Semantic Association Model), TAXIS, and Galileo. These models permit a variety of type constructors that may be applied to atomic printable types. SAM* is oriented in part toward scientific and statistical applications and supports sets, vectors, ordered sets, and matrices; TAXIS and Galileo supports type constructors typical of imperative programming languages.

To summarize, semantic models typically differentiate between abstract and printable types and provide type constructors for aggregation and grouping.

2.2.2 *Attributes*

The second fundamental mechanism found in semantic models for relating objects is the notion of attribute (or function) between types. In this section we articulate a specific meaning for this notion and indicate the various forms it takes in different semantic models. We conclude with a comparison of different modeling strategies using aggregation and attributes.

We begin by defining the notion of attribute as used in the GSM. Speaking formally,



VISIT = DESTINATION:TOURIST-TRAP × LEADER:GUIDE × FOLLOWERS:(*TOURIST)

Figure 6. Recursive application of aggregation and grouping constructs.

a *one-argument attribute* in a GSM schema is a directed binary relationship between two types (depicted by an arrow), and an *n-argument attribute* is a directed relationship between a set of n types and one type (depicted by an arrow with n tails). Attributes can be *single valued*, depicted using an arrow with one pointer at its head, or *multivalued*, depicted using an arrow with two pointers at its head. In an instance, a mapping (a binary or $(n + 1)$-ary relation) is assigned to each attribute; the domain of this mapping is the (cross product of the) active domain(s) of the source(s) of the attribute, and the range is the active domain of the target of the attribute. The mapping may be specified explicitly through updates, or in the case of derived attributes it may be computed according to a derivation rule. In the case of a single-valued attribute, the mapping must be a function in the strict mathematical sense, that is, each object (or tuple) in the domain is assigned at most one object in the range. In GSM, there are no restrictions on the types of the source or target of an attribute.

Of course, there is a close correspondence between the semantics of a multivalued attribute and the semantics of a single-valued attribute whose range is a constructed grouping type. In keeping with the general philosophy that the GSM incorporates prominent features from several representative semantic models, both of these possibilities have been included. Most models in the literature support multivalued attributes and do not permit an attribute to map to a grouping type. Also, some models, including SDM and INSYDE, view all attributes as multivalued and use a constraint if one of them is to be single valued. Similarly, there is also a close relationship between a one-argument attribute whose domain is an aggregation and an n-argument attribute.

We now briefly mention another kind of attribute, called here a *type attribute*. This is supported in several models, including SDM, TAXIS, and SAM*. Type attributes associate a value with an entire type, instead of associating a value with each object in the active domain. For example,

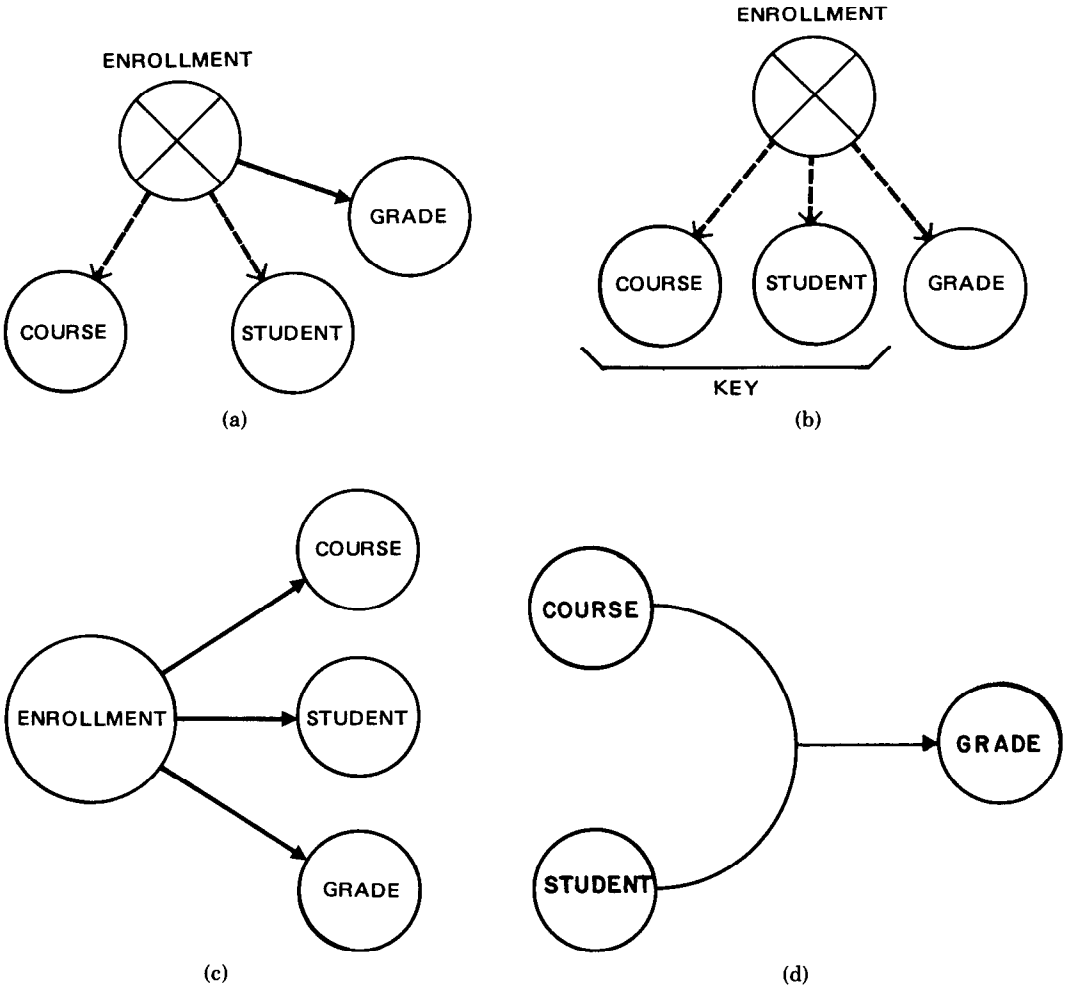


Figure 7. Four alternative representations for ENROLLMENT.

the type attribute COUNT might be associated with the type PERSON and would hold one value: the number of people currently "in" the database. Other type attributes might hold more complex statistics about a type, for example, the average salary or the standard deviation of those salaries. The value associated with a type attribute is generally prescribed in the schema; such attributes thus form a special kind of derived data.

We conclude the section by comparing four different ways of representing essentially the same data interrelationships using the aggregation and attribute constructs. Figure 7 shows four subschemas that might be used to model the type ENROLLMENT. To simplify the pictures, we depict all atomic nodes as circular. In the first subschema, ENROLLMENT is viewed as an aggregation of COURSE and STUDENT. Each object of type ENROLLMENT will be an ordered pair, and a GRADE is associated with it by the attribute shown. The IFO and Galileo models provide explicit mechanisms for this representation. The second approach might be taken in such models as SAM* and SHM$^+$, which do not provide an explicit attribute construct. In this case ENROLLMENT is

viewed as a ternary aggregation of COURSE, STUDENT, and GRADE. As suggested in the diagram, a key constraint is typically incorporated into this schema to ensure that each course–student pair has only one associated grade. The third approach shown in Figure 7c might be taken in models that do not provide an explicit type constructor for aggregation. Many semantic models fall into this category, including SBDM, SDM, TAXIS, and INSYDE (and the object-oriented programming language SMALLTALK, for that matter). Under this approach ENROLLMENT is viewed as an atomic type with three attributes defined on it. Although not shown in Figure 7c, a constraint might be included so that no course–student pair has more than one grade. The fourth approach is especially interesting in that it does not require that the construct ENROLLMENT be explicitly named or defined if it is not in itself relevant to the application. In this case the attribute for GRADE would be a function with two arguments. FDM has this capability.

We now compare the first three of these approaches from the perspective of object identity. In Figure 7a, each enrollment is an ordered pair. Thus, the grade associated with an enrollment can change without affecting the identity of the enrollment. Technically speaking, in the absence of the key dependency, this is not true in Figure 7b, in which an enrollment is an ordered triple. In Figure 7c, the underlying identity is independent of any of the associated course, student, and grade values. An enrollment e with values CS101, Mary, and 'A' might be modified to have values Math2, Mary, 'B' without losing its underlying identity. Also, in the absence of a constraint, the structure does not preclude the possibility that two distinct enrollments e and e' have the same course, the same student, and the same grade.

2.2.3 ISA Relationships

The third fundamental component of virtually all semantic models is the ability to represent *ISA* or supertype/subtype relationships. In this section we review the basic intuitions underlying these relationships and describe different variations of the concept found in the literature. The focus of this section is on the local properties of ISA relationships; global restrictions on how they may be combined are discussed in Section 2.3.1. In several models subtypes arise almost exclusively as derived subtypes; this aspect of subtypes is considered in Section 2.3.2.

Intuitively, an ISA relationship from a type SUB to a type SUPER indicates that each object associated with SUB is associated with the type SUPER. For example, in the World Traveler schema the ISA edge from TOURIST to PERSON indicates that each tourist is a person. More formally, in each instance of the schema, the active domain of TOURIST must be contained in the active domain of PERSON. In most semantic models each attribute defined on the type SUPER is automatically defined on SUB; that is, attributes of SUPER are *inherited* by SUB. It is also generally true that a subtype may have attributes not shared by the parent type.

The family of ISA relationships in a schema forms a directed graph. In the literature this has been widely termed the ISA "hierarchy." However, as suggested in Figure 8, most semantic models permit undirected (or weak) cycles in this graph. For this reason we follow Atzeni and Parker [1986] and Lenzerini [1987] in adopting the term *ISA network*. Although ISA relationships are transitive, it is customary to specify the fundamental ISA relationships explicitly and view the links due to transitivity as specified implicitly.

Speaking informally, ISA relationships might be used in a semantic schema for two closely related purposes. The first is to represent one or more possibly overlapping subtypes of a type, as with the subtypes of PERSON shown in the World Traveler schema. The second purpose is to form a type that contains the union of types already present in a schema. For example, a type VEHICLE might be defined as the union of the types CAR, BOAT, and PLANE, or the type LEGAL-ENTITY might be the union of PERSON, CORPORATION, and LIMITED-PARTNER-

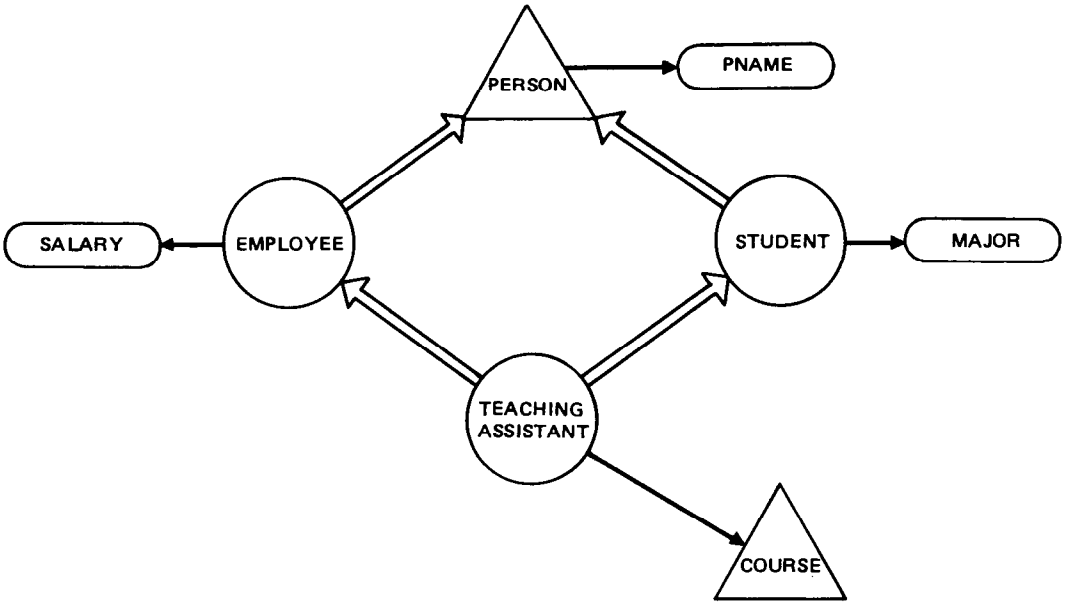


Figure 8. ISA network with undirected cycle.

SHIP. When using ISA for forming a union, it is common to include a *covering constraint*, which states that the (active domain of the) supertype is contained in the union of the (active domains of the) subtypes. Also, the semantics of update propagation varies for the different kinds of ISA relationships.

Historically, semantic models have used a single kind of ISA relationship for both of these purposes. Furthermore, several early papers on semantic modeling (including FDM and SDM) provide schema definition primitives that favor the specification of ISA networks from top to bottom. For example, in these models the type VEHICLE would be specified first, and subtypes CAR, BOAT, and PLANE would be specified subsequently. In contrast, the seminal paper [Smith and Smith 1977] uses ISA relationships to form unions of existing types.

More recent research on semantic modeling has differentiated several kinds of ISA relationship; and some models, including IFO, RM/T, Galileo, and extensions of the ER Model, incorporate more than one type of ISA into the same model. For example, in the extension of the ER Model described in Teorey et al. [1986], *subset* and *generalization* ISA relationships are supported. A subset ISA relationship arises when one type is contained in another; this is the notion already discussed in connection with the GSM. Generalization ISA relationships arise when one type is *partitioned* by its subtypes, that is, when the subtypes are disjoint and together cover the supertype. Generalization ISA relationships could thus be used for the VEHICLE and LEGAL-ENTITY types mentioned above. As noted in Abiteboul and Hull [1987] and Teorey et al. [1986], the update semantics of these two constructs are different. For example, in the first case deletion of an object from a subtype has no impact on the supertype; in the second case deletion from a subtype also requires deletion from the supertype.

A second broad motivation for distinguishing kinds of ISA relationships stems from studies of schema integration [Batini et al. 1986; Dayal and Hwang 1984; Navathe et al. 1986; NEL86]. For example, Dayal and Hwang [1984] study the problem of integrating two or more FDM schemas. Suppose that two FDM schemas contain types EMP1 and EMP2, respectively, for

employees. To integrate these, a new type EMPLOYEE can be formed as the generalization of EMP1 and EMP2. This generalization may have overlapping subtypes but must be covered by them. Interestingly, Dayal and Hwang [1984] also permit ISA relationships between attributes.

2.3 Global Considerations

In Section 2.2 we discussed the constructs used in semantic models largely in isolation. This section takes a broader perspective and examines the larger issue of how the constructs are used to form schemas. The discussion is broken into three areas. The first concerns restrictions of an essentially structural nature on how the constructs can be combined, for example, that there be no directed cycles of ISA relationships. The second and third areas are two closely related mechanisms for extending the expressive power of schemas, namely, derived schema components and integrity constraints.

2.3.1 Combining the Local Constructs

Although many semantic models support the basic constructs of object construction, attribute, and ISA, they do not permit arbitrary combinations of them in the formation of schemas. Restrictions on how the constructs can be combined generally stem from underlying philosophical principles or from intuitive considerations concerning the use or meaning of different possible combinations. Such restrictions have also played a prominent role in theoretical investigations of update propagation in semantic schemas [Abiteboul and Hull 1987; Hecht and Kerschberg 1981]. The restrictions are typically realized in one of two ways: in the definition of the constructs themselves (e.g., in the original ER Model, all attribute ranges are printable types) or as global restrictions on schema formation (e.g., that there be no directed cycles of ISA relationships). The following discussion surveys some of the intuitions and restrictions arising in construct definitions and then considers global restrictions on schema formation.

In the description of the local constructs given in Section 2.2, relatively few restrictions are placed on their combination. For example, aggregation and grouping can be used recursively, and attributes can have arbitrary domain and range types. Indeed, part of the design philosophy of the GSM was to present the underlying constructs in as unrestricted a form as feasible in order to separate fundamental aspects of the constructs from their usage in the various semantic models of the literature. In contrast with the GSM, many semantic models in the literature present constructs in restricted forms; for example, some models permit aggregations in attribute domains but not as attribute ranges or in ISA relationships.

Restrictions explicitly included in the definition of constructs are essentially local. However, these restrictions can affect the overall or global structure of the family of schemas of a given model. A dramatic illustration of this is provided by the original ER Model [Chen 1976]. In that model, aggregation can be used only to combine abstract types. As a result, schemas from the model have a two-tier character; with abstract types in one level and aggregations in the second. Attributes may be defined on both abstract types or aggregations, but they must have ranges of printable type.

We conclude our discussion of local constructs by attempting to indicate why certain models introduce restrained versions of constructs. Intuitively, a model designer tries to construct a simple yet comprehensive model that can represent a large family of naturally occurring applications. Thus, for example, FDM allows grouping only in attribute ranges. As illustrated in the discussion of COMMITTEEs in Section 2.2.1 (see Figure 5b), grouping objects are rarely of interest in isolation.

In addition to restricting the use of constructs at the local level, many semantic models specify global restrictions on how they may be combined (including notably Abiteboul and Hull [1987]; Brodie and Ridjanovic [1984]; Brown and Parker [1983]; Dayal and Hwang [1984]; Hecht and Kerschberg [1981]). The most prominent restrictions of this kind concern the

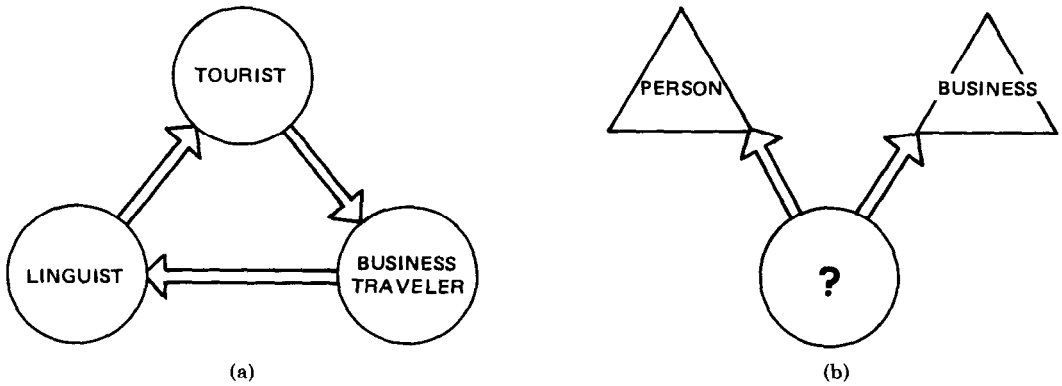


Figure 9. "Schemas" violating intuitions concerning ISA.

combining of ISA relationships. More recently, the interplay between constructed types and ISA relationships has also been studied. To give the flavor of this aspect of semantic models, we present a representative family of global restrictions on ISA relationships. It should also be noted that several models [Albano et al. 1985; Hammer and McLeod 1981; King and McLeod 1985a; Shipman 1981; Su 1983] do not explicitly state global rules of this sort but nevertheless imply them in the definitions of the underlying constructs.

To focus our discussion of ISA restrictions, we consider only abstract types. This coincides with most early semantic models, including FDM and SDM. In schemas for these models, a family of *base types* is viewed as being defined first, and subtypes are subsequently defined from these in a top-to-bottom fashion. The World Traveler schema follows this philosophy, as does the example in Figure 8. In the GSM, subtypes are depicted using a subtype (circle) node, indicating that they are not base types. To enforce this philosophy, we might insist that the tail of each specialization edge is a subtype node and the head of each specialization edge is an abstract or subtype node.

A second general restriction on ISA involves directed cycles. Consider the "schema" of Figure 9a. (We use quotes because this graph does not satisfy the global restriction we are about to state.) It suggests that TOURIST is a subtype of BUSINESS-TRAVELER, which is a subtype of LINGUIST, which is a subtype of TOURIST. Intuitively, this cycle implies that the three types are *redundant*; that is, in every instance, the three types will contain the same set of objects. Furthermore, if the cycle is not connected via ISA relationships to some abstract type, there is no way of determining the underlying type (e.g., PERSON) of any of the three types. Thus, we might insist that there is no directed cycle of ISA edges.

In the "schema" of Figure 9b, the type labeled ? is supposed to be a subtype of the abstract type PERSON and also of the abstract type BUSINESS. If we suppose that the underlying domains of PERSON and BUSINESS are disjoint, then in every instance the node labeled ? will be assigned the empty set. Speaking intuitively, the ? node cannot hold useful information. So, we might insist that any pair of directed paths of ISA edges originating at a given node can be extended to a common node.

The above discussion provides a *complete* family of restrictions on ISA relationships for the GSM considered without type constructors. Speaking informally, the rules are complete because they capture all of the basic natural intuitions concerning how ISA relationships (of the top-to-bottom variety) must be restricted in order to be meaningful. On a more formal level, it can be shown that, if a schema satisfies these rules, then every node will have an unambiguous underlying type, no pair of nodes will be redundant, and every node will be

satisfiable in the sense that some instance will assign a nonempty active domain to that node.

The set of rules given above applies to the special case of abstract types and top-to-bottom ISA relationships. As discussed in Section 2.2.3, some models support different kinds of ISA relationships. Furthermore, in some models constructed types can participate in ISA relationships. Specification of global rules in these cases is more involved; the IFO model presents one such set of rules [Abiteboul and Hull 1987].

2.3.2 Derived Schema Components

Derived schema components are one of the fundamental mechanisms in semantic models for data abstraction and encapsulation. A derived schema component consists of two elements: a structural specification for holding the derived information and a mechanism for specifying how that structure is to be filled, called a *derivation rule.* (Keeping with common terminology, we refer to derived schema components simply as "derived data.") Derived data thus allow computed information to be incorporated into a database schema.

In published semantic models the most commonly arising kinds of derived data are *derived subtypes* and *derived attributes.* Each of these is illustrated in the World Traveler schema: LINGUIST is a derived subtype of PERSON that contains all persons who speak at least two languages, and LANG-COUNT is a derived attribute that gives the number of languages that members of LINGUIST speak. In queries, users may freely access these derived data in the same manner in which they access data from other parts of the schema. As a result, the specific computations used to determine the members of LINGUIST and the value of LANG-COUNT are invisible to the user. The derivation rules defining derived data can be quite complex, and moreover, they can use previously defined derived data.

In any given semantic model, a language for specifying derivation rules must be defined. In the notable models supporting derived data [Hammer and McLeod 1981; King and McLeod 1985a; Shipman 1981], this language is a variant of the first-order predicate calculus, extended to permit the direct use of attribute names occurring in the schema, the use of aggregate attributes, and the use of set operators (such as set membership and set inclusion). This is discussed further in Section 2.4. (Although not traditionally done, the language for specifying derivation rules can, in principle, allow side effects.)

To illustrate the potential power of a derived data mechanism, we present an example that could be supported in the DBMS CACTIS [Hudson and King 1986]. Figure 10 shows a schema involving BUSINESS-TRAVELERs and TRIPs they have taken. The derived attribute TOTAL-MILES-TRAVELED is also defined on business travelers. The attribute uses two pieces of information: the TRIP attribute of BUSINESS-TRAVELER and the ADDRESS attribute of BUSINESS. TRIP consists of ordered pairs of DATE and CITY, each representing one business trip. The definition of TOTAL-MILES-TRAVELED is based on a derivation rule that is a relatively complex function. For each city traveled to on a trip, this function computes the distance between that city and the city the individual works in. Then, the distances are summed and multiplied by 2 to give the total miles traveled per individual. This distance information may be stored elsewhere in the database or elsewhere in the system.

To illustrate further the power of derived data, we present an example showing the interplay of derived data with schema structures. The example also provides a useful comparison of the notions of grouping, subtype, and metatype. Figure 11 shows three related ways of modeling categorizations of people on the basis of the languages they can speak. Figure 11a is taken from SDM and uses the grouping construct in conjunction with a derivation rule stating that the node should include sets of people grouped by the languages they speak. In an instance, this type would include the set of persons who speak French, the set of persons who speak

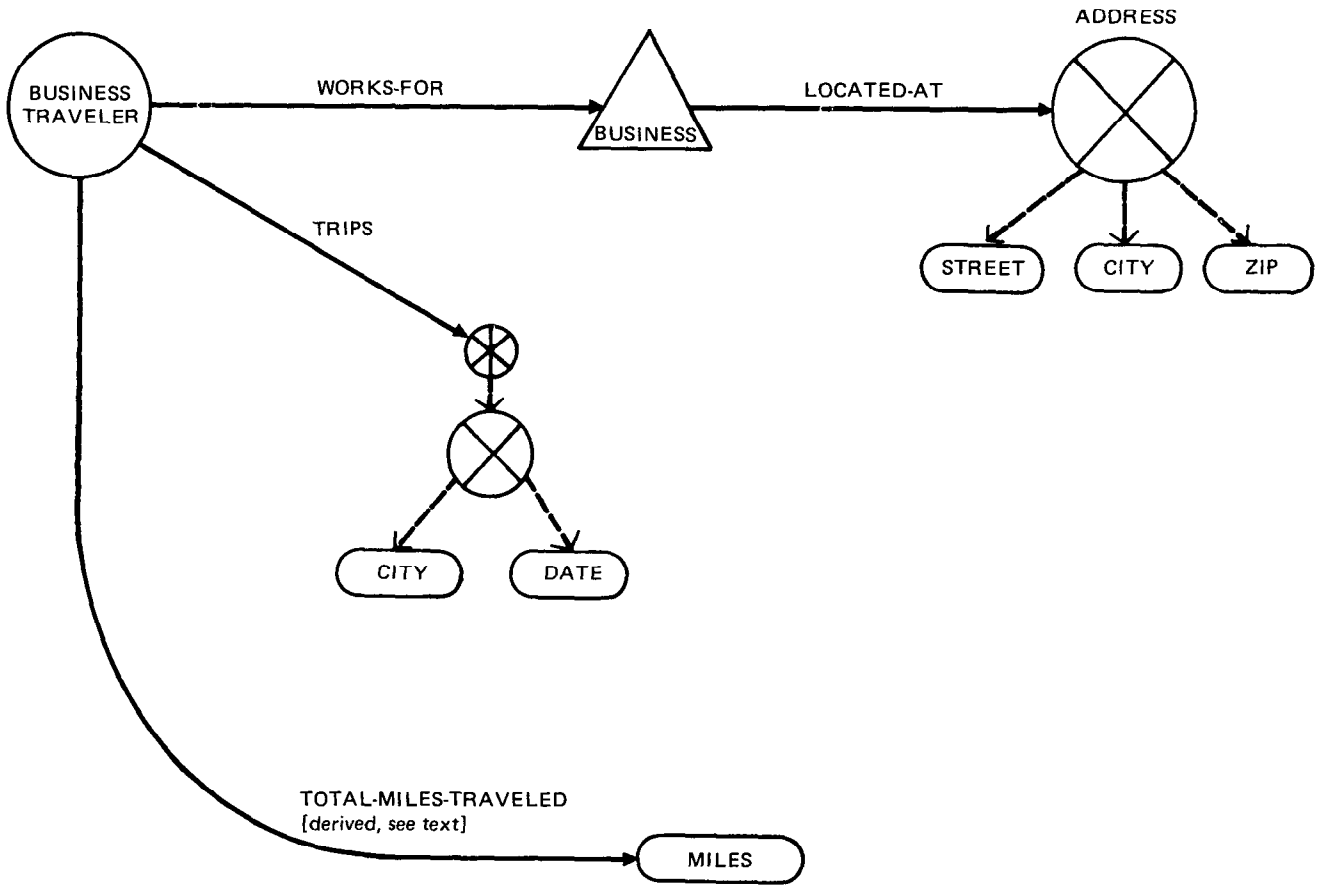


Figure 10. Schema used in example of derived attribute.

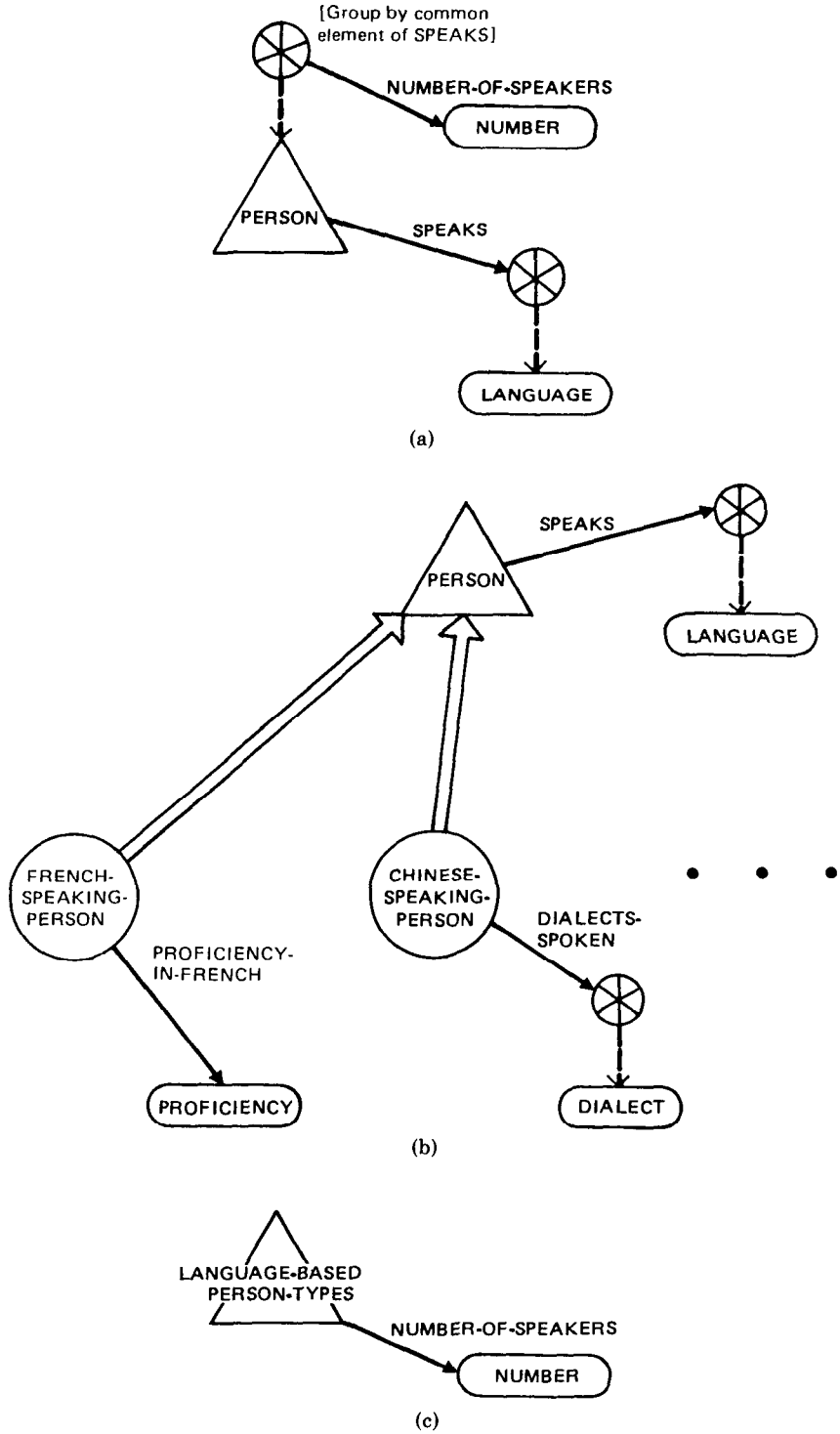


Figure 11. Related uses of derived schema components. (a) Expression-defined grouping type as in SDM. (b) Derived subtypes (derivation rules not shown). (c) Metatype whose elements are types, as in TAXIS.

Chinese, and, more generally, a set of persons for each of the languages in the database. These sets are accessed in queries by referring to languages. (This construction is closely related to forming the inverse function $SPEAKS^{-1}$.) In the example, we also define a (nonderived) attribute on the grouping type.

The schema of Figure 11b includes a derived subtype for each of the languages that arises. In this representation different attributes can be associated with each of the subtypes. Importantly, the number of subtypes is equal to the number of languages arising in the underlying instance, whereas in the schema of Figure 11a, only one additional type is used. Although not shown here, type attributes can be defined on the subtypes to record information on the number of speakers of each language.

The schema of Figure 11b can be extended to include the graph of Figure 11c, which shows the use of a *metatype*, as found in TAXIS. The elements of this metatype are types from elsewhere in the schema. The derived attribute NUMBER-OF-SPEAKERS defined on this metatype shows a third way of obtaining this cardinality information.

Several models, including FDM and SDM, view the specification of derived data as part of the schema design and/or evolution process, whereas others support a much more dynamic view. For example, in the implementation of INSYDE described in King [1984], users can specify derived data at any time and incorporate them as permanent in the schema. Indeed, in the graphics-based interface to this model [King 1984], database queries are formed through the iterative specification of derived data (see Section 4.3).

We close this section with a discussion of the interaction of derived data with database updates. Speaking in general terms, derived data are automatically updated as required by updates to other parts of the schema. For example, in the World Traveler Database, if a person who speaks one language learns a second, that person is automatically placed in the LINGUIST subtype, and the attribute LANG-COUNT is extended to this person. A subtlety arises if the user attempts to directly update data associated with a derived schema component. In many cases such updates would have ambiguous consequences. For example, in an instance of the World Traveler Database, if someone were explicitly deleted from LINGUIST, the set of languages that person speaks would have to be reduced, but the system would not know which languages to remove.

In some cases explicit updates against a derived schema component might have an unambiguous impact on the underlying data. For example, updates on the FRENCH-SPEAKING-PERSON subtype of Figure 11b are easily translated into updates on the SPEAKS attribute. Importantly, FDM as described in Shipman [1981] provides facilities for specifying how updates to the derived data, if permitted at all, should be propagated in the underlying data. Interestingly, the derived update problem is related to the view update problem in relational databases [Cosmadakis and Papadimitriou 1984].

2.3.3 *Static Integrity Constraints*

As is clear from the above discussion, the structural component of semantic models provides considerably more expressive power than that of the record-oriented models. However, there is still a wide variety of relationships and properties of relationships that cannot be directly represented using structure alone. For this reason, semantic models often provide mechanisms for specifying integrity constraints. The discussion here focuses on three topics: the relationship between semantic models and the prominent relational integrity constraints, prominent types of integrity constraints found in semantic models, and the differences between integrity constraints and derived data. Although integrity constraints can in principle focus on both the static and dynamic aspects of data [Tsichritzis and Lochovsky 1982; Vianu 1987], little research on dynamic constraints has been done relative to semantic models. For this reason, we focus on static integrity constraints.

Broadly speaking, semantic models express in a structural manner the most

important types of relational integrity constraints, namely, key dependencies and inclusion dependencies. As suggested by the World Traveler schema in Figure 1 and the associated relational schema in Figure 2, relational key dependencies can be represented using single-valued attributes. Inclusion dependencies arising from subtyping can be represented using ISA relationships. Inclusion dependencies that serve as *referential constraints* are typically modeled in an implicit manner in semantic schemas. For example, the dependency BUSTRAV[EMPLOYER] $\subseteq$ BUSINESS[BNAME] in the relational schema is represented in the semantic schema by the fact that the attribute edge WORKS-FOR points directly to the BUSINESS node as its range. Interestingly, some examples of multivalued dependency [Fagin 1977; Zaniolo 1976] are naturally modeled using multivalued attributes.

We now turn to the various kinds of constraints used in semantic models. Many of these focus on restricting the individual constructs occurring in a schema. On attributes, such constraints include restrictions that they be 1–1, onto, or total. For example, in the World Traveler schema, the HAS-NAME attribute is restricted to be 1–1 and total. ISA relationships can also be constrained in various ways. For example, a *disjointness constraint* states that certain subtypes of a type are disjoint (e.g., that no TOURIST is a BUSINESS-TRAVELER). A *covering constraint* states that a set of subtypes together covers a type. In some investigations, these constraints are applied to types that need not be related by ISA edges [Lenzerini 1987].

An important class of constraints on constructs restrict *cardinalities* in various ways. Perhaps the best known types of cardinality constraint are found in the ER Model: These specify whether a binary aggregation (relationship) is 1:1, 1:N, N:1, or M:N. For example, in Figure 3b, the aggregation EMPLOYMENT between PERSON and BUSINESS is constrained to be N:1. In each instance of this schema, several (N) people can be associated with a given business, but only one (1) business can be associated with a given person. Multivalued attributes can be restricted in a similar manner: An attribute mapping students to courses might be restricted to be [1:6], meaning that each student must be taking at least one course but no more than six courses. As detailed in Section 3.2, the IRIS data model permits the specification of several cardinality constraints on the same n-ary aggregation, thereby providing considerable expressive power.

Another prominent constraint is an *existence* constraint. This is related to a relational inclusion dependency and states that each entity of some type must occur in some aggregation. Consider the schema of Figure 3b, which represents the aggregation EMPLOYMENT. It makes no sense in this particular application for a business to exist in the database unless it participates in an EMPLOYMENT aggregation for at least one employee. To enforce this, we would say that there is an existence dependency between BUSINESS and EMPLOYMENT. It is also natural to place existence dependencies on attribute ranges.

The semantic modeling literature has also described constraints that are computed in nature; such constraints may involve schema components that are arbitrarily separated. These constraints are generally specified using a predicate describing properties of data taken from disparate parts of a schema. Such constraints in the World Traveler Database, for example, can state that for each business-traveler p, the city of p's employer is equal to the city where p lives or that the number of persons living in a given zip-code area is no greater than 10,000. Although several authors have suggested the usefullness of computed constraints in principle [Hammer and McLeod 1981; King and McLeod 1985a; Tsichritzis and Lochovsky 1982], no models in the literature support them formally.

There is a close relationship between integrity constraints and derived schema components. Both require that data associated with different parts of a schema be consistent according to some criteria. The essential difference is that an integrity

constraint does not extend the database with any new information, whereas derived data truly augment the database.

2.4 Manipulation Languages

Up to this point we have provided an overview of the data structuring mechanisms supported by typical semantic models. These capabilities would normally be supported by a *data definition language* associated with a specific model. No data model is complete without a corresponding *data manipulation language*, which allows the database user to create, update, and delete data that correspond to a give schema. In this section, we describe the general structure of a data manipulation language for the GSM and use it as a means of discussing the general nature of semantic data manipulation.

There are three fundamental capabilities that differentiate a semantic data manipulation language from a manipulation language for a traditional record-oriented model. First, the language must be able to query abstract types. Second, it must provide facilities for referencing and manipulating attributes. In this way, abstract, nonprintable information may be manipulated. Third, semantic manipulation languages often allow the user to manage derived data in the form of subtypes and functions constructed from existing (sub)types and functions. Thus, the specification of derived data is not reserved for the user of the data definition languages but may also be performed at run time. This blurs to some degree the traditional boundary between schema and data; the user's view of the world may now be extended dynamically with new information constructed from existing data. This provides a marked contrast with approaches taken in record-oriented models, in which the data definition and data manipulation languages are quite distinct.

Semantic data manipulation languages represent diverse programming language paradigms, but there are strong commonalities in terms of their functionality. Essentially, a semantic manipulation language typically takes the form of an extension to a language resembling a relational query language. Some semantic manipulation languages also include the flow-of-control and computational capabilities of general-purpose imperative programming languages. The GSM data manipulation language is a simple SEQUEL-like language.

Here is a query that lists the names of all linguists who speak three or more languages; it illustrates the basic capabilities of a semantic access language to manipulate types and functions:

```
for each X in LINGUIST
  such that LANGCOUNT(X) ≥ 3
print PNAME(X)
```

The next query prints any address such that more than one person resides at the given address:

```
for each X in ADDRESS
  such that for some Y in PERSON
  and for some Z in PERSON
    Y ≠ Z and
    ADDRESS(Y) = X and
      ADDRESS(Z) = X
print X.STREET, X.CITY, X.ZIP
```

Note that the "." notation is used to reference the various components of an aggregation. It is also true that if, for example, an address could have two components of the same type (e.g., two ZIPs), this notation would create an ambiguity. In general, it is necessary to be able to give names to the components of an aggregation and to reference them by those names, rather than by their types.

The following query illustrates the capability of a semantic language to manipulate derived information:

```
create subtype ROMANCE-LINGUIST of
    LINGUIST
  where SPEAKS includes French, Italian,
      Spanish, Portuguese, Rumanian,
      Sardinian
for each X in ROMANCE-LINGUIST print
    PNAME(X)
record ROMANCE-LINGUIST
```

The query creates a subtype, called ROMANCE-LINGUIST, of all linguists who speak French, Italian, Spanish, Portuguese, Rumanian, and Sardinian. Then the names of all romance linguists are printed, and the subtype is permanently recorded in the database schema. When a query specifies a derived subtype, it must be possible to name the subtype in order to reference it later. Again, we note that as a direct result of their rich modeling capabilities, semantic models require the creation of names that would not exist in a corresponding relational schema. Since such things as aggregations and subtypes may be created and referenced, they need names. This can be viewed as a limitation to the casual user who might feel that a semantic model causes a proliferation of names and therefore creates confusing schemas.

In the examples presented above, the output of the queries was a list of objects or values, not instances of semantic types. This is quite different from relational queries, which take relations as input and produce relations as output. As a result, in most semantic languages operations cannot be composed. Notably, the language FQL does not suffer from this limitation (see Section 3.5).

3. SURVEY

In this section we survey a number of semantic models. In particular, we discuss the first ten models (four horizontal groups) listed in Figure 12. We begin, in Section 3.1, with three models that are highly prominent in the literature. These are the Entity–Relationship (ER) Model, the Functional Data Model (FDM), and the Semantic Data Model (SDM). Then we briefly consider a number of other semantic models in Sections 3.2–3.4. Finally, in Section 3.5 we review the prominent semantic data manipulation languages.

The models of Sections 3.1 and 3.2 embody a number of explicit, distinct constructs in support of complex data modeling. Section 3.3 considers the binary models that offer only a minimal set of simple constructs, which are then used to build up more complex structures. In Section 3.4 we consider models that represent complex data by extending the relational model. The models in the last two horizontal groups of Figure 12 focus primarily on the research goals of encapsulating transaction facilities and theoretical investigations. These models are discussed in Section 4. (In this and all subsequent summary tables, a blank entry indicates that the specified feature is not present to the best of the authors' knowledge.)

The three prominent models and those discussed in Section 3.2 all explicitly support constructs for defining semantic databases. This approach has the advantage of providing a refined set of powerful modeling capabilities that the database designer and user may quickly comprehend. In contrast, the binary and relational extension models represent two very different philosophical approaches. The binary models take a building block approach in that they support only simple constructs that are then used to develop more complicated ones. This minimalist approach has the advantage of being more general; the models are very simple object-oriented ones that allow the designer to develop a wide variety of modeling constructs. In contrast, the relational extensions rely on underlying relational primitives to support higher level constructs. This approach has the advantage of being able to draw on a large body of knowledge concerning relational databases, which is useful in developing implementations and in enriching a system with integrity mechanisms, design methodologies, query optimization, and transaction specification facilities.

Figure 12 describes the various semantic models according to their structural and dynamic aspects. There are four main categories at the top of the figure: *References* indicates references to initial research on the models. *Philosophical Basis* classifies the models along three spectras: their primary research objectives, the nature of their underlying modeling primitives, and their general modeling philosophy. The research objective of each model is defined as providing a general-purpose semantic model, a basis for a structured design methodology, a programming language for

		REFERENCES	PHILOSPHICAL BASIS			SCHEMA COMPONENTS							DYNAMIC COMPONENTS		
						TYPE CONSTRUCTORS		ATTRIBUTES			ISA RELATIONSHIPS	DERIVED SCHEMA COMPONENTS			
			RESEARCH OBJECTIVES	UNDERLYING MODELING PRIMITIVES	GENERAL MODELING PHILOSOPHY	AGGREGATION	GROUPING	PRINTABLE	OBJECT-VALUED	MULTI-VALUED			QUERY LANGUAGE	DATA MANIPULATION PRIMITIVES	TRANSACTION STRUCTURING
PROMINENT MODELS	ER	[Che76]	GENERAL PURPOSE	EXPLICIT	AGGREGATION	EXP	IMP	EXP	IMP	IMP			[Sho82]	[Che76].	
	FDM	[KeP76, Shi81]	GENERAL PURPOSE	EXPLICIT	ATTRIBUTES	IMP	IMP	EXP	EXP	EXP	EXP	EXP	[Shi81, SFL81]	[Shi81, SFLa81]	
	SDM	[HaM81]	GENERAL PURPOSE	EXPLICIT	ATTRIBUTES/TYPE CONSTRUCTORS	IMP	EXP	EXP	EXP	EXP	EXP	EXP	[HaB80]	[HaB80]	
OTHER HIGHLY SRUCTURED MODELS	*SAM**	*[Su83]*	*GENERAL PURPOSE*	*EXPLICIT*	*TYPE CONSTRUCTORS*	*EXP*	*EXP*	*EXP*	*IMP*	*IMP*	*EXP*	*EXP*			
	IFO	[AbH87]	BASIS FOR THEORETICAL INVESTIGATION	EXPLICIT	TYPE CONSTRUCTORS	EXP	EXP	EXP	EXP	EXP	EXP				
	IRIS	[DKL85]	GENERAL PURPOSE	EXPLICIT	ATTRIBUTES	IMP	IMP	EXP	EXP	EXP	EXP	EXP	[DKL85]	[DKL85]	
BINARY MODEL	SBDM	[Abr74]	GENERAL PURPOSE	MINIMALIST	ATTRIBUTES	IMP	IMP	EXP	EXP	EXP	IMP	EXP	[Abr74]	[Abr74]	[Abr74]
RELATIONAL EXTENSIONS	Structural Model	[WiE80]	GENERAL PURPOSE	RELATIONAL EXTENSION	AGGREGATION	EXP	IMP	IMP	IMP	IMP	EXP		[Elm80]	[Elm80]	
	RM/T	[Cod79]	GENERAL PURPOSE	RELATIONAL EXTENSION	AGGREGATION	EXP	IMP	IMP	IMP	IMP	EXP		[Cod79]	[Cod79]	
	GEM	[Zan83, TsZ84]	GENERAL PURPOSE	RELATIONAL EXTENSION	AGGREGATION	EXP	IMP	IMP	IMP	IMP	EXP		[Zan83, TsZ84]	[Zan83, TsZ84]	
MODELS FOCUSED ON DYNAMICS	*TAXIS*	*[M8W80]*	*PROGRAMMING LANGUAGE FOR* DATABASE APPLICATIONS	*EXPLICIT*	*ATTRIBUTES*	*IMP*	*IMP*	*EXP*	*EXP*	*EXP*	*EXP*	*EXP*	*[M8W80, MBG86]*	*[M8W80, MBG86]*	*[MBW80, MG886]*
	Galileo	[ACO85]	PROGRAMMING LANGUAGE FOR DATABASE APPLICATIONS	EXPLICIT	TYPE CONSTRUCTORS	IMP	IMP	EXP	EXP	EXP	EXP	EXP	[ACO85]	[ACO85]	[ACO85]
	INSYDE	[KiM85b]	BASIS FOR STRUCTURED DESIGN METHODOLOGY	EXPLICIT	ATTRIBUTES	EXP	IMP	EXP	EXP	EXP	EXP	EXP	[KiM85b]	[KiM85b]	[KiM85b]
	SHM+	[BrR84]	BASIS FOR STRUCTURED DESIGN METHODOLOGY	EXPLICIT	TYPE CONSTRUCTORS	EXP	EXP	IMP	IMP	IMP	EXP		[BrR84]	[BrR84]	[BrR84]
MODELS FOCUSED ON THEORY	Format	[HuY84]	BASIS FOR THEORETICAL INVESTIGATION	EXPLICIT	TYPE CONSTRUCTORS	EXP	EXP								
	LDM	[KuV84, Kup85]	BASIS FOR THEORETICAL INVESTIGATION	EXPLICIT	TYPE CONSTRUCTORS	EXP	EXP	IMP	IMP	IMP	IMP		[KuV84, Kup85]		

Figure 12. Summary of prominent Semantic Data Models. Blanks indicate capability not supported to the authors' knowledge. References in this figure are keyed to the reference list.

database applications, or a basis for theoretical investigation. The primitives in a given model are classified as being either explicit, minimalist, or relational extensions. The modeling philosophies are classified as being centered around attributes, aggregation, or more general type constructors, including aggregation (see Section 2.1).

The *Schema Components* category concerns the specific schema components supported by the individual models. We consider four capabilities: type constructors (including aggregation and grouping), attributes (including printable, object-valued, and multivalued attributes), ISA relationships, and derived schema components. In the figure, we indicate whether a capability is supported explicitly or implicitly, using combinations of other constructs (see Section 2). It is true, of course, that the lack of one of these capabilities or the choice of representing it implicitly rather than explicitly is not necessarily a deficiency; the underlying philosophy of a model is often simpler and cleaner if the number of constructs is kept minimal.

The *Dynamic Component* category concerns dynamic capabilities that are associated with each model. Specifically, literature references are provided to research that has been performed in the support of semantic query languages, data manipulation primitives, and transaction structuring. (Query languages and data manipulation primitives are discussed in Section 2.4, a survey of the prominent query languages is given in Section 3.5, and transaction structuring is discussed in Section 4.2.)

The general philosophy of semantic modeling has been applied to the development of special-purpose database models for applications such as very-large-scale integration (VLSI) design [Batory and Kim 1985]. Although these efforts are of significant interest and further validate the usefulness of semantic modeling, they are beyond the scope of this paper.

3.1 Prominent Models

In this section we consider three well-known models from the literature, which are considered by many to be the most influential semantic models. We see that, even though they are all explicit models, they vary greatly in their modeling capabilities and in their philosophical approach to database design. The first model constructs complex objects out of aggregations, which are visualized at the same level of abstraction as types. Consequently, such aggregations can be considered irrespective of the types they interrelate; that is, they are semantic entities on their own (see Section 2.1). In contrast, the two latter models use attributes as their primary method of constructing complex data; attributes are viewed as secondary to types and have no meaning at all without defining domain and range types. Thus, we see that the prominent models are representative of the two general modeling approaches described in Section 2.1.

3.1.1 The Entity–Relationship Model

The *Entity–Relationship* (ER) Model, proposed by Chen in 1976, is generally considered to be one of the first true semantic data models to appear in the literature, although the term "semantic" was not in use at the time. Schemas of this model have a natural graph-based representation and support the representation of abstract sets of entities, relationships between these entity sets, and attributes defined from both entity and relationship sets to printable values. Thus, an ER schema consists of types and relationships interconnecting these types, along with printable attributes of the types and relationships. Relationships can be restricted to 1:1, many:1, or many:many, and attributes can be restricted to 1:1.

Figure 13 shows a portion of the World Travelers schema using the standard ER notation. Rectangles represent *entity sets*, that is, abstract types, diamonds represent *relationships*, that is, aggregation, and ovals represent the ranges of attributes. The ER schema of Figure 13 illustrates a few differences between the original ER Model and other, later semantic models. The most fundamental difference is that ISA relationships are not represented. Recently, the ER Model has been extended to include ISA relationships [Batini et al. 1986; Teorey et al. 1986] (see Section 2.2.3).

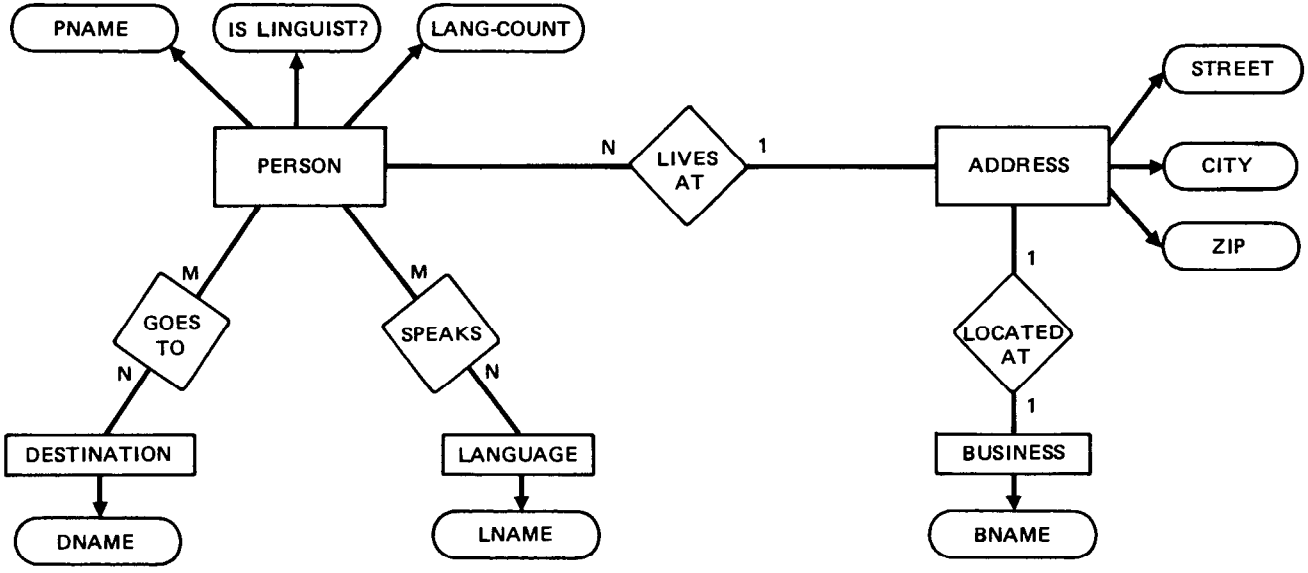


Figure 13. ER representation of part of the World Traveler schema.

Perhaps the most distinctive feature of the ER Model is the way in which it restricts the use of attributes and aggregation. Real-world attributes whose ranges are printable can be represented directly in the ER Model. For example, in Figure 13 ADDRESS has three printable attributes, STREET, CITY, and ZIP. On the other hand, real-world attributes that map to entity sets must be represented using relationships, as illustrated by the LIVES-AT relationship. (This relationship is shown to be many:1 because several persons might share the same address.) Because relationships are given names and, in a sense, viewed as entities themselves, it is straightforward in the ER Model to represent attributes of relationships, as illustrated in Figure 3b.

In the original ER Model, multivalued attributes also require the use of a relationship. This is because, as discussed above, attributes must be single valued. Thus, to represent the fact that a person may speak more than one language, the relationship SPEAKS is used to collect a number of languages into a set. The underlying philosophy is that an attribute is restricted to being a single fact about an entity, whereas a relationship can model the construction of more complex entities from other entities.

The ER Model was originally proposed [Chen 1976] as a schema design aid, permitting users to design schemas using a high-level object-based approach. The resulting ER schema would then be translated into either the relational or the network model. Within this framework, an ER schema is designed primarily for the purpose of articulating the overall data management objectives of an organization, but is not implemented per se. A detailed design methodology based on a generalization of the ER Model appears in Teorey et al. [1986].

In sum, the ER Model was the first semantic model centered around relationships, not attributes. It views the world as consisting of entities and relationships among entities. Both entities and relationships may have single-valued printable attributes. In the original ER Model ISA relationships are not supported, but recent researchers have proposed mechanisms for supporting ISA within the ER Model.

3.1.2 The Functional Data Model

The *Functional Data Model* (FDM) was introduced in 1976 [Kerschberg and Pacneco 1976] and is recognized as the first semantic model centered around functional relationships, that is, attributes. Like the ER Model, a considerable amount of research has developed around FDM, and several other semantic models have adopted the attribute-based approach. Attributes in FDM can be either single- or multivalued and can be defined on domains that are Cartesian products of the atomic entity sets. FDM also supports ISA relationships. Significantly, the work of Shipman [1981] on FDM is among the first to include derived schema components as an integral part of a semantic model.

An informal graph-based representation of FDM schemas is introduced in Shipman [1981] and extended in Dayal and Hwang [1984] and elsewhere. An FDM schema corresponding to the World Traveler schema is shown in Figure 14. FDM connects objects directly with attributes without the use of intermediate constructs such as aggregation and grouping. This may be viewed as producing simpler schemas.

The data language DAPLEX [Shipman 1981] for this model was the first integrated data definition and access language formulated entirely in the high-level terms provided by an object-oriented semantic database model. DAPLEX was also the first database access language to give a prominent role to attributes, permitting their direct usage and also the use of their inverses and compositions. This and other semantic data access languages are discussed in Section 3.5.

FDM has spawned several research projects. It has been used to provide a unified interface to distributed heterogeneous databases in the Multibase project [Landers and Rosenberg 1982; Smith et al. 1981]. Integration of FDM schemas is studied in Dayal and Hwang [1984]. FDM also served as the basis for one of the

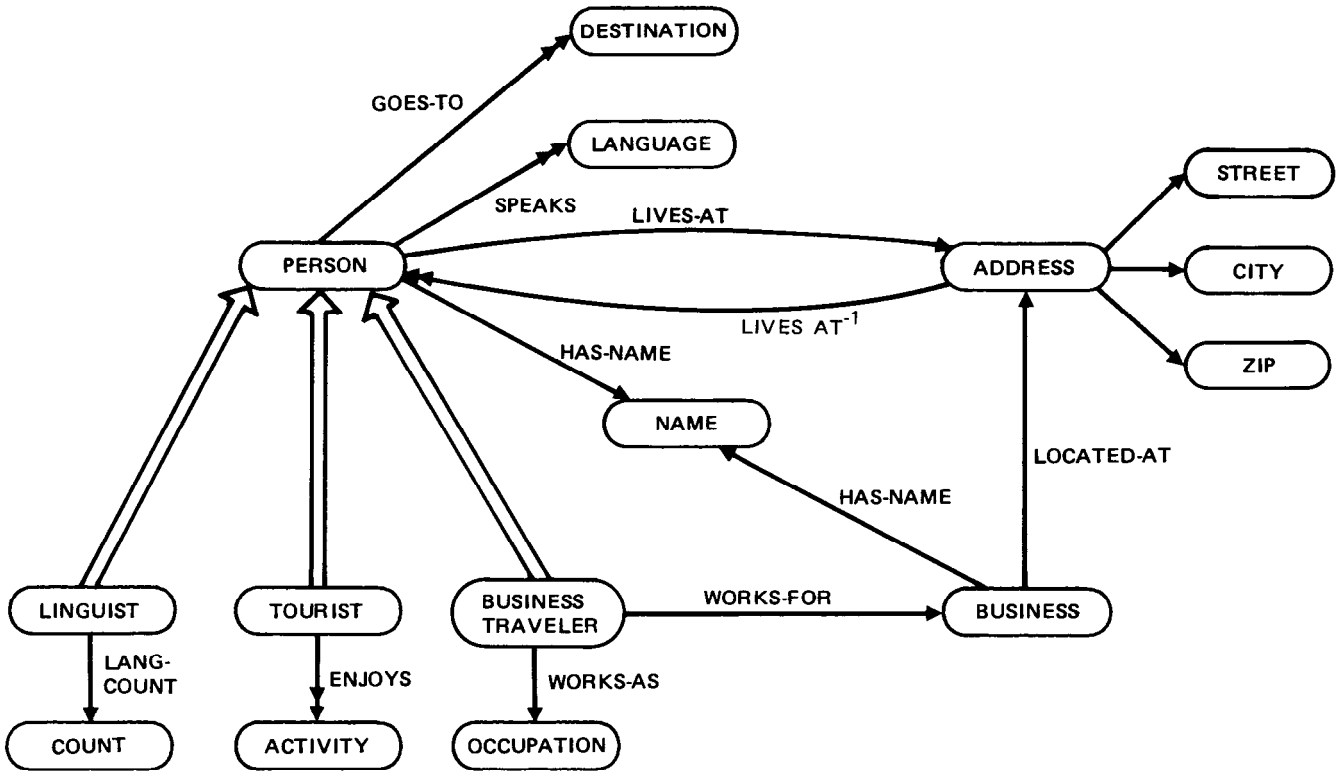


Figure 14. FDM representation of World Traveler schema.

original studies of update propagation in the context of semantic interconnections between data [Hecht and Kerschberg 1981] (see Section 4.4).

In summary, FDM was the first of a number of semantic models based on explicit representation of attributes with printables or objects as their ranges. It is a simple, elegant model with an easily understood visual representation. It gives prominence to atomic types and attributes, not to type constructors like aggregation and generalization. One of the major benefits of this model is the capacity to reference functions directly when manipulating properties of data.

3.1.3. The Semantic Database Model

The *Semantic Database Model* (SDM) [Hammer and McLeod 1981] was among the first published models to emphasize the use of the grouping constructor and the support of derived schema components. In particular, derived schema components permit *data relativism*, that is, multiple perspectives on the same underlying data set. SDM does not provide an explicit type constructor for aggregation, and in that sense it is attribute oriented (see Section 2); SDM does simulate aggregation with the attribute primitive.

SDM is unique in that it provides a rich set of primitives for specifying derived attributes and subtypes. For example, subtype relationships in SDM are broken into four categories: (i) those that are *attribute defined*, (ii) those defined by set operations (e.g., intersection) on existing types, (iii) those that serve as the range of some attribute, and (iv) those that are user specified (or *user controllable* in the terminology of Hammer and McLeod [1981]).

As an example of the second sort of subtype, we might form a subtype called RETIRED-TOURIST (retired people who travel), give it the same properties of TOURIST, and union it with TOURIST to give the new subtype ALL-TOURIST. An example of a subtype that exists explicitly to serve as the range of an attribute would be if TOURIST in the World Traveler schema were the range of an attribute, say HAS-BEEN-VISITED-BY, of a type COUNTRY. An example of the fourth sort of subtype would be SUSPICIOUS-TOURISTS, whose contents would be updated directly by the end user on the basis of personal criteria. Primitives for specifying derived attributes are also supported in SDM.

One of the predicates used in conjunction with the grouping construct is of particular interest because it provides expressive power of a structural nature. For example, in the World Traveler Database an *enumerated grouping class* called EMPLOYABLE-PERSON-TYPES can be defined to hold the already existing types LINGUIST and BUSINESS-TRAVELER. This is roughly equivalent to forming a metatype from a user-specified set of types. Attributes of this new type (e.g., number of elements, median income) are type attributes on the underlying types. Although this ability to view types as both sets and individual elements is found in only a few semantic data models (SAM* and TAXIS being other notable models), it is commonly supported by frame-based approaches to knowledge representation in AI [Fikes and Kehler 1985].

The richness of SDM as a schema specification language highlights the trade-off in semantic modeling between providing a small or large number of primitive data structuring constructs. In models with a small number of constructs, the representation of some data sets requires the artificial combination of these constructs; in a model with many constructs such as SDM, the designer is continually forced to choose from among a variety of ways of representing the same data. Thus, a model like FDM might seem stark compared with SDM, but for some users it might prove easier to learn and use.

SDM refines the notions of subtype and attribute by considering how they are defined. However, it does not support an explicit aggregation type constructor. Of our three prominent models, it appeared in the literature most recently. SDM, like FDM and unlike the ER Model, is centered

around attributes, but it is richer (and thus more complex) than either FDM or the ER Model.

3.2 Other Highly Structured Models

In this section we consider three other highly structured models, the Semantic Association Model (SAM*), the IFO Model, and the IRIS Model. SAM* and IRIS have been developed to support full-fledged database applications, whereas IFO was developed primarily for theoretical investigation. SAM* focuses largely on special forms of the aggregation construct, and both IFO and IRIS include both type constructors and attributes.

The *Semantic Association Model* (SAM*) [Su 1983], an extension of SAM [Su 1980], attempts to provide a set of constructs rich enough to exhaust the possible relationship types that might arise in both commercial and statistical applications. The model distinguishes different uses of some of the fundamental structural constructs of semantic models and in some cases provides them with different update semantics. As a result, the model supports a limited form of data relativism whereby a given construct might be viewed as having two or more different underlying structures within the same schema. The paper [Su 1983] presents a graph-based representation for SAM* schemas and also suggests an approach to implementing SAM* based on data structures called *G-relations*, which are closely related to non-first-normal-form relations [Abiteboul and Bidoit 1984; Fischer and Thomas 1983; Jaeschke and Schek 1982; Makinouchi 1977]. Schema definition and data manipulation languages for SAM* are under development [Su 1986].

The basis for SAM* schemas is provided by what are called *atomic concepts* in Su [1983]. These include integer, real, character-string, and Boolean types, as well as structured programming language data types constructed from these, including vectors, arrays, and ordered and unordered sets, time, time series, text, and G-relations.

In SAM* physical or abstract objects are represented using *nonatomic concepts*. Seven kinds of *associations* or type constructors for building these concepts are used. The different associations have somewhat specific intended uses, and operations especially tailored to them.

We first consider four associations typical of most database applications. First, *membership association* permits the specification of the domain of a type or an attribute range. For example, a membership association could be used to specify that the type CITY has individual members Paris, Beijing, and so on. The SAM* representation of this is shown in Figure 15a. (The M in the root node indicates that it is a membership association.) The next two kinds of association are closely related to aggregation. The first of these is called *aggregation association* and is illustrated in Figure 15b. This kind of association is typically used to provide a description of an entity and its attributes. In the figure, PNAME is indicated to be a key. In terms of the GSM, SAM*'s *interaction association* also has the structure of aggregation. In SAM*, this is typically used to represent abstract concepts based on more concrete concepts. For example, the ENROLLMENT type (consisting of pairs of courses and students) can be modeled as an interaction association. Cardinality conditions can be specified for these associations. In some cases a given concept can be viewed in a SAM* schema as both an aggregation association and an interaction association. In this way different users can view the same schema in different ways. Finally, *generalization associations* are used to model ISA relationships. A generalization association forms the union of its underlying types. Constraints such as disjointness can be specified.

The remaining three associations focus on aggregate properties of sets and are especially suited for statistical applications. One of these is *cross-product association*. This form of association is again an aggregation in the terminology of GSM but typically holds *all* of the Cartesian product of the underlying types instead of a subset of

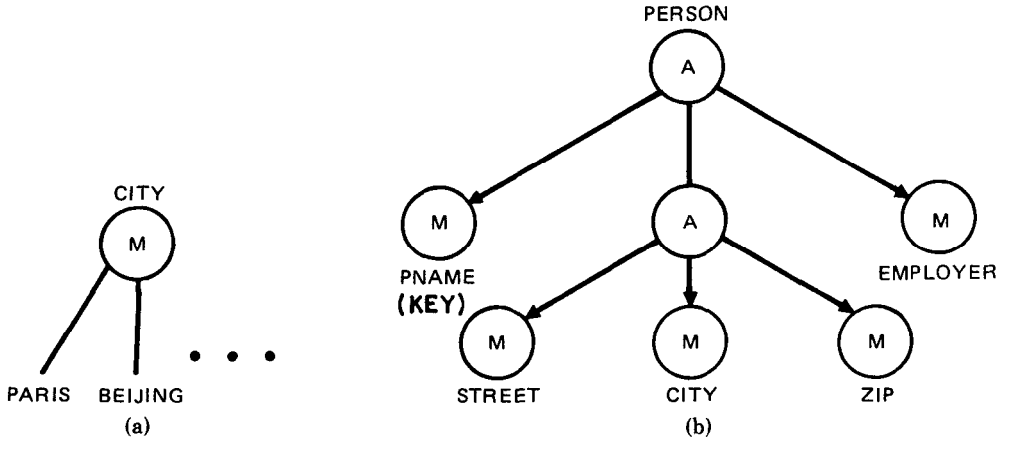


Figure 15. Membership and aggregation associations in SAM*.

that product. This association might be used to identify a variety of categories; for example, in a statistical database on populations it may be useful to consider categories formed from triples of AGE-RANGES, RACE, and SEX. These triples would then serve as the domain of functions describing statistical features of the different population groups they delimit. A *composition association* is used to hold a vector of sets. For example, a composition association for CURRENT-FLEET might be a triple, with three coordinates for the current sets of cars, trucks, and boats, respectively, that a business owns. In an instance, a composition association node will hold exactly one such tuple. Composition associations can participate in aggregations. Finally, *summarization association* is used to specify attributes for both cross-product associations and composition associations. These attributes typically hold statistical values and are thus a form of derived data. In the case of cross-product associations the attributes will be attributes in the sense of GSM; in the case of composition associations they will be type attributes.

To summarize, the seven kinds of associations used in SAM* have overlapping semantics but are distinguished by their associated update semantics and the constraints permitted. Although not discussed here, a variety of local restrictions is placed on how the constructs can be combined with each other, thereby ensuring that SAM* schemas are meaningful.

SAM* has recently been applied to the area of manufacturing data [Su 1986]. In particular, SAM* has been shown to be useful in representing the semantics of such complex data types as temporal data, recursively structured data, replicated data, and versions.

The *IFO Model* [Abiteboul and Hull 1987] was developed to provide a theoretical framework for studying the structural aspects of semantic data models. The model incorporates attributes and type constructors for aggregation and grouping at a fundamental level and distinguishes between two kinds of ISA relationships. The model is used in Abiteboul and Hull [1987] to characterize the propagation of simple updates in the presence of semantic model relationships and to analyze formally the interplay between constructed types and ISA relationships.

In many respects the IFO Model is similar to (and inspired) the structural portion of the GSM; a fundamental difference concerns how IFO combines the semantic constructs in forming schemas. The basic building block of an IFO schema is called a *fragment.* Fragments are used as abstraction mechanisms for representing an object type along with its internal structure and its attributes. Figure 16, which shows the IFO representation of a portion of the

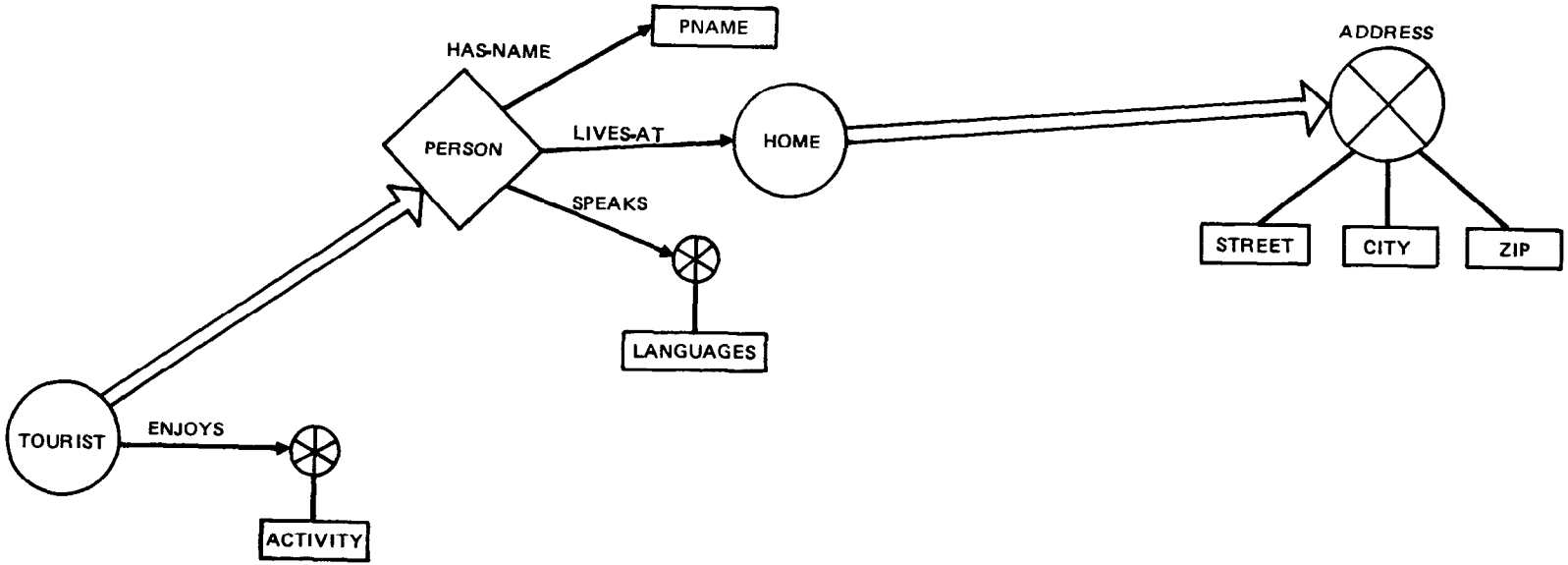


Figure 16. IFO representation of portion of World Traveler schema.

World Traveler schema, contains three fragments: one for PERSON, one for TOURIST, and one for ADDRESS. The fragment for PERSON illustrates how IFO clusters information about a type and its attributes. In particular, the HOME node is used as a place holder for the range of LIVES-AT. This node is shown as a *free* (circle) node and is restricted by a specialization edge to take its values from the type ADDRESS. Free nodes are used in IFO to indicate that the type of objects populating it is determined through ISA relationships by another part of the schema. In IFO, nodes such as HOME that occur as attribute ranges cannot be used in the same manner as atomic type nodes or the roots of constructed types. The specialization edge from HOME to ADDRESS enforces a form of referential constraint; this is related to but somewhat different from the subtype constraint represented by the ISA edge from TOURIST to PERSON. In IFO the leaves of constructed types are also represented as free types that are restricted using referential constraint ISA edges.

The use of fragments in IFO highlights some of the similarities between semantic models and frame-based approaches to knowledge representation such as KL-ONE [Brachman and Schmolze 1985]. A fragment in IFO corresponds loosely to a frame. Fragments provide a natural way of representing *nested* or context-dependent attributes. To illustrate, consider the set-valued attribute SPEAKS of the World Traveler schema. In IFO this could be augmented with a nested attribute WITH-PROFICIENCY, which would specify the proficiency. For example, if Mary spoke French and Chinese, this nested attribute might state that she speaks French with proficiency of 2 and Chinese with a proficiency of 3. Furthermore, the IFO model distinguishes between two kinds of ISA relationships, essentially as described in Section 2.2.3.

The *IRIS Model* [Derrett et al. 1985] was introduced recently, and a number of research projects using the model have been undertaken. The model is based primarily on object types, specialization, multivalued attributes, and some forms of derived schema components. An initial prototype version of IRIS [Derrett et al. 1985] will include a nonprocedural language for queries and specifying derived data, as well as schema definition capabilities. The use of the IRIS Model as the basis for software specification has been investigated [Lyngbaek and Kent 1986], and a theoretical investigation of the model has been initiated [Lyngbaek and Vianu 1987]. We focus on the support of derived data in the model and on the constraints used.

The basic building blocks of the structural portion of IRIS are readily described in terms of the GSM. IRIS supports both *literal* (printable) and *nonliteral* (abstract) object types. These types participate in a directed acyclic graph of ISA relationships that has the unique type OBJECT at its top. Objects may also be related through (typically multivalued) attributes, whose domains and ranges may be types or cross products of types. Aggregations are modeled using a single-valued attribute from a cross product of types into the Boolean type.

IRIS uses derived schema components to support a form of data relativism whereby the same data can be viewed structurally from more than one perspective. In particular, IRIS permits the derivation of several attributes from a single *base predicate*, that is, aggregation. Figure 17 illustrates this point with a simple example. The base predicate shown in Figure 17a specifies 4-tuples describing enrollments. We assume here that each COURSE is offered in several different LECTUREs, that STUDENTs take a given lecture of a course, and that students receive a GRADE. In Figure 17b, this information is viewed using the attribute COURSE-STUDENT-STATUS, which maps each course–student pair into the lecture the student is taking and the grade received. IRIS permits the specification of this attribute as being derived from the base predicate. An important underlying principle of IRIS is highlighted by this example: The attributes ENROLLMENT and COURSE-STUDENT-STATUS are viewed as independent of any particular underlying type. In general,

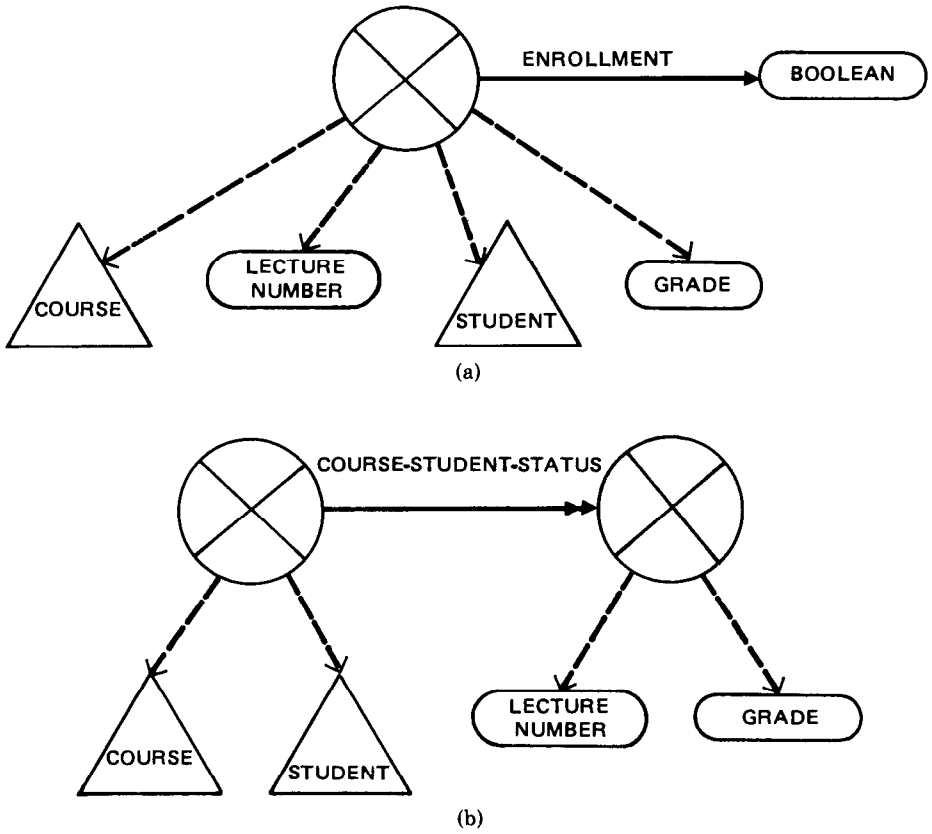


Figure 17. Two views of aggregation supported in IRIS.

several different attributes can be derived from a given base predicate.

IRIS supports a powerful kind of cardinality constraint called an *object participation* constraint. On the base predicate of Figure 17a, the constraint STUDENT[1, 5] would indicate that each student must be enrolled in at least one course–lecture pair and cannot be enrolled in more than 5. In Lyngbaek and Vianu [1987] these constraints are extended to attributes to address more than one coordinate at a time. For example, COURSE-STUDENT[0, 1] is used to state that each course–student pair can appear 0 or 1 time in the base predicate. This implies that the attribute of Figure 17b is single valued. An analysis of these constraints is presented in Lyngbaek and Vianu [1987]. Translation of IRIS schemas into relational schemas is also presented there.

3.3 Binary Models

In this section, we consider a representative of the family of "binary" models, which attempts to supply a small, universal set of constructs that are used to build more powerful structures. These models are thus minimalist in the sense that they require the database designer to understand fewer constructs.

The *Semantic Binary Data Model* (SBDM) [Abrial 1974] is representative of this family of models [Bracchi et al. 1976; Deheneffe et al. 1974; Hainaut and Lecharlier 1974; Senko 1975], all of which represent data using two constructs: entity sets and binary relations. As indicated in Figure 18, schemas of these models typically consist of labeled nodes for entity sets and labeled arcs corresponding to binary relationships between them. The primary

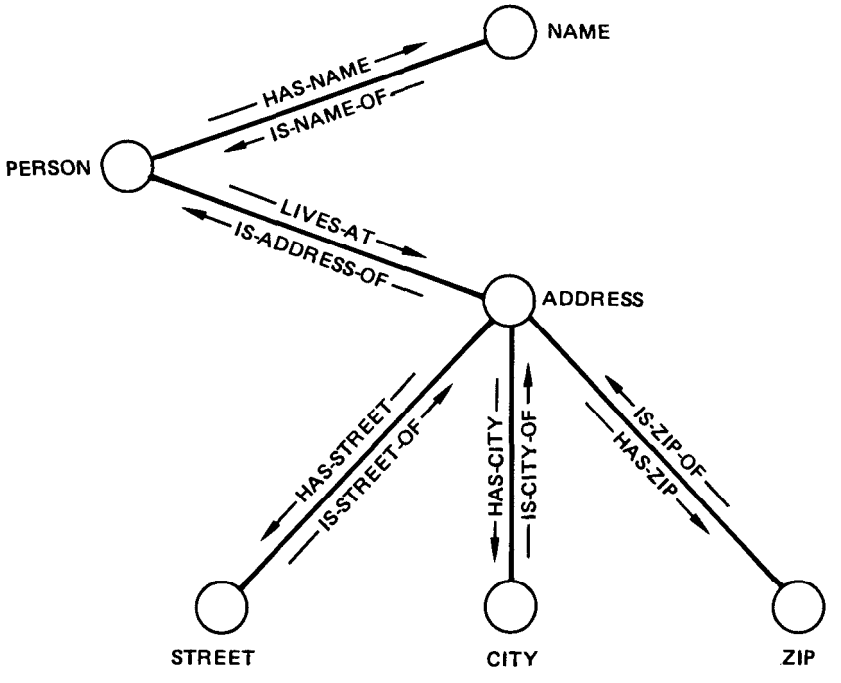


Figure 18. SBDM representation of part of World Traveler schema.

contribution of these models was to provide an early vehicle within which a number of fundamental types of data relationships could be articulated and studied.

In the SBDM, each binary relation is actually viewed as an inverse pair of possibly multivalued functions. For example, the binary relation connecting PERSON with ADDRESS in Figure 18 is viewed as consisting of the (single-valued) function LIVES-AT and the (multivalued) function IS-ADDRESS-OF. A data definition language and data manipulation language for the SBDM are defined in Abrial [1974]. The defined database transactions can be stored within the database; in this respect the SBDM follows INGRES [Stonebraker et al. 1976], in that the data dictionary access language is the data manipulation language applied to a certain part of the database. Recent work with these models includes the use of the SBDM as the basis for a formal schema design methodology [Dardailler et al. 1985], and the use of an extended binary model that incorporates ISA relationships and local constraints on relationships (e.g., 1:1 or N:1) [Rishe 1985, 1986].

As just described, the SBDM is closely related to the GSM in representational power. A major difference is that the type constructors of GSM must be simulated in SBDM. For example, the aggregation ADDRESS of the World Traveler schema is modeled as an abstract type with three binary relationships in the SBDM schema of Figure 18. As with FDM, a constraint must be added to ensure that each STREET, CITY, and ZIP tuple is unique. Also, the SBDM as described in Abrial [1974] does not support ISA relationships, although such relationships can essentially be represented within the framework.

3.4 Relational Extensions

The ER Model, FDM, SDM, and the other explicit models are similar in that they all take the approach of supplying a handful of distinct constructs that together are designed to serve the vast majority of modeling situations arising in typical database

application environments. They differ largely in the approach taken to interrelate data (type constructors versus attributes) and in the number of constructs supported. In this section we consider a very different approach taken by some researchers. We examine several models that support complex data as an extension to the well-known relational model. In these models the designer and user of a database view data from the relational perspective but are given mechanisms built out of the relational model to construct semantic schemas. A benefit of this approach is that these models may draw on the large body of knowledge known about the relational model, including query optimization, implementation strategies, and query language formulation. A potential drawback is that each of these models, like the relational model, imposes a level of indirection owing to the representation of objects and relationships based on records and identifier comparisons. By some users this might be viewed as tedius and inelegant.

The *Structural Model* [Wiederhold and El-Masri 1980] is a relatively simple extension of the relational model, which was introduced primarily as a tool for designing and integrating schemas for the record-oriented models. In this model, data are stored in relations, and five types of relation are distinguished. First, *primary entity* relations are used to store sets of tuples that closely correspond to classes of entities in the world. These relations store identifiers for these entities, along with single-valued attributes defined on them. In general, primary entity relations will not be affected by updates on other parts of the schema. *Referenced entity* relations, on the other hand, are used for entity sets that serve primarily as the range of attributes defined on the primary entity types. *Nest* relations are used for holding many-valued attributes, and *lexicon* relations are used to hold 1:1 correspondences between different names for the same object (e.g., person-name and Social Security number). Finally, *association* relations serve the same role as relationships in the ER Model and are used to model many:many relationships between primary entity types. Additional semantics are incorporated into the model by restricting how these different types of relations can reference each other (e.g., a referenced entity relation must be referenced by at least one other relation). In sum, the Structural Model uses relations to simulate an object-oriented approach that incorporates aggregation and single- and multivalued attributes in a fairly direct manner. On the other hand, ISA relationships are not incorporated as directly. An interesting application of the Structural Model is described in Brown and Parker [1983]. This paper introduces a graph-based representation of Structural Model schemas and describes a methodology for simplifying them.

RM/T [Codd 1979] is Codd's extension of the relational model. As in the Structural Model, various kinds of relations for representing different semantic modeling constructs are distinguished, and update semantics are specified for them. In RM/T, abstract objects are represented by *permanent surrogates*, and each type has an associated one-column *E-relation* that holds the surrogates of the objects currently populating that type. Although users can request that surrogates be created or deleted, they can never explicitly reference or view them. In this way, RM/T closely follows the object-oriented spirit of most semantic models. Single- and multivalued attributes are stored in relations using surrogates and printable values. Two forms of aggregation are supported: The so-called *associative entities* are aggregation objects that are assigned a new surrogate. The *nonentity associations* are aggregations for which no surrogates are assigned; these aggregations cannot have multivalued attributes, nor can they participate in ISA relationships. Grouping types based on attribute values (see Section 2.3.2) are called *cover aggregations* in RM/T. The model also provides explicit constructs for representing precedence relationships between entities that have time-valued attributes.

RM/T supports two types of ISA relationships. *Unconditional generalization* is

essentially the notion of ISA used in the GSM; each object in a subtype of an unconditional generalization must be a member of the supertype. *Alternative* (or conditional) *generalization* is used to form subsets of a union of types. For example, the type CUSTOMER might be defined as an alternative generalization of PERSON, BUSINESS, and PARTNERSHIP. In RM/T, this means that each customer must be either a person, a business, or a partnership, but that the set of customers does not have to contain all persons, businesses, and partnerships. Note that in some models, the type CUSTOMER could be modeled by first forming a supertype LEGAL-ENTITY of the three base types and then defining CUSTOMER as a subtype of that.

GEM [Tsur and Zaniolo 1984; Zaniolo 1983] is a relational extension that can also be viewed as an extension of the ER Model. In particular, this model supports entities and relationships, as well as subtyping and nonatomic attribute ranges. An unusual aspect of GEM is that it was developed as an experiment in supporting a semantic data language by extending a relational query language (see Section 3.5). This is in contradistinction to investigations that extend the relational model itself.

3.5 Access Languages

We conclude our survey of semantic models by examining various access languages that have appeared in the literature and that support semantic modeling constructs. We include these languages in this section (and not in Section 4, which discusses research directions of semantic modeling) because several of them were defined concurrently and independently of the various semantic models. Indeed, numerous researchers have taken the approach of viewing data modeling and data manipulation as an integrated mechanism. In this section we illustrate the general capabilities of semantic access languages using a language similar in form to DAPLEX [Shipman 1981] and Semdal [King 1984] and briefly survey the prominent languages in the literature. Figure 19 gives a brief survey of these languages. In this section we do not consider data manipulation for deeply nested constructed types; theoretical approaches to this problem are discussed in Section 4.4.

Three of the access languages, DAPLEX, GEM, and ARIEL, are extensions of the relational calculus [Date 1981; Ullman 1982] designed to encompass standard relational as well as semantic data structures. FQL is unique because it is based on the paradigm of functional programming (not to be confused with the Functional Data Model, which it supports). Unlike most database query languages, FQL does not support update specification or schema definition. Finally, TAXIS, DIAL, Semdal, and Galileo are imperative languages, with philosphical similarities to typical Pascal-like languages, including standard flow of control facilities and arithmetic capabilities.

DAPLEX supports the Functional Data Model. Like the other languages in this class, the query specification portion of this language contains syntactically elegant renditions of most of the basic elements of the first-order predicate calculus and is thus fundamentally nonprocedural. DAPLEX also supports the direct mention of attributes, their inverses, and their compositions and thus permits queries to have a somewhat navigational flavor. It also supports the specification of aggregate values such as averages, of orderings and similar properties of entity sets, and of database updates. It also supports the definition of schema components, including derived data.

FQL finds its roots in the work of Backus [1978] on functional programming languages. This approach offers several advantages. In particular, the functional approach reduces the use of secondary storage, simplifies the interface needed to provide outside programs with database access, and also typically enjoys an implementation that is quite compact [Buneman et al. 1982].

A query in FQL is formed by composing one or more functions, which may themselves be formed using transformations such as inverse or *, which turns a single-valued function into its analog that maps sets to sets. Another important operator is *restriction*, denoted using a |, which acts like a filter on a list of values. To illustrate

	REFERENCES	MODEL	PARADIGM	FUNCTIONALITY: QUERY SPECIFICATION	FUNCTIONALITY: SCHEMA DEFINITION	FUNCTIONALITY: SIMPLE UPDATE SPECIFICATION	FUNCTIONALITY: GENERAL PURPOSE PROGRAMMING LANGUAGE CONSTRUCTS	IMPLEMENTATION: DBMS INTERFACE	IMPLEMENTATION: SYSTEM EQUIP/OS/IMPL LANG
DAPLEX	[Shi81]	FDM	RELATIONAL CALCULUS EXTENSION	√	√	√	√	ADAPLEX EXTENSION	VAX/VMS/ADA [CDF82]
FQL	[BFN82]	FDM	FUNCTIONAL PROGRAMMING LANGUAGES	√				FUNCTIONAL DATA MODEL	VAX/VMS/PASCAL, CODASYL [Nik84]
TAXIS	[MBW80, MBG86]	TAXIS	IMPERATIVE WITH DERIVED SCHEMA COMPONENTS	√	√	√	√	RELATIONAL FRONT-END	VAX/UNIX/PASCALR [MBG86]
DIAL	[Bet80]	SDM	IMPERATIVE WITH DERIVED SCHEMA COMPONENTS	√	√	√	√	NOT IMPLEMENTED	
Semdal	[Kin84]	INSYDE SUBSET	IMPERATIVE WITH DERIVED SCHEMA COMPONENTS	√	√	√	√	DATA LANGUAGE FOR SEMBASE	SUN WORKSTATION, VAX, PYRAMID/UNIX/C
GEM	[Zan83, TsZ84]	GEM	RELATIONAL CALCULUS EXTENSION	√		√		INGRES FRONT-END	BRITTON LEE IBM 500 BACK-END TO VAX 780 / UNIX / RELATIONAL INTERFACE LANGUAGE
ARIEL	[Mac85]	TYPE/SUBTYPE/ ATTRIBUTE	RELATIONAL CALCULUS EXTENSION	√		√		RELATIONAL FRONT-END	SUN WORKSTATION VAX / UNIX / PASCAL
Galileo	[ACO85]	Galileo	IMPERATIVE WITH DERIVED SCHEMA COMPONENTS	√	√	√	√	GALILEO	VAX/UNIX/VAX MACHINE CODE [AOO85]

Figure 19. Data languages based on semantic models. Blanks indicate capability not supported to authors' knowledge. References in this figure are keyed to the reference list.

the approach, we present an FQL-like query that corresponds to the first query of Section 2.4. In this query a list of all linguists is formed using !LINGUIST. This list is filtered by an operator, which first forms ordered pairs with first coordinate the language count associated with a given linguist, and second coordinate the number 3, and then determines whether this pair stands in the ≥ relation. At this point, then, a list of linguists who speak at least three languages has been formed. This functional is now composed with *PNAME (which is the natural extension of the attribute PNAME to lists of persons):

!LINGUIST ∘ | ([LANG-COUNT,3] ∘ ≥)
∘ *PNAME

As discussed above, FQL does not support database updates or schema definition, and it is not clear how such capabilities could be incorporated into the language in a natural fashion.

TAXIS, Galileo, DIAL, and Semdal are designed to act as both query and application languages and are thus called *database programming languages*. The advantage of this approach is a cleaner integration of semantic database operations into application programs. The goal is to support the flexibility of general-purpose programming language constructs in constructing database transactions. In a database programming language, the user can write application programs that use semantic data types as program data types—and count on them to persist until the next use of the database, without having consciously to separate database archival tasks from general programming tasks. In this way, the user may extend the conceptual benefits of manipulating complex objects and ISA hierarchies into nondatabase manipulations. TAXIS and Galileo are discussed further in Section 4.2 below; an excellent survey of these and other programming languages can be found in Atkinson and Buneman [1987].

4. FROM IMPLEMENTATIONS TO THEORETICAL ANALYSIS

In this section we examine a number of research projects that have evolved around semantic modeling. These projects, instead of focusing on constructing new modeling techniques, have studied and applied semantic modeling to such areas as the implementation of physical access methods of semantic databases, the construction of mechanisms for the specification of database dynamics, the design of graphical interfaces to databases, and the theoretical analysis of database issues.

After semantic models and data languages based on semantic models had been in the research literature for a few years, some researchers began to wonder about the possibility of constructing full, efficient DBMSs based on these constructs. In Section 4.1 we discuss various implementation issues unique to semantic models and survey five implementations described in the literature.

In this article we have focused primarily on the static aspects of semantic models and discussed how they are used to structure and modularize database schemas. Some researchers have extended the area by developing mechanisms for structuring and modularizing the dynamic aspects of these models, with a primary focus on database transactions. In Section 4.2 we discuss two approaches researchers have taken in this area and survey the TAXIS, Galileo, SHM+, and INSYDE data models.

Section 4.3 examines what is perhaps the newest application of semantic modeling. A number of researchers have discovered that semantic models make useful mediums for providing interactive database interfaces. We examine several experimental graphics-based interfaces that use this approach.

Finally, Section 4.4 discusses theoretical investigations of concepts arising in semantic modeling. These include the development of formal query languages for constructed types, characterizations of update propagation, and other topics. This section also surveys the Format and Logical Data Models.

4.1 Systems

Two fundamental issues arise in the construction of full, efficient DBMSs based on the paradigm of semantic modeling. On the one hand, semantic models contain unusual constructs that force the physical designer

of a database to develop unique implementation constructs—abstract objects, subtype hierarchies, and derived data, in particular, introduce new issues in database physical design. On the other hand, like hierarchical and network databases, semantic schemas suggest expected access patterns to the designer. The designer of the physical implementation may capitalize on this information in selecting appropriate data structures.

In studying this area researchers have concentrated on two broad strategies: the use of existing data management capabilities (like relational systems and persistent programming languages) and the development of special-purpose, highly efficient access mechanisms. These efforts vary in the depth of their implementations and in the goals of the projects themselves. Figure 20 describes five sample systems in chronological order of their presentation in the literature. In the following these systems are described in some detail.

Three of the systems use existing tools to perform low-level data management: EFDM uses persistent ALGOL, GEM is built on top of Britton Lee's IDM500 relational database machine, and TAXIS is implemented in PASCALR. In both GEM and TAXIS each type is mapped to a relation. For greater efficiency, ADAPLEX and Sembase directly implement storage and access mechanisms to support semantic constructs. Objects are represented as variable-length records, with a field for each attribute connection. Each such field contains a reference to an object or a set of objects. These two implementations use conventional access methods, like B-trees, to manage sets of related objects (e.g., subtypes).

The five systems differ fundamentally in their intent as research projects. TAXIS is not intended to be a DBMS; rather, it is designed to support a programming language that encompasses data management facilities based on a semantic model. This language is described further in Section 4.2. ADAPLEX is an experiment in efficiently implementing a semantic database using general-purpose operating system files and in embedding a semantic database language (DAPLEX) in the programming language Ada. GEM is an attempt to support a semantic front end that is compatible with existing relational DBMSs and to assess the usefulness of relational database machines in supporting semantic models. The EFDM system concentrates on using a persistent language as a database implementation tool.

Sembase is an experiment in producing an integrated semantic language and graphical interface (see Section 4.3) and in providing efficient maintenance of derived data. A novel aspect of the Sembase implementation is the homogeneous treatment of records corresponding to objects of various types. All object records are stored in a heap on disk and accessed using a system of B-trees. In this way Sembase can adapt to predominantly used access patterns. Sembase also provides very efficient means of keeping derived subtypes and some forms of derived attributes up to date so that only a minimal amount of processing is required when a user requests a piece of derived information. To do this, Sembase maintains complex dependency information at the data level. For example, in the World Traveler Database, if a person speaks one language and learns another, the system will know that the given traveler must be reevaluated vis-à-vis the derivation rule for LINGUIST; the subtype as a whole will not be reevaluated. A recent extension of Sembase called CACTIS [Hudson and King 1986, 1987] supports the elegant and efficient maintenance of a much wider class of derived attributes.

An extension of ADAPLEX, called the Distributed Data Manager [Chan et al. 1983], supports a distributed semantic DBMS. This system uses the Functional Data Model and demonstrates that it is possible to implement a distributed semantic DBMS efficiently. Further, a semantic model is shown to be useful in creating more efficient distributed systems by making use of the extra semantics supplied in a schema when deciding how to distribute data. In particular, the Distributed Data Manager uses "fragment groups" as a means of localizing interobject references. Essentially, derived subtyping is used as a means of creating a small level of granularity to describe how different types of data

	REFERENCES	MODEL	IMPLEMENTATION STRATEGY	SYSTEM MACHINE/OS/IMPLEMENTATION LANG	ASSOCIATED SYSTEM COMPONENTS
TAXIS	[MBG86]	TAXIS	RELATIONAL FRONT-END, TYPE PER RELATION	VAX/UNIX/PASCALR	FULL PROGRAMMING LANGUAGE CAPABILITY [MBG86]
EFDM	[AtK83, Kul83]	FDM	PERSISTENT ALGOL FILE PER TYPE OR SUBTYPE	VAX/VMS/PERSISTENT ALGOL	QUERY LANGUAGE [Shi81]
GEM	[TsZ84]	GEM	RELATIONAL FRONT-END; TYPE PER RELATION	BRITTON LEE IDM 500 BACK-END TO VAX 780 / UNIX / RELATIONAL INTERFACE LANGUAGE	QUERY LANGUAGE [Zan83]
ADAPLEX	[CDF82]	FDM	VMS FILE PER TYPE OR SUBTYPE	VAX/VMS/ADA	EMBEDDED IN HOST PROGRAMMING LANGUAGE [SFL81] DISTRIBUTED IMPLEMENTATION [CDF83]
Sembase	[Kin84, FKM85]	INSYDE	EXTERNAL NON-UNIFORM RECORD PER OBJECT	SUN WORKSTATION VAX, PYRAMID / UNIX/C	QUERY LANGUAGE [Kin84] GRAPHIC INTERFACE [KiM84]

Figure 20. Systems based on semantic models. Blanks indicate capability not supported to authors' knowledge. Keys to references in this figure precede entries in reference list.

relate. Thus, for example, travelers may be grouped according to where they have been and then localized near the appropriate site information. In this way fragment groups are used as the basis of distribution and replication.

4.2 Dynamics

Until recently, most semantic modeling research concentrated on specifying the structural aspects of constructed types and the relationships between them, not on the behavioral components of semantic models. In Section 2 we discussed the dynamic components of semantic models for schema specification and data manipulation at a fundamental level. In this section we consider the broader issue of providing facilities for structuring database manipulation primitives into transactions.

The work on semantic transaction specification stems from early work on the design of programming languages with embedded database access mechanisms (e.g., [Rowe and Schoens 1979; Wasserman 1979]) and the work of Abrial [1974] on data semantics in the context of the SBDM. Two philosophies emerge in the four models discussed here. TAXIS and Galileo synthesize semantic modeling constructs with control and typing mechanisms from imperative programming languages. In contrast, SHM+ and INSYDE develop control mechanisms that closely follow the structure of semantic schemas.

TAXIS [Mylopoulous et al. 1980] is typically viewed as a programming language for data-intensive applications that incorporates several of the fundamental principles of data representation found in the semantic database literature. In particular, TAXIS is recognized as one of the first systems to merge semantic data modeling concepts, including attributes and ISA relationships, with more general programming language facilities such as abstract data types and exception handling. TAXIS provides tools for modularizing the specification of database transactions and can support a wide class of interactive data management applications.

To support the semantic data modeling concepts, TAXIS uses an extended form of the relational model (see Section 3). TAXIS incorporates most of the semantics associated with data in the specification of the operations associated with those data.

A more recently developed language, Galileo [Albano et al. 1985], is similar to TAXIS in that it incorporates semantic data model mechanisms within a strongly typed programming language. Galileo introduces a general family of type constructors for both database and nondatabase types and then uses them to support aggregation, attributes, and ISA relationships between database types. An interesting contribution of Galileo is the concept of *environments*, which are used as an abstraction mechanism in the support of modularization. Environments allow data operations to be incorporated into the database schema. Also, environments allow schemas to be built incrementally by giving the database user a mechanism for specifying, in a controlled fashion, exactly how new operators and schema components should interact with the existing schema.

The *Extended Semantic Hierarchy Model* (SHM+) [Brodie and Ridjanovic 1984] has highly structured static and dynamic components. The model for static data representation used in SHM+ is based on aggregation, grouping, and subtypes and represents attribute relationships using aggregation and keys. Static schemas in the model have a very hierarchical and modular flavor. In SHM+, behavioral components form an integral part of the structural building blocks. For example, a primitive for iterative application of an operation is directly associated with the grouping construct. As a result, it is very easy to specify involved database updates as highly structured transactions [Brodie and Ridjanovic 1984]. A comprehensive methodology for incremental design of the static and dynamic components of schemas is developed in Brodie and Ridjanovic [1984].

The *INSYDE Model* [King and McLeod 1985a], which is derived from the Event Model [King and McLeod 1982, 1984] is a semantic database model that combines static as well as dynamic primitives at a fundamental level. The static portion of the model is essentially a rich subset of SDM, and the dynamic component includes

language constructs to support data manipulation as well as schema design and evolution. The INSYDE Model provides a unified support system for the entire life cycle of a database, providing tools and abstraction mechanisms for the design of the static schema, the design of database transactions, and the support of schema evolution within an integrated framework. In particular, it includes a prescriptive, stepwise methodology for logical database design, the purpose of which is to guide a designer through both the specification and maintenance of a semantically expressive schema. This methodology gives a primary role to transactions: The expected transactions are a fundamental part of the initial functional specification of a system and are used to drive much of the refinement and implementation process. Use of the INSYDE Model is particularly appropriate in the design of Office Information Systems [King and McLeod 1985a].

4.3 Graphical Interfaces

Several graphical interfaces to databases based on semantic models have been developed. This is due largely to the fact that semantic models are conducive to visual representations. In fact, as we have seen, the data definition languages of some semantic models are already graphical in nature. The development of these interfaces has also been encouraged by the advent of workstation technology, which provides relatively inexpensive bit-mapped displays and pointing devices (such as mice). Semantic models are an obvious choice for capitalizing on the ability of bit-mapped workstations to display two-dimensional images. In this section we examine a few noteworthy attempts at using a semantic model as the basis of an interactive database interface.

Figure 21 surveys six experimental database interfaces based on semantic models. In Figure 21 we construct a taxonomy of these systems along three central axes: functionality, implementation environment, and the extent to which graphics are used. With each system, we also list the data model the interface supports. The six systems are listed chronologically, in terms of their presentation in the literature. We see that they tend to support more and more complex data models. GUIDE and LID support the ER Model; DDEW is based on an ER Model extended to provide subtyping. SKI and ISIS support models that are functionally very similar to restricted subsets of SDM. SNAP supports a subset of the IFO model.

As indicated by the functionality area of the chart, all of the systems except LID concentrate on schema management, providing capabilities for schema definition, schema browsing, and query formulation. The ISIS and SNAP systems also provide some limited facilities for convenient representation and perusal of printed data. And, in contrast to the other five systems, LID is oriented entirely toward data browsing and allows the user, given a particular object in the database, to traverse schema relationships to find related objects. In LID there is no capability of viewing sets of objects, only individual ones. Therefore, if the user wishes, say, to view all travelers who have been to Japan, the locations visited by each traveler would have to be examined individually. The system will not perform the search and return the result as one conceptual entity.

With respect to the implementations of these six systems, Figure 21 indicates that all but DDEW are experimental research prototypes and DDEW and SKI are the only ones that interface with actual, disk-based databases. DDEW may be used to specify and browse schemas; SKI may also be used to formulate a query on an existing semantic database. Clearly, however, the lack of actual DBMSs underneath these systems does not detract from the significance of their research contributions. Any issues concerning the speed of disk accesses while examining data are largely irrelevant. Finally, we note that these systems indicate a historical trend toward workstation environments.

The final axis in the chart refers to the use of graphics in the six example systems and indicates a progression from fairly simple graph- (or network-) based representations of schemas to richer visual (but still

	REFERENCES	DATA MODEL	FUNCTIONALITY: SCHEMA BROWSING	FUNCTIONALITY: SCHEMA DEFINITION	FUNCTIONALITY: QUERY SPECIFICATION	FUNCTIONALITY: DATA BROWSING	IMPLEMENTATION: GRAPHICS INTERFACE	IMPLEMENTATION: INTERFACE TO DBMS	IMPLEMENTATION: SYSTEM EQUIP/OS/IMPL LANG	USE OF GRAPHICS: VISUAL REP OF SCHEMA	USE OF GRAPHICS: PARADIGM OF SCHEMA DEFINITION	USE OF GRAPHICS: PARADIGM OF SCHEMA VIEWING	USE OF GRAPHICS: QUERY SPECIFICATION
GUIDE	[WoK82]	ER	√		√		PROTOTYPE		VAX/VMS/C	FIXED ER DIAGRAM	NA	DIRECT MANIPULATION OF SCHEMA	DIRECT MANIPULATION OF SCHEMA
LID	[Fog84]	ER				GRAPHICAL	PROTOTYPE	IN MEMORY	?	ONLY LOCAL STRUCTURE	NA	NA	TRIPLE-LEVEL NAVIGATION
DDEW	[RBB84]	ER+	√	√			PRODUCTION-QUALITY	GENERATES RELATIONAL, NETWORK, HIERARCHICAL SCHEMAS	JUPITER 12/UNIX/C	ER+ DIAGRAMS	TEXT-BASED	DIRECT MANIPULATION OF SCHEMA	NA
SKI	[KiM84]	INSYDE SUBSET	√	√	√		PROTOTYPE	SEMBASE DBMS	SUN WORKSTATION/ UNIX/C	FORMATTED TYPE/ SUBTYPE/ ATTRIBUTE DIAGRAMS	GRAPHICAL	SEMANTICALLY-DRIVEN OPERATORS	SPECIFICATION OF DERIVED DATA
ISIS	[GGK85]	SDM SUBSET	√	√	√	TEXT-BASED	PROTOTYPE	IN MEMORY	APOLLO/UNIX/C	TYPE/ SUBTYPE/ ATTRIBUTE DIAGRAMS	GRAPHICAL	MODE-ORIENTED	SPECIFICATION OF DERIVED DATA
SNAP	[BrH86]	IFO SUBSET	√	√	√		PROTOTYPE		SYMBOLICS 3600/ ZETA LISP	IFO DIAGRAMS	GRAPHICAL	DIRECT MANIPULATION OF SCHEMA	DIRECT MANIPULATION OF SCHEMA

Figure 21. Graphics-based interfaces based on semantic models. Blanks indicate capability not supported to authors' knowledge. Key to references in this figure precedes entries in reference list.

graphlike) representations. Correspondingly, there has been a development in terms of the naturalness with which these systems allow the user to interact with the schema while browsing and specifying queries. In Foley and Dam [1982] the authors differentiate three sorts of feedback that an interactive system can give: *lexical* (such as the echoing of typed characters), *syntactic* (such as highlighting a selected menu item), and *semantic*. Semantic feedback is the most sophisticated sort of feedback and might, for example, indicate in an obvious fashion that a user-requested operation has been performed (e.g., that a subtype has been added to the schema). In general, the six selected interactive systems provide good semantic feedback, with the visual representation of schemas being dynamically modified as they are manipulated. The trend has been toward richer semantic feedback.

All six systems represent the schema using graph structures. GUIDE and DDEW do this essentially using ER diagrams. In GUIDE the schema is statically structured and is not altered by the user. In DDEW the system suggests an appropriate schema representation, but the user may alter it. In ISIS and SNAP the user directly defines the visual representation of the schema. SKI completely controls the visual structure of the schema for the user. (Of course, LID, since it only supports the browsing of individual data items, does not provide visual access to the entire schema.) All these systems (except LID) allow users to pan and zoom, and hide irrelevant portions of the schema. SKI, ISIS, and SNAP are now discussed in more detail.

In the SKI system there is no notion of a statically defined graphical layout of the underlying schema. Instead, a graphical representation of the relevant subschema is created dynamically as the user specifies that various connections be displayed. The subschema is placed on a formated screen; the screen is broken into stripes, each containing components with different semantic significance (e.g., one for object types, another for attributes defined on these types). The philosophy behind this is to remove from the user the burden of having to manage a large, complex graph that represents a schema and to automatically format subschemas according to their content. Of course, this approach does not allow the user to return easily to familiar visual representations during a lengthy session.

ISIS and SNAP both take the approach of providing more permanent representations of schemas. In both systems the user creates much of the original visual representation of a schema; this is saved by the system and provides the basis for the display of both the schema and portions of it. Both systems also permit users to modify the representation of schemas.

With respect to the manner in which the user peruses the schema while browsing, all of the systems (except for LID) provide mechanisms for massaging the visual representation directly in order to focus on specific areas of interest within the schema. The user follows semantic connections (e.g., attributes and type/subtype relationships) in order to isolate various sorts of data that relate to each other. This is much more effective than forcing the user to peruse the schema only by navigating up and down, and right and left over a schema that is too large to fit on a screen. Thus, the user may hide levels of detail and/or annotate the schema while browsing. Schneiderman [1980] has coined the phrase *direct manipulation* to characterize this style of interaction, where the user has the feeling of manipulating a real-world object while interfacing with the system. This provides very rich and effective semantic feedback for the user. These systems vary somewhat in their approach to direct manipulation during browsing. ISIS supports operators for maneuvering between its diagram for representing ISA relationships and its diagrams for representing attribute relationships. Also, SKI provides several high-level operators that may be used to make very dramatic shifts in focus, such as requesting to view all areas of the schema that would be affected if a particular data operation was performed.

The six systems also vary in how they support query specification, with a growing tendency toward more graphics-based methods of specifying an entire query.

GUIDE supports selection-like queries; to specify them, the user graphically identifies relevant portions of the schema, and a text-based query is constructed in parallel by the system. SKI was the first system to embody the concept of expressing queries through the iterative specification of derived data, thus making a query essentially an extension of the schema. The user may traverse the schema to find the general type of data of interest and then repeatedly refine this type by specifying subtype predicates. These predicates are defined textually. SKI provides a series of pop-up menus that support an editor that displays the syntax permitted in subtype definitions. ISIS uses a similar paradigm but provides a much more sophisticated editor that allows the user to specify a query in an almost entirely mouse-based fashion. SNAP uses a different approach in which copies of schema components are directly manipulated to form queries. Intuitively, SNAP queries can be viewed as a generalization of Zloof's Query-by-Example [Zloof 1987] to a graphics-based model.

A commercial implementation of a graphics interface based on a semantic model has recently been developed. It is similar in structure and function to SKI, ISIS, and SNAP and includes a schema design tool and a data browser [Rogers and Cattell 1987].

4.4 Theory

Over recent years, there has been increasing interest in the application of theoretical techniques to investigating concepts and problems raised by semantic database models. Considerable work has been performed on constructed types considered in isolation, with a primary focus on data access and manipulation languages for them. Another important topic considers ISA networks in isolation and studies the inference of various properties in them. Other important topics include comparing the data capacity of constructed types and characterizing update propagation in semantic schemas. These topics have been studied using a variety of different models; no one model has emerged as the common basis for theoretical investigations of these issues. In this section we briefly survey the work in these various areas. A more comprehensive survey of work on theoretical research on constructed types may be found in Hull [1987]. Also, we note that some of the theoretical work on the IFO [Abiteboul and Hull 1987] and IRIS [Lyngbaek and Vianu 1987] models is discussed in Section 3.2.

Surprisingly, a significant amount of the research on access languages for constructed types has been performed using an offshoot of the relational model called non-first-normal-form relations [Abiteboul and Bidoit 1986; Fischer and Thomas 1983; Jaeschke and Schek 1982; Makinouchi 1977]. These are relations where some columns may hold relations instead of atomic values; nesting of relations in this way is permitted to arbitrary depths. The structure of such a non-first-normal-form relation may be viewed as a constructed type formed using alternating layers of aggregation and grouping. Calculus-based and algebraic languages for non-first-normal-form relations have been developed [Fischer and Thomas 1983; Jaeschke and Schek 1982] and shown to have equivalent expressive power [Abiteboul and Beeri 1987; Kuper and Vardi 1984].

Another model of constructed types is the Format Model [Hull and Yap 1984]. Formats are built hierarchically from three constructs: aggregation, grouping, and marked or disjoint union. The third construct is used to model types that result from generalizations involving two or more disparate types. Original work on the Format Model focused on comparing the data capacity of formats [Hull and Yap 1984; O'Dunlaing and Yap 1982]. A more recent paper [Abiteboul and Hull 1986] proposes a query language for formats that is based on a rewrite operator.

The *Logical Data Model* (LDM) [Kuper and Vardi 1984, 1985; Kuper 1985] can be viewed as a further extension of constructed types. LDM was introduced primarily to provide a logic-based model lying between the record-oriented models (relational, network, hierarchical) and actual physical implementation, but issues studied

with the LDM are also important from the point of view of semantic models.

As with the Format Model, LDM schemas are built from basic types and type constructors for aggregation, grouping, and marked union. (In Kuper and Vardi [1984, 1985] marked union is not included.) Unlike the constructed types of GSM and the Format Model, however, schemas of the LDM are directed graphs rather than trees. Instances of LDM schemas are defined using the complementary notions of *l-* and *r-values* from the theory underlying programming language assignment statements. This formalism captures the object-oriented nature of semantic models in a novel manner: *l*-values correspond to object identities, while *r*-values correspond to their values. In Kuper and Vardi [1984, 1985] calculus-based and algebraic manipulation languages are introduced for LDM and shown to have equivalent expressive power. In Kuper and Vardi [1985], the relative data capacity of LDM schemas is studied.

We now turn to research focused primarily on ISA relationships. Two recent papers study the interplay of ISA relationships in connection with integrity constraints [Atzeni and Parker 1986; Lenzerini 1987]. Both of these works use a simple abstract semantic model based on abstract types and ISA relationships. In Atzeni and Parker [1986] the interaction of ISA relationships and disjointness constraints is studied, and a sound and complete set of inference rules for these properties is presented. Also, it is shown that various problems are decidable in polynomial time. One such problem is the *satisfiability* problem, which concerns whether a node in a schema is nonempty in at least one instance. In Lenzerini [1987], covering constraints are studied in addition to disjointness constraints. In this case many problems are NP-complete, including the satisfiability problem.

We now turn to the problem of determining how atomic updates propagate in semantic schemas. This is of particular interest because it considers the various constructs of semantic models taken together rather than in isolation. As a simple example of update propagation, suppose that in the World Traveler Database Pam is both a person and a tourist. If Pam is deleted from PERSON, she should also be deleted from TOURIST. Update propagation relative to the basic structural components of semantic models has been studied in the context of both FDM [Hecht and Kerschberg 1981] and IFO [Abiteboul and Hull 1985, 1987]. In both papers the semantics of update propagation is broken into two logical pieces: one concerned with the impact of updates on the local constructs of a schema and the other with the global impact implied by their combination. The overall impact of an update at a given node is essentially defined to be the sum of the impacts implied by the local update semantics. In both models, acyclicity conditions on ISA relationships ensure that each node need be visited at most once during this computation.

5. CONCLUDING REMARKS

In this paper we have surveyed a wide area of research, all of it centered around semantic database modeling. We have taken an in-depth look at the fundamental motivations and aspects of semantic models and examined a number of specific models. Further, several research directions that are based on semantic models have been discussed, including semantic data access languages, graphical database interfaces based on semantic models, physical implementations of semantic DBMSs, and theoretical investigations of semantic models.

Clearly, there are many more research issues relating to semantic models that could be investigated, such as the integration of temporal reasoning into semantic models, the optimization of semantic database queries, the development of semantic database machines, and the construction of expert database systems that use semantic models (such databases would be capable of making inferences about complex, semantic data). However, although some research is currently being conducted in these areas, it has not reached the level of maturity appropriate for a survey paper.

We would, however, like to conclude this paper by mentioning a rapidly growing area of database research that is related to semantic modeling. Recently, a number of

research projects have focused on the development of data models that are more expressive than conventional models but use techniques different from those of semantic models. Experimental systems based on these *object-oriented* models are typically centered around the concepts of large objects and extensible type structures, such as arise in engineering design applications, and the concept of local behavior stemming from object-oriented programming languages.

Object-oriented database models are fundamentally different from semantic models in that they support forms of local behavior in a manner similar to object-oriented programming languages. This means that a database entity may locally encapsulate a complex procedure or function for specifying the calculation of a data operation. This gives the database user the capability of expressing, in an elegant fashion, a wider class of derived information than can be expressed in semantic models. For example, in the system Postgres [Stonebraker and Rowe 1986] a data item may have an attribute that is a database query or an application program. This project is an experiment in extending already-developed relational techniques to the handling of complex data. In the system Cactis [Hudson and King 1986], an object may have attribute values that are computed by arbitrary computable functions. This project focuses on the design of formalisms that may be used to implement complex derived data efficiently and uses attributed graphs as an underlying physical construct. The Gemstone [Maier et al. 1986] project focuses on providing database users with message-passing mechanisms similar to those of Smalltalk. Thus, generalized methods may be defined for specifying how an object should react to messages from another object.

On another dimension, object-oriented models are similar to semantic models in that they provide mechanisms for constructing complex data by interrelating objects. The Exodus [Carey et al. 1986] system attempts to support the storage and access of very large objects. The Exodus project also provides the capability of allowing the user to easily extend the type structure. Another extensible system, designed for such applications as engineering, is Probe [Manola and Dayal 1986]. The GENESIS project [Batory et al. 1988] focuses on the rapid development of customized database management systems.

Speaking broadly, semantic modeling has concentrated largely on building complex data via mechanisms like attributes, aggregation, and generalization, which are widely viewed to be adequate for most business and commercial applications. In contrast, object-oriented models are oriented toward novel applications that must support complex domains such as software design [Hudson and King 1987], VLSI and printed circuit board design, and CAD/CAM [Andrews and Harris 1987; Su et al. 1988]. These applications are generally interactive and require highly dynamic database systems where the user may control local behavior and dynamically modify the type structure. Software specifications, text, and engineering designs are also much larger objects than typical business objects. Other novel research issues also arise in the context of object-oriented databases, including very long transactions (to support interactive design), nested transactions (to support complex design functions), and mechanisms for obtaining multiple blocks of data from a mass storage device quickly (to allow the efficient retrieval of large objects). Research on these and related topics will be crucial in expanding the usefulness of database systems to nontraditional, nonbusiness applications.

ACKNOWLEDGMENTS

We would like to acknowledge the anonymous referees and Salvatore March for their very helpful comments concerning previous versions of this survey. Richard Hull was supported in part by the National Science Foundation under grants IST-83-06517 and IST-85-11541. Roger King was supported in part by the Office of Naval Research under contract number N00014-86-K-0054 and by the National Science Foundation under grant DMC-8505164.

REFERENCES

[AbB87] ABITEBOUL, S., AND BEERI, C. 1987. On the power of languages for the manipulation of complex objects (extended abstract). In *Proceedings of International Workshop on Nested Relations and Complex Objects*. (Darmstadt, Germany, Apr.). INRIA, Roquencourt, France.

[AbB84] ABITEBOUL, S., AND BIDOIT, N. 1986. Nonfirst normal form relations: An algebra allowing data restructuring. *J. Comput. Syst. Sci. 33*, 361–393.

[AbH85] ABITEBOUL, S., AND HULL, R. 1985. Update propagation in the IFO database model. In *Proceedings of the International Conference on Foundations of Data Organization*. Organizing Committee of the International Conference on Foundations of Data Organization, c/o S. P. Ghosh, IBM Research, Almaden, Calif., pp. 243–251.

[AbH86] ABITEBOUL, S. AND HULL, R. 1986. Restructuring of complex database objects and office forms. In *Proceedings of the International Conference on Database Theory* (Rome, Sept.).

[AbH87] ABITEBOUL, S., AND HULL, R. 1987. IFO: A formal semantic database model. *ACM Trans. Database Syst. 12*, 4 (Dec.), 525–565.

[Abr74] ABRIAL, J. R. 1974. Data semantics. *Data Base Management*. North-Holland, Amsterdam, pp. 1–59.

[AfM84] AFSARMANESH, H., AND MCLEOD, D. 1984. A framework for semantic database models. In *Proceedings of the NYU Symposium on New Directions for Database Systems* (New York, May 16–18). New York Univ., New York.

[Aik85] AIKENS, J. 1985. A representation scheme using both frames and rules. In *Rule-Based Expert Systems*, B. Buchanan and E. Shortliffe, Eds. Addison-Wesley, Reading, Mass., pp. 424–440.

[ACO85] ALBANO, A., CARDELLI, L., AND ORSINI, R. 1985. Galileo: A strongly-typed, interactive conceptual language. *ACM Trans. Database Syst. 10*, 2 (June), 230–260.

[AOO85] ALBANO, A., OCCHIUTO, M. E., AND ORSINI, R. 1985. Galileo Reference Manual, VAX/UNIX Version 1.0. Tech. Rep. Dipartimento di Informatica, Univ. di Pisa, Pisa, Italy.

[AnH87] ANDREWS, T., AND HARRIS, C. 1987. Combining language and database advances in an object-oriented development environment. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications* (Oct.), pp. 430–440.

[AtB87] ATKINSON, M. P., AND BUNEMAN, O. P. 1987. Database programming languages, *ACM Comput. Surv. 19*, 2 (June), 105–190.

[AtK83] ATKINSON, M. P., AND KULKARNI, K. G. 1983. Experimenting with the functional data model. Tech. Rep. Persistent Programming Research Rep. 5, Univ. of Edinburgh, Edinburgh, Scotland.

[AtP86] ATZENI, P., AND PARKER, D. S. 1986. Formal properties of net-based knowledge representation schemes. In *Proceedings of the 2nd IEEE International Conference on Data Engineering*. IEEE, New York, pp. 700–706.

[Bac78] BACKUS, J. 1978. Can programming be liberated from the Von Neumann style? A functional style and its algebra of programs. *Commun. ACM 21*, 8 (Aug.), 613–641.

[BLN86] BATINI, C., LENZERINI, M., AND NAVATHE, S. B. 1986. A comparative analysis of methodologies for Database schema integration. *ACM Comput. Surv. 18*, 4 (Dec.), 323–364.

[BaK85] BATORY, D. S., AND KIM, W. 1985. Modeling concepts for VLSI CAD objects. *ACM Trans. Database Syst. 10*, 3 (Sept.), 322–346.

[BBG] BATORY, D. S., BARNETT, J. R., GARZA, F. F., SMITH, K. P., TSUKAUDA, K., TWICHELL, B. C., AND WISE, T. E. 1988. GENESIS: A reconfigurable database management system. *IEEE Trans. Softw. Eng.* to appear.

[BKK86] BOBROW, D., KAHN, K., KICZALES, G., MASINTER, L., STEFIK, M., AND ZDYBEL, F. 1986. CommonLoops: Merging Lisp and object-oriented programming. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications* (Mar.). ACM, New York, pp. 17–29.

[Bor85] BORGIDA, A. 1985. Features of languages for the development of information systems at the conceptual level. *IEEE Software 2*, 1 (Jan.), 63–72.

[BPP76] BRACCHI, G., PAOLINI, P., AND PELAGATTI, G. 1976. Binary logical associations in data modelling. In *Modelling in Data Base Management Systems*. North Holland, Amsterdam, pp. 125–148.

[BrS85] BRACHMAN, R. J., AND SCHMOLZE, J. G. 1985. An overview of the KL-ONE knowledge representation system. *Cognitive Sci. 9* (1985), 171–216.

[Bro84] BRODIE, M. L. 1984. On the development of data models. In *On Conceptual Modelling*, M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, Eds. Springer-Verlag, New York, pp. 19–48.

[BrR84] BRODIE, M. L., AND RIDJANOVIC, D. 1984. On the design and specification of database transactions. In *On Conceptual Modelling*. Springer-Verlag, New York, pp. 277–306.

[BMS84] BRODIE, M. L., MYLOPOULOS, J., AND SCHMIDT, J. W. Eds. 1984. *On Conceptual Modelling*. Springer-Verlag, New York.

[BrP83] BROWN, R., AND PARKER, D. S. 1983. LAURA: A formal data model and her logical design methodology. In *Proceedings of the 9th International Conference on Very Large Data Bases*. Very Large Database Endowment, Saratoga, Calif., pp. 206–218.

[BFN82] BUNEMAN, P., FRANKEL, R. E., AND NIKHIL, R. 1982. An implementation technique for database query languages. *ACM Trans. Database Syst. 7*, 2 (June), 164–186.

[CDR86] CAREY, M. J., DEWITT, D. J., RICHARDSON, J. E., AND SHEKITA, E. J. 1986. Object and file

management in the EXODUS extensible database system. In *Proceedings of the 12th International Conference on Very Large Databases* (Aug.). Very Large Database Endowment, Saratoga, Calif., pp. 91–100.

[CGT75] CHAMBERLIN, D. D., GRAY, J. N., AND TRAIGER, I. L. 1975. Views, authorization and locking in a relational database system. In *Proceedings of AFIPS National Computer Conference*, vol. 44. AFIPS Press, Reston, VA., pp. 425–430.

[CDF83] CHAN, A., DAYAL, U., FOX, S., AND RIES, D. 1983. Supporting a semantic data model in a distributed database system. In *Proceedings of the 9th International Conference on Very Large Data Bases*. Very Large Database Endowment, Saratoga, Calif., pp. 354–363.

[CDF82] CHAN, A., DANBERG, S., FOX, S. LIN, W. T. K., NORI, A., AND RIES, D. 1982. Storage and access structures to support a semantic data model. In *Proceedings of the 8th International Conference on Very Large Data Bases*. Very Large Database Endowment, Saratoga, Calif., pp. 122–130.

[Che76] CHEN, P. P. 1976. The entity-relationship model–Toward a unified view of data. *ACM Trans. Database Syst. 1*, 1 (Mar.), 9–36.

[Cod79] CODD, E. F. 1979. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst. 4*, 4 (Dec.), 397–434.

[CoP84] COSMADAKIS, S. S., AND PAPADIMITRIOU, C. H. 1984. Updates of relational views. *J. ACM 31*, 4 (Oct.), 743–760.

[DDG85] DARDAILLER, P., DELOBEL, C., AND GIRAUDIN, J. P. 1985. Modelisation progressive d'une base de donnees. Tech. Rep. 493, Laboratoire de Genie Informatique.

[Dat81] DATE, C. J. 1981. *An Introduction to Database Systems*, vol 1. Addison-Wesley, Reading, Mass.

[DaH84] DAYAL, U., AND HWANG, H. Y. 1984. View definition and generalization for database integration in a multidatabase system. *IEEE Trans. Softw. Eng. SE-10*, 6, 628–644.

[DHP74] DEHENEFFE, C., HENNEBERT, H., AND PAULUS, W. 1974. Relational model for a data base. In *Proceedings of the IFIP Congress*, pp. 1022–1025.

[DKL85] DERRETT, N., KENT, W., AND LYNGBAEK, P. 1985. Some aspects of operations in an object-oriented database. *IEEE Database Eng. Bull. 8*, 4 (Dec.).

[Fag77] FAGIN, R. 1977. Multivalued dependencies and a new normal form for relation databases. *ACM Trans. Database Syst. 2*, 3 (Sept.), 262–278.

[FKM85] FARMER, D. B., KING, R., AND MYERS, D. A. 1985. The semantic database constructor. *IEEE Tran Softw. Eng. SE-11*, 7, 583–591.

[FiK85] FIKES, R., AND KEHLER, T. 1985. The role of frame-based representation in reasoning. *Commun. ACM 28*, 9 (Sept.), 904–920.

[Fin79] FINDLER, N., Ed. 1979. *Associative Networks*. Academic Press, New York.

[FiT83] FISCHER, P., AND THOMAS, S. 1982. Operators for non-first-normal-form relations. In *Proceedings of the 7th International Computer Software Applications Conference* (Chicago, Nov.). IEEE, New York, pp. 464–475.

[Fog84] FOGG, D. 1984. Lesson from a "Living in a database" graphical query interface. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data* (Boston, Mass.). ACM, New York, pp. 100–106.

[FoD82] FOLEY, J. D., AND VAN DAM, A. 1982. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, Mass.

[GoR83] GOLDBERG, A., AND ROBSON, D. 1983. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, Reading, Mass.

[GGK85] GOLDMAN, K. J., GOLDMAN, S. A., KANELLAKIS, P. C., AND ZDONIK, S. B. 1985. ISIS: Interface for a semantic information system. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. ACM, New York, pp. 328–342.

[Gut77] GUTTAG, J. 1977. Abstract data types and the development of data structures. *Commun. ACM 20*, 6 (June), 396–404.

[HaL74] HAINAUT, J. L., AND LECHARLIER, B. 1974. An extensible semantic model of database and its data language. In *Proceedings of the IFIP Congress*, pp. 1026–1030.

[HaB80] HAMMER, M., AND BERKOWITZ, B. 1980. DIAL: A programming language for data intensive applications. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. ACM, New York, pp. 75–92.

[HaM81] HAMMER, M., AND MCLEOD, D. 1981. Database description with SDM: A semantic database model. *ACM Trans. Database Syst. 6*, 3 (Sept.), 351–386.

[HeK81] HECHT, M. S., AND KERSCHBERG, L. 1981. Update semantics for the functional data model. Tech. Rep., Bell Laboratories, Holmdel, N.J.

[HuK86] HUDSON, S. E., AND KING, R. 1986. CACTIS: A database system for specifying functionally-defined data. In *Proceedings of the Workshop on Object-Oriented Databases* (Asilomar, Pacific Grove, Calif., Sept.). IEEE, New York.

[HuK87] HUDSON, S. E., AND KING, R. 1987. Object-oriented database support for software environments. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (San Francisco, Calif., May). ACM, New York, pp. 491–503.

[Hul87] HULL, R. 1987. A survey of theoretical research on typed complex database objects. In *Databases*, J. Paredaens, Ed. Academic Press, London.

[HuY84] HULL, R., AND YAP, C. K. 1984. The format model: A theory of database organization. *J. ACM 31*, 3 (July), 518–537.

[IsB84] ISRAEL, D. J., AND BRACHMAN, R. J. 1984. Some remarks on the semantics of representation languages. In *On Conceptual Modelling*. Springer-Verlag, New York, pp. 119–146.

[JaS82] JAESCHKE, B., AND SCHEK, H. J. 1982. Remarks on the algebra of non first normal form relations. In *Proceedings of the ACM Symposium on Principles of Database Systems*. ACM, New York.

[KeC83] KEHLER, T. P., AND CLEMENSON, G. D. 1983. An application development system for expert systems. *Syst. Softw. 3*, 1 (Jan.), 212–223.

[Ken78] KENT, W. 1978. *Data and Reality*. North-Holland, Amsterdam.

[Ken79] KENT, W. 1979. Limitations of record-based information models. *ACM Trans. Database Syst. 4*, 1 (Mar.), 107–131.

[KeP76] KERSCHBERG, L., AND PACHECO, J. E. S. 1976. A functional data base model. Tech. Rep., Pontificia Univ. Catolica do Rio de Janeiro, Rio de Janeiro, Brazil.

[KKT76] KERSCHBERG, L., KLUG, A., AND TSICHRITZIS, D. 1976. A taxonomy of data models. In *Systems for Large Data Bases*. North-Holland, Amsterdam, pp. 43–64.

[KhC86] KHOSHAFIAN, S. N., AND COPELAND, G. P. 1986. Object identity. In *Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications* (Portland, Oreg., Sept.). ACM, New York, pp. 406–416.

[Kin84] KING, R. 1984. Sembase: A semantic DBMS. In *Proceedings of the First International Workshop on Expert Database Systems* (Oct.). Univ. of Southern Carolina, Columbia, S.C., pp. 151–171.

[KiM82] KING, R., AND MCLEOD, D. 1982. The event database specification model. In *Proceedings of the 2nd International Conference on Databases: Improving Usability and Responsiveness* (Jerusalem, Israel), pp. 299–321.

[KiM84] KING, R., AND MCLEOD, D. 1984. A unified model and methodology for conceptual database design. In *On Conceptual Modelling*. Springer-Verlag, New York, pp. 313–331.

[KiM85a] KING, R., AND MCLEOD, D. 1985a. A database design methodology and tool for information systems. *ACM Trans. Off. Inf. Syst. 3*, 1 (Jan.), 2–21.

[KiM85b] KING, R., AND MCLEOD, D. 1985b. Semantic database models. In *Database Design*. Springer-Verlag, New York, pp. 115–150.

[Kup85] KUPER, G. M. 1985. The logical data model: A new approach to database logic. Ph.D. dissertation, Computer Science Dept., Stanford Univ.

[KuV84] KUPER, G. M., AND VARDI, M. Y. 1984. The logical data model. In *Proceedings of ACM SIGACT News–SIGMOD Symposium on Principles of Database Systems*. ACM, New York, pp. 86–96.

[KuV85] KUPER, G. M., AND VARDI, M. Y. 1985. On the expressive power of the logical data model (extended abstract). In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. ACM, New York.

[LaR82] LANDERS, T. A., AND ROSENBERG, R. L. 1982. An overview of multibase. *Distributed Databases*. North-Holland, Amsterdam.

[Len87] LENZERINI, M. 1987. Covering and disjointness constraints in type networks. In *Proceedings of the IEEE Conference on Data Engineering* (Los Angeles, Calif.). IEEE, New York, pp. 386–393.

[LSA77] LISKOV, B., SNYDER, A., ATKINSON, R., AND SCHAFFERT, C. 1977. Abstraction mechanisms in CLU. *Commun. ACM 20*, 8 (Aug.), 564–576.

[LyK86] LYNGBAEK, P., AND KENT, W. 1986. A data modeling methodology for the design and implementation of information In *International Workshop on Object-Oriented Database Systems* (Asilomar, Pacific Grove, Calif., Sept.). IEEE, New York.

[LyV87] LYNGBAEK, P., AND VIANU, V. 1987. Mapping a semantic database model to the relational model. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (San Francisco, Calif., June). ACM, New York.

[Mac85] MACGREGOR, R. M. 1985. ARIEL—A semantic front-end to relational DBMSs In *Proceedings of the 11th International Conference on Very Large Data Bases*. Very Large Data Base Foundation, Saratoga, Calif., pp. 305–315.

[MSO86] MAIER, D., STEIN, J., OTIS, A., AND PURDY, A. 1986. Development of an object-oriented DBMS. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications* (Sept. 29–Oct. 2). ACM, New York, pp. 472–482.

[Mak77] MAKINOUCHI, A. 1977. A consideration on normal form of not-necessarily-normalized relation in the relational data model. In *Proceedings of the 3rd International Conference on Very Large Databases* (Tokyo, Oct.), pp. 447–453.

[MaD86] MANOLA, F., AND DAYAL, U. 1986. PDM: An object-oriented data model. In *Proceedings of the Workshop on Object-Oriented Databases* (Pacific Grove, Calif., Sept. 23–26). IEEE, New York, pp. 18–25.

[MaP86] MARYANSKI, F., AND PECKHAM, J. 1986. Semantic data models. Tech. Rep. CSTR 86-15, Dept. of Computer Science and Engineering, Univ. of Connecticut, Storrs, Conn.

[Min84] MINSKY, M. L. 1984. A framework for representing knowledge. In *The Psychology of Computer Vision*, P. H. Winston, Ed. McGraw-Hill, New York, pp. 211–277.

[Moo86] MOON, D. A. 1986. Object-oriented programming with flavors. In *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages and Applications*. ACM, New York, pp. 1–8.

[Myl80] MYLOPOULOS, J. 1980. An overview of knowledge representation. In *Workshop on Data Abstract, Databases, and Conceptual Modelling* (Pingree Park, Colo.). ACM, New York, pp. 5–12.

[MBW80] MYLOPOULOS, J., BERNSTEIN, P. A., AND WONG, H. K. T. 1980. A language facility for designing database-intensive applications. *ACM Trans Database Syst. 5*, 2 (June), 185–207.

[MBG86] MYLOPOULOS, J., BORGIDA, A., GREENSPAN, S., MEGHINI, C., AND NIXON, B. 1986. Knowledge representation in the software development process: A case study. Tech. Rep., Univ. of Toronto, Toronto, Canada.

[NEL86] NAVATHE, S., ELMASRI, R., AND LARSON, J. 1986. Integrating user views in database design. *IEEE Computer 19*, 1, 50–62.

[Nik84] NIKHIL, R. 1984. An incremental, strongly typed applicative programming system for databases. Ph.D. dissertation, Dept. of Computer and Information Sciences, Univ. of Pennsylvania, Philadelphia.

[O'Y82] O'DUNLAING, C., AND YAP, C. K. 1982. Generic transformation of data structures. in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*. IEEE, New York, pp. 186–195.

[Ris85] RISHE, N. 1985. Semantic modeling of data using binary schemata. Tech. Rep. TRCS85-06, Univ. of California, Santa Barbara, Calif.

[Ris86] RISHE, N. 1986. On representation of medical knowledge by a binary data model. In *Proceedings of the 5th International Conference on Mathematical Modelling*,X. J. R. Avula, G. Leitman, C. D. Mote, Jr., and E. Y. Rodin, Eds. Pergamon Press, Elmsford, N.Y.

[RoC87] ROGERS, T. R., AND CATTELL, R. G. G. 1987. Entity-relationship database user interfaces. Tech. Rep. Sun Microsystems, Mountain View, Calif.

[RoY84] ROUSSOPOULOS, N., AND YEH, R. T. 1984. An adaptable methodology for database design. *IEEE Computer* (May), 64–80.

[RoS79] ROWE, L., AND SCHOENS, K. A. 1979. Data abstractions, views, and updates in RIGEL. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. ACM, New York, pp. 71–81.

[Sch80] SCHNEIDERMAN, B. 1980. *Software Psychology: Human Factors in Computer and Information Systems*. Prentice-Hall, Englewood Cliffs, N.J.

[Sen75] SENKO, M. E. 1975. Information systems: Records, relations, set, entities, and things. *Inf. Syst. 1*, 1, 3–13.

[Shi81] SHIPMAN, D. 1981. The functional data model and the data language DAPLEX. *ACM Trans. Database Syst. 6*, 1 (Mar.), 140–173.

[Sho82] SHOSHANI, A. 1982. Statistical databases: Characteristics, problems, and some solutions. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). Very Large Data Base Endowment, Saratoga, Calif., pp. 208–222.

[SiK77] SIBLEY, E. H., AND KERSCHBERG, L. 1977. Data architecture and data model considerations. In *Proceedings of the National Computer Conference*. AFIPS Press, Reston, Va., pp. 85–96.

[SmS77] SMITH, J. M., AND SMITH, D. C. P. 1977. Database abstractions: Aggregation and generalization. *ACM Trans. Database Syst. 2*, 2 (June), 105–133.

[SFL81] SMITH, J. M., FOX, S., AND LANDERS, T. 1981. Reference manual for ADAPLEX. Tech. Rep., Computer Corporation of America.

[SBD81] SMITH, J. M., BERNSTEIN, P. A., DAYAL, U., GOODMAN, N., LANDERS, T., LIN, K. W. T., AND WONG, E. 1981. Multibase–Integrating heterogeneous distributed database systems. In *Proceedings of AFIPS National Computer Conference*. AFIPS Press, Reston, Va., pp. 487–499.

[SBM83] STEFIK, M., BOBROW, D. G., MITTAL, S., AND CONWAY, L. 1983. Knowledge programming in LOOPS: Report on an experimental course. *Artif. Intell. 4*, 3, 3–14.

[StR86] STONEBRAKER, M., AND ROWE, L. A. 1986. The design of postgres. In *Proceedings of International Conference on the Management of Data* (May). ACM, New York, pp. 340–355.

[SWK76] STONEBRAKER, M. R., WONG, E., KREPS, P., AND HELD, G. 1976. The design and implementation of INGRES. *ACM Trans. Database Syst. 1*, 3 (Sept.), 189–222.

[Su83] SU, S. Y. W. 1983. SAM*: A semantic association model for corporate and scientific statistical databases. *Inf. Sci. 29*, 151–199.

[Su86] SU, S. Y. W. 1986. Modeling integrated manufacturing data with SAM*. *IEEE Computer Magazine* (Jan.), 34–49.

[Su80] SU, S. Y. W., AND LO, D. H. 1980. A semantic association model for conceptual database design. In *Entity-Relationship Approach to Systems Analysis and Design*. North Holland, Amsterdam, pp. 147–171.

[SK88] SU, S. Y. W., KRISHNAMURTHY, V., AND LAM, H. 1988. An object-oriented semantic association model (OSAM*). In *AI in Industrial Engineering and Manufacturing: Theoretical Issues and Applications*, S. Kumara and R. L. Kashyap, Eds. American Institute of Industrial Engineers, Norcross, Ga.

[TYF86] 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), 197–222.

[TsK77] TSICHRITZIS, D., AND KLUG, A. C. 1977. *American National Standards Institute/X3/SPARC DBMS Framework: Report of the Study Group on Database Management Systems*. AFIPS Press, Reston, Va.

[TsL82] TSICHRITZIS, D. C., AND LOCHOVSKY, F. H. 1982. *Data Models*. Prentice-Hall, Englewood Cliffs, N.J.

[TsZ84] TSUR, S., AND ZANIOLO, C. 1984. An implementation of GEM–Supporting a semantic data model on a relational back-end. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. ACM, New York, pp. 286–295.

[Ull82] ULLMAN, J. D. 1982. *Principles of Database Systems*. 2nd ed. Computer Science Press, Rockville, MD.

[Ull87] ULLMAN, J. 1987. Database theory: Past and future. In *Proceedings of ACM SIGACT News–SIGMOD-SIGART Principles of Database Systems*. (San Diego, Calif., Mar.). ACM, New York.

[UrD86] URBAN, S. D., AND DELCAMBRE, L. M. L. 1986. An analysis of the structural, dynamic, and temporal aspects of semantic data models. In *Proceedings of the 2nd IEEE International Conference on Data Engineering* (Feb.). IEEE, New York, pp. 382–389.

[VeB82] VERHEIJEN, G., AND BEKKUM, J. V. 1982. NIAM: An information analysis method. In *Information Systems Design Methodologies: A Comparative Review*, T. Otte, H. Sol and A. Verrijn-Stuart, Eds. IFIP.

[Via87] VIANU, V. 1987. Dynamic functional dependencies and database aging. *J. ACM 34*, 1 (Jan.), 28–59.

[Was79] WASSERMAN, A. I. 1979. The data management facilities of PLAIN. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. ACM, New York.

[WiE80] WIEDERHOLD, G., AND EL-MASRI, R. 1980. Structural model for database design. In *Entity–Relationship Approach to Systems Analysis and Design*. North Holland, Amsterdam.

[WoK82] WONG, H. K. T. AND KOU, I. 1982. GUIDE: A graphical user interface for database exploration. In *Proceedings of 8th International Conference on Very Large Data Bases*. Very Large Data Base Endowment, Saratoga, Calif., pp. 22–32.

ZANIOLO, C. 1976. Analysis and design of relational schemata for database systems. Tech. Rep. UCLA-Eng-7668, Dept. of Computer Science, Univ. of California at Los Angeles.

ZANIOLO, C. 1983. The database language GEM. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. ACM, New York, pp. 207–217.

[Zlo77] ZLOOF, M. 1977. Query-by-example: A data base language. *IBM Syst. J. 16*, 324–343.

Received May 1986; final revision accepted December 1987.