Learning from AI:
New Trends in Database Technology

Technical Report 85–22

by
Lubomir Bic†
and
Jonathan P. Gilbert

August 1985

Department of Information and Computer Science
University of California, Irvine

Key Words and Phrases: *Artificial intelligence, database, data model, knowledge representation, user interface, user views.*

---

† This work was supported in part by NSF Grant MCS-8117516: The UCI Dataflow Project.

Abstract

Recently some researchers in the areas of database data modelling and knowledge representations in artificial intelligence have recognized that they share many common goals. In this survey paper we show the relationship between database and artificial intelligence research. We show that there has been a tendency for data models to incorporate more modelling techniques developed for knowledge representations in artificial intelligence as the desire to incorporate more application oriented semantics, user friendliness, and flexibility has increased. Increasing the semantics of the representation is the key to capturing the "reality" of the database environment, increasing user friendliness, and facilitating the support of multiple, possibly conflicting, user views of the information contained in a database.

Introduction

Traditionally, little interaction existed between researchers in the areas of databases and artificial intelligence (AI). Database management systems and AI knowledge representation formalisms evolved as solutions to apparently quite different problems. Recent trends have shown, however, that techniques developed in each of these areas may be adapted for use in the other area. Our major interest, in this context, is in the evolution of more expressive database modelling formalisms and the development of enhanced database management systems. It is from this one-sided perspective that we examine the potential overlap between these two major thrusts in computer science research.

Database management systems evolved in response to the need for efficiently maintaining increasingly large amounts of data. The relatively slow speed of secondary storage devices holding the data is one of the main limitations to database design. Hence, the internal organization and structuring of databases has been the primary focus of research in the past. Another major force that influenced database research was the need to share information among a variety of users. In such an environment, strict rules governing the manipulation of data had to be imposed to preserve the integrity of the database and to guarantee privacy for each user. These needs lead to the development of several basic data models, the best known of which are the *relational*, the *hierarchical*, and the *network* models.

With the steadily increasing needs for user-oriented systems, new trends in database technology have evolved outside of the scope of traditional data models. In this paper we are concerned with two closely related efforts:

1. The incorporation of more *semantic modelling capabilities* into database models.

2. The development of better *user environments* which include *user friendly interfaces* and the support of different *user views* of the information content and organization of the data.

There is a definite relationship between the above two issues: A more powerful and flexible user environment necessarily requires a system with increased semantics in the stored data, and vice versa. Hence, the two issues may jointly be viewed as pointing toward a new goal — the development of database systems which incorporate a higher degree of *intelligence.*

As computers have become commonplace, the number and type of users have changed dramatically. This change has had a significant effect on researchers in the database field. Many applications although conceptually simple (e.g. report generation) require extensive manipulation of low level data structures which is a significant task for even an experienced programmer. Unfortunately, experienced programmers are no longer the majority of database users. More intelligent database management systems capable of interacting with both the casual user and the database structure have become a necessity. In order to construct more intelligent user interfaces, *meta-knowledge* must be included in the database. This is information about the objects and relationships which collectively represent the world of the database enterprise being modelled — in other words, it is information about the database itself. Unfortunately the traditional record-oriented data models do not have the facilities for representing meta-knowledge. We will look at two basic approaches to solving this problem. The first is the development of new "more powerful" *semantic* data models that model database concepts at a higher, more user oriented level. The second is to build a higher level conceptual database on top of a conventional database. Both solutions are promising; however, neither is without problems of its own. Semantic data models tend to be extremely complex and,

therefore, are at best only partially implemented. The major problem with many conceptual data models occurs because they are mapped onto a data model which is incapable of incorporating meta-knowledge. In such cases it becomes necessary to record the database semantics separately from the data.

Similar problems are being studied extensively by researchers in AI[1,2,3,4]. While their motivations may be quite distinct, many results are closely related to the needs of modern database management systems. Techniques and knowledge from AI may be usefully applied, however, to problems in database development. A number of novel approaches demonstrate that a major shift from traditional database technology towards AI has already started and will influence the development of new generations of data management systems.

The Three Classical Models

The need to organize data in some well defined, rigorous manner has lead to the development of a number of data models, the best known of which are the relational, the network and the hierarchical models.

According to Kent[5] all three of the classical models are based on the *record model.* In a record based system, data are arranged in *fixed* linear sequences of field values. Records are machine oriented — they provide an efficient basis for *storing* and *processing* data but are not, in the least, user oriented. Each of the classical models is based on some *idealized data structure* and has a set of operations associated with this structure.

The Relational Data Model

In a relational database[6] information is organized in tables. Each table has a unique name and is a special case of the set-theoretic *relation.* The rows of a relation

table are called *tuples*; columns are called *attributes*. Each column within a relation has a unique name. The set of values from which actual values in a column are selected is called the *domain* of the attribute and may be shared among different columns. A relation name and its set of attribute names is called a *relational schema* and a collection of relational schemas is a *relational database*. One of the most important features, and perhaps the biggest drawback, of the relational approach is that associations between tuples are exclusively represented by *attribute values drawn from a common domain*.

The main attraction of the relational model is its mathematical clarity which facilitates the formulation of non-procedural, high level queries and thus separates the user from the internal organization of data. Among the three classical database models, the relational model is, therefore, considered the most user oriented. It is, however, far from being able to satisfy the needs and requirements of an increasingly diversified user community.

The Hierarchical Data Model

Data in the hierarchical model[7] is organized in simple tree structures; a hierarchical database is a collection of such trees — a *forest*. This organization necessarily requires that some types of records are subordinate to others. In fact, a crucial characteristic of the hierarchical model is that any given record takes on its full meaning only when seen in the context of the hierarchy. Furthermore, operations are usually procedurally oriented and are at a lower level than relational database operations since much of the internal data organization must be visible to the user.

A typical hierarchical database contains trees of many different heights, where trees of height 1 are not uncommon. This reveals a misfit between the information to be modelled and the rigid hierarchical structure into which it must be forced.

Another indication that a strict hierarchical structure is not a natural model for many database applications is the necessary use of the *virtual logical record type*, which is a pointer to another record type. These pointers provide a mechanism which allows a record type to "exist" in several trees or even in two places in the same tree. Without this concept, large numbers of records would have to be replicated when transforming many-to-many relationships into a hierarchical structure, resulting in an unnecessarily high degree of redundancy in the data representation.

The Network Data Model

The network data model[8] was introduced by the Data Base Task Group (DBTG) of the Conference on Data Systems Languages. In this model, information is represented by *records*, interconnected via *links* to form a *directed graph*. Records are grouped into *sets*, each of which consists of one *owner* record and zero or more *member* records.

The set concept is the most important single aspect of the DBTG model. The *set type* is defined in the schema by specifying the types of the owner and the member records. The major deficiency of the DBTG model is that many-to-many relationships cannot be represented directly. In order to accommodate many-to-many relationships, the concept of *connection record type* is included in the DBTG model. A connection record would contain the information common to the two record types linked by the many-to-many relationship.

The network data model is, in general, more symmetrical than the hierarchical data model. The basic operations on a network database are, therefore, more symmetrical than those on a hierarchical database. However, they tend to be procedural, implying that the user must be familiar with the lower level of data representation and, consequently, are more complicated than operations on a relational database.

The User Environment

The User Interface: In terms of providing users with friendlier interfaces, the relational model is the most interesting of the classical models because of its inherent suitability for supporting high-level, non-procedural languages. Non-procedural languages allow users to express queries (and updates) in terms of the properties of the resulting set rather than the organization of the database. Each provides a simple query language which, compared with a general purpose programming language, is relatively easy to master. The user is, however, forced to explicitly manipulate the data structures (e.g. tables) in the database. Furthermore, facilities for adapting the database to new uses — a problem of deriving multiple user views — are very limited.

Supporting Multiple User Views: The second aspect of providing user oriented interfaces is the support of *multiple user views*. In any multi-user environment the needs of various users will be multifarious. In order to support this variety of user expectations, the interface to an integrated database must support multiple user views of the data. Most research concerned with user views in the classical data models focused on the relational model. Relational views are usually simple variations of base relations. In particular, virtual relations, representing a type of derived data, are defined in terms of existing relations. A user view of the database is then the collection of base and virtual relations. Virtual relations may be defined using data abstraction techniques[9], which hide the original underlying base relations. This means that a user cannot, in general, tell which relations are base and which are virtual. Internally, however, just the virtual schema is stored with the database and *not* the relation itself. One of the major problems with defining views in this way is the difficulty of performing update operations via a user view. It is not always possible to translate an update request on derived rela-

tions to an *unambiguous* request on base relations; therefore, in order to maintain data integrity, it may not be permissible to perform unrestricted modifications on derived relations; Rowe and Shoens[9] describe several strategies for dealing with this problem. The facility for defining and maintaining user views is often embedded in the language of the user interface. In this case, the interface must automatically handle the translation of a query from virtual to base relations and then translate the result back to the form expected by the user.

In addition to facilitating user views, virtual relations have several other potential uses. They can be employed, for example, to predefine certain information retrieval operations which are complicated to specify yet frequently executed. Virtual relations may also be used to produce *snapshots*, i.e. materializations of a view at some point in time. Snapshot relations are *read-only* relations and can be useful when applications require access to earlier versions of the database. In a distributed environment local snapshots may be used to *approximate* remote data, thus avoiding the expensive maintenance of a local replica.

Higher Level Models

Inadequacies of Classical Database Models

As users of computer systems have become more diversified, the need for better ways to model concepts and processes of the application domain has increased. Unfortunately, the classical models are highly machine oriented; user friendliness is only of secondary importance. For this reason, all three classical models are considered to be *syntactic* data models. Attempts have been made to extend the classical models to incorporate more *semantics* into data as well as to embed them into environments with more user oriented interfaces. We will first examine the most significant deficiencies of the classical database models, and then discuss several

approaches that database researchers have taken to enhance their capabilities.

1. Because the primary concern of most database research was efficiency in accessing stored data, classical data models are machine rather than user oriented.

2. Most database models have only two levels: the database schema, which describes the data types constituting the database, and the actual collection of records, which are instances of the existing types. There are no provisions to extend the two levels into a more general hierarchy or taxonomy of types, subtypes, and instances, even though this extension would increase the model's expressive power and take advantage of the concepts of property inheritance across different levels of the hierarchy.

3. Another problem is the inability to distinguish between a type and a set — records which form the schema usually represent both. Consider, for example, an employee database. The schema will describe the form of an employee record by listing the names of possible attributes each employee might have. The actual attribute values are kept with each employee record. From this point of view, the schema contains the description of a "typical employee". Alternately, the schema could be interpreted as representing the "set of all employees" constituting the database. Taking the latter point of view, it should be possible to include attributes that apply to the set as a whole but not to each individual element (e.g. the set cardinality or the average salary of an employee). Unfortunately, most conventional database models have no facilities for making this distinction.

4. When modelling an enterprise, elements and concepts representing it may be viewed from different perspectives depending on the application.

In particular, the concepts of entity (object), relationship, and attribute are interchangeable. Similarly, what is considered a type from one user's point of view may be seen as an instance in another. The ability to model such phenomena, referred to as *relativism*, is missing in conventional database models.

5. The distinction between data and program in a database system was always clear in the past but is disappearing as behavioral aspects of modeling are considered. The need for recording events, which require the inclusion of temporal information has recently been recognized. With the inclusion of events a new area of problems related to time-oriented integrity constraints has emerged.

The Entity-Relationship Model

One of the first steps taken toward developing a higher level data model was the Entity-Relationship (ER-)model, introduced by Chen[10]. It was aimed primarily at the first deficiency listed above — an increased orientation toward the user needs and expectations rather than machine efficiency. The ER-model may be viewed as a *generalization* of the three classical data models. Chen presented it as "... a *basis for unification of different views of data* ..." — in other words, it emphasizes the similarities and not the differences in the classical models.

The basic components of the ER-model are *entity sets* and *relationship sets*, where each entity set and each relationship set represents a generic classification of *entities* and *relationships*, respectively. The membership of entities in entity sets is determined by a predicate. Hence each entity may be a member of more than one set. For example, a person could be a student as well as an employee of a given university.

Relationships are associations among entities and are defined as mathematical n-ary relations of the form $R = \{[r_1/e_1, ..., r_n/e_n]$ *where* $e_1 \in E_1, ..., e_n \in E_n\}$. The terms r_i represent *roles* played by the corresponding entity e_i in the relationship R. For example, if "marriage" is represented as a relationship between a man and a woman, then the roles "husband" and "wife" may be associated with the two participating entities, "man" and "woman", respectively.

Both entity and relationship sets may have properties called *attributes* associated with them, where each attribute is defined as a mapping between the entity or relationship set and a *value set*. *Multivalued* attributes are permitted in the ER-model.

The ER-model is used primarily as a *database design tool*; the actual database is then implemented with some other data model. The key step in designing a database enterprise is determining the point of view from which the real world is to be modelled. When the ER-model is used, the entity sets and the relationships among them are chosen by the designer. The procedure for this selection cannot be precisely defined — it is a rather subjective process. Once the entity and relationship sets have been selected, the next crucial (and subjective) step is the selection of the relevant attributes. The design process is facilitated by a pictorial design tool called *entity-relationship diagrams*, in which rectangular nodes represent entity sets, circular nodes represent attributes and diamond shaped nodes represent relationships. The arcs in an entity-relationship diagram are undirected and may be labelled with appropriate role names. The numbers on these arcs indicate the possible relationship type, that is, 1-to-1, 1-to-many, or many-to-many. Figure 1 is an illustrative example of a simple ER-diagram.

The richer semantic basis of the ER-model lead to the development of the language CABLE[11] (ChAin Based LanguagE), which provides a user friendly interface

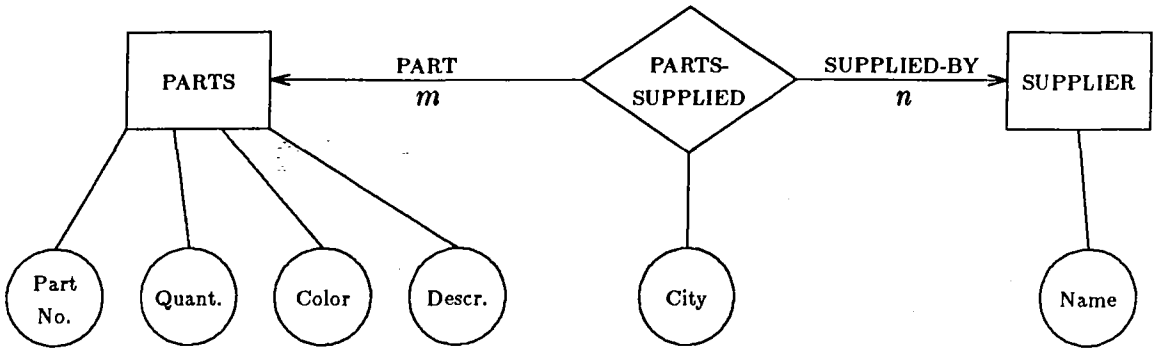

Figure 1: An Entity-Relationship Diagram.
PARTS-SUPPLIED is a many-to-many relationship between PARTS and SUPPLIERS. Each SUPPLIER supplies m PARTs and each part is SUPPLIED-BY n suppliers.

to an ER-database. CABLE queries are specified as chains of *beads*, where each bead specifies a set (entity or relationship) and some restrictions to be applied to elements of that set. These chains form a pattern which the system tries to match against the ER-database. Elements which conform to this pattern constitute the answer to the query. An important feature of CABLE, increasing its user friendliness, is that it permits beads to be omitted from the query as long as enough information is given to make each chain unique.

While efficiency in the ER-model was not of major concern, it gave rise to another project whose primary objective was to develop a database system suitable to execute on a highly parallel computer architecture. The resulting scheme, named the Active Graph Model[12], rejects the sequential von Neumann model of computation by adopting a data-driven view of processing. Since the need for centralized control is eliminated, the Active Graph Model is capable of exploiting the potential bandwidth of a significant number of disk units as well as the computational power of a correspondingly large number of asynchronous processing elements.

In summary, the ER-model, together with its various extensions and derivatives, is a significant improvement over the classical database models. In particular, the explicit distinction between entities and relationships and the introduction of

distinct types of arcs (i.e. role arcs) has significantly increased its expressive power. Unfortunately, it is not always easy to categorize things as either entities or relationships. This difficulty can be illustrated with the concept of *marriage* which, from one point of view, can be categorized as a relationship between a man and a woman, and from another, as a legal entity, in which the man and the woman participate as attributes. The ER-model does not allow any specific "thing" to be both an entity and a relationship. Hence a decision in favor of one or the other must be made by the database designer. Furthermore, the model does not allow for the fact that some information cannot be categorized as either an entity or a relationship. A marriage, for example, may be interpreted as an *event* or a *contract*, with which a certain time interval is to be associated. To resolve such problems, models with considerably greater expressive power must be introduced.

Extending the Relational Data Model

The Hierarchical Semantic Model: Numerous attempts have been made recently to extend the usefulness and the expressive power of the relational model. Most of these have their roots in the work of D. Smith and J. Smith, who introduced two important concepts to database modelling: *aggregation* and *generalization*[13]. The first of these permits a relationship between certain objects to be viewed as an aggregate object; at the same time, properties of the individual objects may be ignored, implying that aggregation is a form of abstraction. For example, an "employee" could be viewed as the aggregation of lower level objects, such as "name", "address", "salary", "dependents", etc.

The second concept — generalization — is aimed at modelling a hierarchical ordering of objects constituting an enterprise. It is an abstraction which permits a class of objects to be viewed as a typical (generic) object of that class. It allows attributes with a common value for all members of the class to be recorded with

the generic object, rather than being replicated many times at lower levels. For example, the fact that all secretaries have typing skills may be kept with the generic object "secretary" and inherited by each individual belonging to that class. The generalization concept, however, does not distinguish between attributes which are inheritable by individuals and those which apply to the set as a whole. Thus the model would, for example, permit the recording of the average age of all employees as a value attached to the generic object "employee" even though it applies only to the set as a whole; i.e., it may not be inherited by any individual in that set.

The aggregation and generalization abstractions form the basis for what became known as Smith and Smith's *Hierarchical Semantic Model* (HS-model). This system comprises a methodology for database modelling. Schemas are built by stepwise decomposition of initial objects into smaller components along the aggregation and generalization hierarchies. A set of semantic and syntactic rules is provided, which guarantee that the decomposition process yields a collection of valid relations in Codd's relational model. Thus the HS-model may be viewed as a significant extension of the classical relational model toward a more accurate and more powerful modelling tool for database applications.

The Tasmania Relational Model: Codd[14] describes another extension of the relational data model, named after the conference site at which it was originally presented. Like the HS-model, the objective is to capture more meaning in data to facilitate the process of database design and to permit the system itself to respond in a more intelligent manner. It encompasses many forms of abstraction (including aggregation and generalization); the approach, however, is much more theoretically oriented than in the HS-Model.

As a first step, two kinds of data semantics are identified: *atomic* semantics, representing the basic building blocks of the model, and *molecular* semantics, which

permit clusters of atoms, constituting meaningful units of information, to be formed. Any n-ary relation is interpreted as an atomic fact. These may be combined according to the following four rules of the molecular semantics:

1. *Cartesian aggregation* is the type of aggregation described by Smith and Smith.

2. *Generalization* (also defined by Smith and Smith).

3. *Cover aggregation* extensionally describes a subset of objects — a convoy of ships is often used as an example of this type of aggregation.

4. *Event precedence* — entities of type event have a start time and an end time. For some applications, ordering events is important. This is facilitated by *alternative* and *unconditional successor* and *precedence relations* which, respectively, define what *may* and *must* follow and precede a given event. For example, suppose we have an inventory database that includes two event entities called ORDERS and SHIPMENTS. It is desirable to ensure that only those goods ORDERed are received in SHIPMENTS. An *unconditional precedence* relation would be used to specify that all shipments must be preceded by an order for the goods delivered.

These enhancements allow the relational model to represent situations that can be represented by any semantic data model. Unfortunately, incorporating the additional semantics, together with the corresponding operators, has made this model *extremely* complicated. For this reason, no claim regarding the ease of use is made for the extended relational model. In fact, it is "... *intended primarily for database designers and sophisticated users* ..."[14].

Although the extended relational model is more data structure than user ori-

ented (and, therefore, by many definitions would not be considered to be a semantic data model), it does provide four different types of user interface: *tables* are used for extensional information, a *set-theoretic* interface can be used to specify searches without including navigational information, an interface based on *inferential predicate logic* for stringwise expression of intensional information, and a *graph-theoretic* interface which provides a pictorial medium to aid in the design and maintenance of the database.

The Semantic Data Model (SDM)

The Semantic Data Model (SDM)[15] was developed as an alternative to the classical data models, which were considered to be "... *too low level and machine oriented, requiring the users to think in terms of representation rather than in terms of meaning* ...". SDM was designed to be a high level user oriented model for database application environments. Contrary to knowledge representation systems in AI, the objective of SDM is not to model the "real" world; rather, it is a model of a database enterprise which is quite different.

An SDM database is a collection of *classes* which represent "relevant abstractions" in a particular application environment. There are two types of class — *base* and *nonbase* classes. The former are defined independently of other classes while nonbase classes are defined in terms of other classes. Classes are logically linked to other classes via *interclass connections* and are composed of entities called *members*. There are five types of entity: the *concrete object* (e.g. Cars, Students), point and duration *events* (actions and activities), *abstractions* (e.g. the generalizations described previously) and *names* which are identifiers for objects and events. Both classes and entities have *attributes*, which may fall into three categories: *member attributes* (each member of a class has this property), *class-determined attributes* (all members of a class have the same *value* associated with this property) and *class*

attributes (which are properties of the class as a whole).

SDM has a structured user interface that provides facilities for three different classes of users: *naive* nonprogrammers, *routine* users and *experienced* programmers. For naive nonprogrammers, SDM provides an interactive query system that can make suggestions, offer advice, and generally help the user to formulate queries. Routine users are those executing predictable, repetitive tasks. This class of user performs the "busy work" associated with database maintenance, i.e. database editing, which involves simple updates requiring only minor changes to the database contents, but not to its structure, and periodic report generation. Finally, for the experienced programmer, it is suggested that the SDM formalism *could be* integrated into a conventional general purpose programming language (like COBOL) — a non-trivial, and for the most part unexplored area.

When comparing the higher level models presented above we notice that all of them present methodologies for designing more intelligent database environments. The ER-model presents a unifying view of data, which permits the designer to organize knowledge in terms of his own human perception of the enterprise to be modelled, rather than in response to the constraints imposed by a computer architecture. Together with the language CABLE, the ER-model also presents a step toward a high-level user oriented interface. While the HS-model and the Tasmania relational model are extensions of the classical relational data model, the ER-model is meant to aid in the design of databases that will be implemented with a classical data model. Finally, SDM, with its built-in structured user interface, is more user oriented than either the extended relational model or the ER-model; thus it is both an improvement and an alternative to these models. Although all these approaches fulfill many of the requirements and expectations of information management systems, the desire to include more "intelligence" (for example, in-

ferential information, human oriented semantics, and more flexible user interfaces) still persists. The higher level models described above are representative of research done by database researchers on "semantic data models". We will now briefly examine some closely related work on "conceptual database models" which attempt to exploit techniques and results from AI in order to produce intelligent database systems.

The AI Connection

We have shown a shift in database research away from the traditional machine oriented data model toward more user oriented semantic models. In this context, database researchers have begun to recognize the value of AI research in knowledge representation. There are, however, a number of significant differences between the type of information that database researchers are concerned with and the type of information studied in AI. In the former, representations tend to be biased toward a large number of instances of a small number of formatted data types. Knowledge representations in AI, on the other hand, are designed to deal with a small number of instances of a much larger variety of types and classes. This implies that knowledge bases tend to be rather amorphous while databases are highly structured. Another significant difference is in the amount of implicit information. Knowledge base queries must often use inferential information inherent in the structure of the data to produce a result. In databases, such capabilities either have a very rudimentary form or, more typically, are not present at all. Finally, knowledge bases are usually special-purpose systems, aimed at a particular application, while databases are often constructed to facilitate the needs of a community of users whose requirements may be quite diverse.

Despite these fundamental differences, there are a number of concerns common to both areas. In particular, the experience gained by AI researchers in attempt-

ing to model the real world is invaluable when applied to data modelling in a database environment. Database researchers are beginning to apply AI techniques to database problems particularly in the areas of *conceptual modelling*, *user interfaces*, and *relativism and user views*.

Conceptual Modelling

A conceptual database model is object- rather than record-oriented. The basic building blocks are entities which have fixed properties associated with them and can be created and destroyed for the duration of the application. Knowledge is organized along many dimensions — there may be aggregation, generalization and classification hierarchies, or the information may be partitioned into spaces corresponding to particular user views. Conceptual models are usually built on top of an existing (classical) database management system. Thus knowledge must be organized, structured and represented so that the translation from conceptual model to data model is easy.

To illustrate how AI has influenced this area of research, we present several conceptual models that have applied one or more of the following knowledge representation techniques developed in AI: *semantic networks, logic, procedural* representations, and *frame based* representations. It should be noted that the following discussion is not intended as an exhaustive survey of existing conceptual models; the goal is to demonstrate the four approaches to knowledge representation in databases, so we consider only general-purpose conceptual application models. We have chosen to group them according to the four categories of knowledge representation in AI, but most approaches combine more than one of these techniques.

Using Semantic Networks: Semantic networks[16] were originally developed as a psychological model of the human mind. They have since been used by com-

puter scientists to model knowledge in various intelligent systems. Semantic networks are a knowledge representation formalism that have *labelled nodes* and *labelled arcs*. Nodes usually represent *objects*, *concepts* or *situations* in the domain being modelled while arcs represent *relationships* between nodes.

Roussopoulos and Mylopoulos[17] present a data model in which semantic networks are used in the design and implementation of relational databases. A database is designed with the semantic network formalism, then translated into a relational database schema. Semantic operators are defined on the semantic networks, not directly on the database. Since there is a strong similarity between the semantic operators and ordinary relational operators, mapping is simple. One major problem with this approach is a substantial loss of meaning during the translation of the semantic network into a relational schema. To compensate it is necessary to store the semantic network together with the schema and use it, for example, to maintain database consistency.

In a later work, Mylopoulos et al[18] developed a system called TAXIS, characterized as *"...a language for the design of interactive information systems..."*. Its semantic network structure is based on procedural semantic networks. The underlying data model is once again relational. This time, however, it is extended to include *classes* (of objects) and a *generalization relationship*, which can be used to implement an *IS-A hierarchy*.

Following the principles of data abstraction, TAXIS uses appropriate procedures to integrate the database. In addition to exploiting knowledge representation techniques from AI, it combines ideas from programming language research with the basic principles of the relational data model. This cross-fertilization process simplifies use of the system. Applications are described at a higher level than the underlying relational database, and the application description can itself be manip-

ulated by programming language commands. Since TAXIS also provides a higher level *application specific* user interface, it goes a step beyond the semantic network database system,[17] described previously.

Using Logic: Mathematical logic was used as a formal language by philosophers and mathematicians long before the first electronic computers emerged. Its value to computer science was recognized by AI researchers during early attempts to construct automatic theorem provers. Since then the principles of logic have been developed further and have given rise to a new discipline called *logic programming*, which has gained considerable attention in both AI and database technology.[19]

A logic program is a set of *clauses* of the form $p_0 \leftarrow p_1, ..., p_n$. Each p_i is called a *literal* and has the form $p(t_1, ..., t_m)$, where p is a *predicate* symbol and $t_1, ..., t_m$ are *terms*. Terms may be constants, variables, or functors. p_0 is called the *head* or *conclusion*, and p_1 through p_n form the *body* or *conditions* of the clause. In terms of operational semantics, the meaning of a clause may be paraphrased as follows: in order to prove that p_0 is true it is sufficient to prove that p_1 through p_n are true. A clause with an empty set of conditions is always true; it is called an *assertion*. A clause with an empty head, on the other hand, is interpreted as a *goal* which the system tries to solve using Robinson's resolution principle.[20]

A logic program can be considered a natural extension of the relational database model because any relational tuple can be expressed as an assertion, i.e., a predicate of the form $p(t_1, ..., t_m)$. The use of mathematical logic in describing such (conventional) database models has helped to solve a number of important problems. Among these are the definition of formal query languages, the treatment of incomplete information (null values) in databases, and the definition and enforcements of integrity constraints. (Gallaire et al[21] provide a comprehensive survey.) The primary attraction of logic here is the elegant formalism capable of express-

ing facts, deductive information, meta-information (such as integrity constraints), and queries, in a uniform way. For example, to represent the fact that "Bill is Jane's father," the following single clause, consisting of a head literal only, might be used: father(jane, bill) ←. To formulate a query such as "Who is Jane's father?", a similar clause, this time with an empty head literal, would be constructed: ← father(jane, X), where X is a variable to be bound to the desired answer. Thus a single scheme — logic programming — can serve as both a database definition language and as a practical high-level query language. Consequently, the distinction between "database" and "program," which was always clear in the past, is beginning to disappear.

The main contribution of logic programming to database research, from the AI point of view, is the incorporation of deductive information. In many situations, it is advantageous to represent certain facts intensionally in terms of a logical implication, rather than extensionally as lists of assertions. For example, to represent relationships such as "grandfather," it is possible to state that grandfather is defined as the father of a parent, i.e., grandfather(X, Y) ← father(X, Z), parent(Z, Y). Statements of this form make it unnecessary to store all the individual grandfather relationships between elements explicitly. Instead, the system applies inference rules using this definition when queried about any particular grandfather relationship.

There is a strong relationship between logic programming and semantic networks.[22] Deliyanni and Kowalski have shown that logic programs, restricted to binary predicates, may be viewed as a form of extended semantic networks. Using this approach, a knowledge base, expressed in the form of a logic program, may be transformed into a semantic network of *explicit* facts (assertions) and a collection of network *templates* which represent deductive information and queries. Extracting information from such a knowledge base then corresponds to finding *matches* between portions

of the underlying network and the templates involved in a given query.

The principles of data-driven computation may usefully be applied to this model to make it suitable to implementation on a highly parallel computer architecture.[23,24] In this approach, the semantic network, rather than being only a passive representation of knowledge, is a *dataflow graph.* That is, each node is an *active* component capable of accepting, processing, and emitting value *tokens* (messages) traveling asynchronously along the network arcs. These tokens are used to carry portions of the templates to be matched against the semantic network. Each token is injected into a node of the network, replicated and sent in possibly many directions in the search of patterns that match the given template.

The processing and propagation of tokens is distributed throughout the network and is performed in a *data-driven* manner; i.e., each node of the network represents an independent unit of computation, whose execution is triggered solely by the arrival of a token. The main advantages of this approach are the *lack of centralized control* and the *lack of a centralized memory*, which, under the von Neumann model of computation, are the main limitations to high performance. The data-driven model, on the other hand, permits many independent processing elements to be engaged simultaneously in the processing and forwarding of tokens, and thus a highly-parallel computer architecture can be constructed to support that model.

Using Procedural Semantics: A procedural knowledge base system consists of a collection of processes, an activation mechanism, and a control structure. Control structures, like heuristic search techniques, help limit the size of the problem being solved. The activation mechanism is usually some form of pattern matching operation; when a procedure's pattern is matched, it may execute. Frequently, a given pattern will match several procedures; which of these is to execute may be determined by a number of different selection strategies. At one extreme, a fixed

ordering of the procedures may be maintained and the first match executed (various production systems[25] use this strategy). At the other end of the spectrum, the order may be completely undetermined; for example, in PLANNER,[26] each procedure is a *demon* that watches for a particular pattern to occur and executes as soon as that pattern is detected. SWYSS and IM illustrate the use of procedural representation in database research.

SWYSS[27] (Say What You See System) was originally designed to answer questions about static scenes. The experience gained from this work is now being applied in the database domain to develop a model displaying a higher degree of intelligence. The system is based on a functional data model; its atomic units are unary and binary functions that roughly correspond to attributes and relationships in the entity-relationship model. Concepts denote elements of the world being modelled and may represent individuals, sets, or generalizations. It is possible for a single entity to be represented by more than one concept, and it is even permissible for these concepts have conflicting properties. For example, the concepts "Dave is an good swimmer" and "Dave is a bad worker" both describe some aspect of the individual "Dave". The system has a taxonomy hierarchy in which concepts are connected via ISA (subset) and INSTANCE-OF (set membership) arcs.

The Information Management (IM) Project[28] at the USC Information Sciences Institute is an information and computing system based on the concept of *Active Databases*. In an active database environment the emphasis is on providing the user with real time up-to-date information. Different from a conventional database which presents information based on a static snapshot of the database at the time that the query is processed, the information displayed by an active database is exactly the same as the information contained in the database at the moment that it is displayed. IM supports dynamic views that allow a user to make changes to

the data and see the results immediately. In addition, a user can move through the database by following a chain of associations between entities. Data objects can be accessed via patterns; that is, instead of using a key to refer to an object, a user can reference objects by specifying a set of attributes and/or relationships associated with the object. A procedural representation scheme also helps automate repetitive actions and supports the use of demons to send mail, produce reports, etc. The underlying data model is similar to the entity-relationship model in that multi-valued attributes and n-ary relationships are supported. Furthermore, types of objects are arranged in a lattice to facilitate attribute inheritance.

Using Frames: Frames [29] are a combination of two ideas, one from programming language research and the other from AI. Similar to a well structured program, a frame is a collection of small "chunks" of information that are easy to understand and modify. The knowledge is domain-specific and, like the rules in a production system, is a list of facts that contain no information about how or in what order they are to be interpreted. Frames are a dynamic representation; they change constantly as *short-term memory* is updated — when the system learns more facts or changes its orientation, a frame changes.

The Knowledge-Based Management Systems (KBMS) Project[30] at Stanford University addresses the problems of intelligent processing in large databases. It upgrades the database from a passive store of information to an active information understander. The database must, therefore, contain knowledge about the application domain in addition to the actual data. KBMS combines two distinct conceptual models. The first of these, called the *structural model*, is used for designing the database; the basic idea is to design separate models for each individual application, then integrate them into a single database. The second model is a frame-based representation, used to model domain-specific knowledge; it is partic-

ularly useful for processing time-oriented data and has been successfully applied to the analysis of large medical databases.

The User Interface

Although these non-procedural database interfaces described previously are much more user oriented than those provided by procedural programming languages, such as COBOL, they are still based on formal query languages. The necessity of learning a formal language, usually with little interactive help or guidance on the part of the system, is a serious deterrent to database users, especially those with little computer science background. To overcome this problem researchers are designing more intelligent interfaces which incorporate knowledge about the domain being modelled, the objects and organization of the database, and the needs of individual users or user classes. This information then may be used to facilitate the interaction with the system, for example, through a graphic or a menu driven interface, or with natural language, each of which, in theory, requires less expertise from the user. In addition to accepting queries from users, it is desirable that the interface be able to process and answer questions "intelligently" — a task which cannot be done in a conventional syntactic database system. Projects in AI-oriented solutions to user/database interfaces illustrate the impact of AI on database interfaces. (Additional sources of information may be found in the references[1,2,3,4].)

Using Natural Language: Natural language interfaces have two obvious advantages over non-procedural languages: users can express themselves in familiar terms without knowing about the logical structure of the information and without having to learn a specialized formal language. Natural language interfaces do, however, suffer from some serious drawbacks. Because of the large body of domain specific knowledge needed by an intelligent interface, it is often difficult to adapt existing interfaces to different application environments. A more serious problem with

natural language interfaces is, however, the inherent *ambiguity* of natural language itself, which exacerbates describing problems to a machine.

Current natural language based interfaces either use a very limited subset of natural language (in vocabulary and syntax, as well as semantics), or must interact with the user in the form of a dialog to clarify the query. Most existing systems take the first approach by restricting the interface to a domain specific subset of natural language. RESEDA[31], which deals exclusively with biographical data, SWYSS[27], which was designed to answer questions about static scenes, and HAM-ANS[32], which contains fishery information collected on two international expeditions, are examples of systems of this kind. Unfortunately, using a restricted subset of natural language may cause significant difficulties even for the sophisticated user. For example, if a query is rejected by the system as unanswerable or if it is answered in some unexpected way, it may be very difficult to determine why the request failed. One solution to this problem is to use the second approach mentioned above, i.e., design an interface which interrogates the user and gradually formulates an unambiguous query. Codd's RENDEZVOUS[33] was a fairly successful attempt to do just that. Unfortunately, a dialog with RENDEZVOUS is extremely cumbersome because the user is forced to deal with a more or less conventional database structure rather than conceptual objects.

In contrast to special purpose knowledge systems developed in AI, database systems must be suited for a variety of different applications. A part of this requirement is the issue of adaptability of a natural language interface. Several recent research projects were devoted to studying this problem. In the PROLOG based Chat-80[34] *horn-clause logic* is used for formalizing *and* implementing all aspects of the relational database system. The natural language front end is unique in that it has been implemented in several languages (Spanish, English and French) and can be

fairly easily adapted to different database environments — small sample databases have been developed for both geographical information and employee records. The Chat-80 systems are efficient and compare favorably with relational systems without natural language front ends. However, because the database is implemented entirely in PROLOG it must reside in *virtual memory*; therefore, Chat-80, in its present form, is not practical for implementing a large database. Another transportable natural language interface is currently under development at Duke University[35]. It differs from Chat-80 in several ways, the most significant of which are the following: (1) it interacts with loosely structured text files, and (2) the system is capable of learning about new domains from its interactions with a user. The natural language processor is called the Layered Domain Class (LDC) system that consists of two subsystems called *prep* and the *User-Phase processor. Prep* is the "learning" part of the system. During a user's initial session, *prep* produces a description of the *conceptual domain* which is used subsequently by the user-phase subsystem. The latter is a real time interactive natural language interface.

Semantic Query Optimization: Natural language processing is not the only AI technique applied in the development of database interfaces. Another area in which AI had a significant impact is *semantic query optimization*, which permits the interface to act as an intelligent mediator between the database and the user. There are two distinct motivations for performing semantic query optimization — performance and convenience. In the first case, the system uses knowledge about the domain being modelled, possible processing strategies, and the physical structure of the database to transform the original query into one that produces the same answer but can be processed more efficiently. In the second case, semantic query optimization is used to simplify the use of the system by attempting to find a "better" answer to a query than a simple yes/no or a list of values satisfying the given restrictions. For example, if Jane is a secretary, the query "Which courses are

taught by Jane" should not simply produce an empty list; rather, it should inform the user that the question is semantically incorrect since Jane is not an instructor. The MRPPS 3.0 system[36], an experimental logic based relational data base, was designed to cope with such problems. It uses many-sorted logic (an extension of the more commonly used logic of first order predicate calculus) in which predicates and terms may belong to different semantic categories. These categories are organized into a semantic graph, which is an intrinsic part of the system. Since quantifiers are restricted to range over different categories it is possible to define the legal domain for each predicate attribute and thus detect statements that are syntactically correct but meaningless in terms of the semantic knowledge they represent.

Semantic query optimization can go beyond a simple detection of semantically meaningless queries by offering more elaborate services to satisfy users' expectations. An example of such a system is the AI/Database project at the University of Pennsylvania (UP). Its major objective is to cope with the differences in the users' and the system's view of the stored information. To understand this type of problem, consider the following situation. Suppose the system is given the query "Which suppliers supply pencils?". If *all* suppliers supply pencils then producing a list of suppliers would imply that there are other suppliers who do not supply pencils thus misleading the user. A more appropriate response to eliminate this problem, referred to as an *implicature*[37], would be a statement informing the user that all suppliers currently represented in the database satisfy the given restriction.

A related problem the UP project is investigating is that of *misconceptions*[38], which are incorrect beliefs a user might have about the information contained in the database. The objective is to identify the information a system must have in order to identify and rectify user misconceptions. Several types of misconception are identified:

1. *Type Misconceptions* occur when a user specifies a query in which some arguments to a relation are of the wrong type or which includes a relationship that does not exist. An example of such a misconception would be the query to find all courses taught by a person who was not an instructor.

2. *Object-related Misconceptions* have to do with the properties of the objects modelled by the system. Consider the query "Find all employees whose wage is more than $1000 per hour". Instead of simply answering that there are no employees in this category the system should draw the user's attention to the fact that $1000 per hour is outside of the range of possible employee wages. The goal in this respect is to detect this type of error and then offer helpful advice even if the exact nature of the misconception is unknown.

3. *Event-related Misconceptions* happen when a user is mistaken about the necessary order of events which may currently be taking place or may have taken place in the past. For example the query "Is John divorced?" should not respond with a simple "no" if John has never been married. Rather, it should inform the user that John does not satisfy the necessary precondition of having been married.

Supporting Multiple User Views

We stated earlier that user oriented interfaces should be able to maintain *multiple user views* of the data constituting a database in order to support the great variety of user needs and expectations. (For the purpose of this paper, user views are defined as a mapping between the conceptual database and the user interface and *not* as an inter model mapping.)

Most existing database interfaces do not support multiple user views; those that do provide very rudimentary customization facilities. Typically, these are limited to making certain portions of the database invisible to a given user and, in more sophisticated systems, to permit new derived objects and relationships to be included. An ideal system would, in addition, permit the same fragment of information to be perceived in different ways depending on a particular user's point of view. Such versatility, referred to earlier as relativism, requires a polymorphous representation, which goes beyond the capability of present-day systems. Some advocates of the relational model argue that the versatility of the relation *allows* the user to view an object in any way desired. While this is partially true, the problem of relativism is not solved. A model using relations as the only modelling concept is *too amorphous*; it captures little of the application's semantics and, therefore, offers little guidance for interpreting the data.

AI provides a number of techniques for supporting different points of view, as well as contexts and beliefs. Among these are modal logic (where beliefs are treated as static objects), frame based representations (described previously), and partitioned semantic networks. The latter, because of their similarity to user views, show a strong potential for use in the realm of database modelling. Although these techniques seem promising as enhancements to database modelling they do not help solve the view update problem.[9] AI research has traditionally concentrated on designing knowledge representations which, for example, facilitate natural language processing. In this kind of environment new data may be added to the knowledge base but old data is not removed. Because data and contextual information usually share the same data structures new information may be added to a local context without having any global side effects.

Partitioned semantic networks, or K-Nets, are a knowledge representation

scheme developed by Fikes and Hendrix[39] to enhance the expressive power of conventional semantic networks. K-Nets incorporate the capabilities of first order predicate calculus, facilitate linkage to external procedural knowledge, and, most significantly, provide a mechanism that allows subnetworks to be grouped together and referred to as single objects. This grouping of subnetworks is accomplished by introducing the concepts of *spaces* and *vistas*. All nodes and arcs of a K-Net are *elements* of at least one space (also called a partition). Because each space can be referred to as a *single* unit it is possible to specify relationships *between* spaces. *Vistas* are lists of spaces; they are intended to give users a manageable perspective of the information. Any access to the knowledge base is performed through one (or more) of these vistas. The similarity between the concepts of vistas in partitioned semantic networks and the idea of customized user views is obvious. By applying this powerful modelling technique to the database interface, significant gains towards the support of individualized user environments should be realized.

Concluding Remarks

In this paper, we have studied the evolution of data models from the early record oriented systems to the user oriented semantic database models. Each database model examined has made some contribution to the organization, implementation or semantics of database systems. We also observed that significant progress has already been made in increasing the overall semantics of the stored data and providing tools, like high-level query languages and simple user views, which make data manipulation easier for the user. The relatively new trend of adapting AI techniques to database environments seems extremely promising. In the near future we believe that such techniques will permit the construction of information management systems much more sophisticated than those in existence today.

One major requirement on intelligent databases is the ability to use information

about the actual application domain to retrieve intentional data, i.e., data that is not explicitly represented in the database but can be deduced from explicit facts and the knowledge of the application domain. Intelligent databases must also be capable of assisting users with query specification. Future systems based on AI formalisms will include a significant amount of contextual information. This kind of information may be used to help users formulate efficient unambiguous queries, correct users' misconceptions about the structure and content of the data, and provide answers to queries that are practical to use. An answer to a query is meaningful only if the user can actually use it. For example, it may not be helpful to a human user to simply list all data that satisfy a given query if such a list contains thousands or even millions of data items. In order for such information to be useful, future systems will have to be able to assist the user by producing responses to queries in a summarized form and/or to give additional (meta) information about the possible results.

Other areas in which database research can benefit greatly from AI are the use of natural language as a user interface and the definition and maintenance of customized user views of the data. To permit the latter in its full generality, it will be necessary to develop a framework for the support of relativism in which individual user views will evolve independently of each other within a shared database environment. Such user friendly systems will also have to facilitate many different types of user interfaces, ranging from interactive, natural language based, for the novice, to general purpose programming language interfaces, for system developers and maintainers.

The systems and techniques surveyed in this paper show that there is great promise for future application of AI techniques to database technology. It is our hope that this paper will help increase the level of interdisciplinary awareness and

cooperation and that it will spawn new ideas to advance the current state of information management systems.

References

1. *Proc. Workshop on Data Abstraction, Databases and Conceptual Modelling,* M.L. Brodie and S.N. Zilles eds., ACM, June 23–26, 1980.

2. R. Reiter, H. Gallaire, J. King, J. Mylopoulos and B. Webber, "A Panel on AI and Databases," *Proc. Eighth IJCAI,* , 8–12 Aug. 1983, pp. 1199–1206.

3. J. King (Ed.), "Special Issue on AI and Database Research," *ACM SIGART Newsletter*, Vol. , No. 86, Oct. 1983, pp. 32–72.

4. M.L. Brodie, J. Mylopoulos and J.W. Schmidt (Eds.), *On Conceptual Modelling,* Springer-Verlag, 1984.

5. W. Kent, *Data and Reality — Basic Assumptions in Data Processing Reconsidered,* North-Holland, 1978.

6. E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, Vol. 13, No. 6, June 1970, pp. 377–387.

7. W.C. McGee, "The Information Management System IMS/VS," *IBM Systems Journal*, Vol. 16, No. 2, 1977, pp. 82–168.

8. R.W. Taylor and R.L. Frank, "CODASYL Data-base Management Systems," *Computing Surveys*, Vol. 8, No. 1, Mar. 1976, pp. 67–103.

9. L.A. Rowe and K.A. Shoens, "Data Abstraction, Views and Updates in RIGEL," *Proc. SIGMOD Conference*, ACM, 1979, pp. 71–81.

10. P.P. Chen, "The Entity-Relationship Model — Toward a Unified View of Data," *ACM Trans. on Database Systems*, Vol. 1, No. 1, Mar. 1976, pp. 9–36.

11. A. Shoshani, "CABLE: A Language Based on the Entity-Relationship Model," *Lawrence Berkeley Lab.* , Berkeley, CA.

12. L. Bic and R. Hartmann, "Hither Hundreds of Processors in a Database Machine.," *Proc. Fourth Int'l Workshop on Database Machines* , Springer-Verlag, 6–8 Mar. 1985.

13. J.M. Smith and D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Trans. on Database Systems*, Vol. 2, No. 2, June 1977, pp. 105–133.

14. E.F. Codd, "Extending the Database Relational Model to Capture More Meaning," *ACM Trans. on Database Systems*, Vol. 4, No. 4, Dec. 1979, pp. 397–434.

15. Hammer M. and McLeod D.J., "Database Description with SDM: A Semantic Data Model," *ACM Trans. on Database Systems*, Vol. 6, No. 3, Sept. 1982, pp. 351–386.

16. N. Findler (Ed.), *ASSOCIATIVE NETWORKS Representation and Use of Knowledge by Computers*, Academic Press, 1979.

17. N. Roussopoulos and J. Mylopoulos, "Using Semantic Networks for Data Base Management," *Proc. First Int'l Conf. on Very Large Data Bases*, ACM, 1975, pp. 144–172.

18. J. Mylopoulos, P.A. Bernstein and H.K.T. Wong, "A Language Facility for Designing Database-Intensive Applications," *ACM Trans. on Database Systems*, Vol. 5, No. 2, June 1980, pp. 185–207.

19. H. Gallaire and J. Minker (Eds.), *Logic and Data Bases*, Plenum Press, 1978.

20. A. Deliyanni and R.A. Kowalski, "Logic and Semantic Networks," *Comm. ACM*, Vol. 22, No. 2, Mar. 1979, pp. 184–192.

21. L. Bic., "A Data-Driven Model for Parellel Interpretation of Logic Programs," *Int'l. Conf. on Fifth Generation Computer Systems*, Inst. for New Gen. Comp. Tech., 1984, pp. 517–523.

22. L. Bic., "Dataflow Architectures for Knowledge Representation Systems," *Proc. Nat'l. Computer Conf.*, AFIPS, 1985, pp. (in print).

23. J.A. Robinson, "A Machine Oriented Logic based on the Resolution Principle," *Journal of the ACM*, Vol. 12, No. 1, Jan. 1965, pp. 23–41.

24. H. Gallaire, J. Minker and J.-M. Nicolas, "Logic and Databases: A Deductive Approach," *Computing Surveys*, Vol. 16, No. 2, June 1984, pp. 153–185.

25. R. Davis and J. King, "An Overview of Production Systems," *Stanford Artificial Intelligence Laboratory Memo AIM-271*, Stanford University, 1975.

26. C.E. Hewitt, "PLANNER: A Language for Proving Theorems in Robots," *Proc. IJCAI*, , Aug. 1971.

27. M. Hussmann and P. Schefe, "The Design of SWYSS, a Dialogue System for Scene Analysis," *Fachbereich Informatik IfI-HH-B-95/83*, University of Hamburg, Feb. 1983.

28. R. Balzer et al, "Specification-Based Computing Environments," *AAAI-83 Proc. Nat. Conf. on Artificial Intelligence*, AAAI, Aug. 1983.

29. M. Minsky, "A Framework for Representing Knowledge," *The Psychology of Computer Vision* P. Winston (Ed.), McGraw-Hill, 1975.

30. G. Wiederhold, "The Knowledge and Database Management," *IEEE Software*, Vol. 1, No. 1, Jan. 1984, pp. 63–73.

31. J. Leon, D. Memmi, M. Ornato, J. Pomian and G.P. Zarri, "Conversion of a French Surface Expression into its Semantic Representation According to the RESEDA Metalanguage," *Proc. Nineth Int'l Conf. on Computational Linguistics*, North-Holland, 1982.

32. W. Hoeppner, T. Christaller, H. Marburger, K. Morik, B. Nebel, M. O'Leary and W. Wahister, "Beyond Domain Independence: Experience with the development of a German Access System to Highly Diverse Access Systems," *Proc. Eighth IJCAI*, , 8–12 Aug. 1983.

33. E.F. Codd, R.S. Arnold, J.M. Cadiou, C.L. Chang and N. Roussopoulos, "RENDEZVOUS Version 1: An Experimental English-Language Query Formalism System for Casual Users of Relational Data Bases.," *IBM Research Report RJ2144*, San Jose, CA, Jan. 1978.

34. V. Dahl, "On Database Systems Development Through Logic," *ACM Trans. on Database Systems*, Vol. 7, No. 1, Mar. 1982, pp. 102–123.

35. B. Ballard and J. Lusth, "An English-language processing system which "learns" about new domains," *National Computer Conference*, NCC, 1983, pp. 41–46.

36. J. Minker, "An Experimental Relational Data Base System Based on Logic (or Clause Encounters of a Logical Kind)," *Logic and Data Bases.* H. Gallaire and J. Minker (Eds.), Plenum Press, 1978.

37. J. Hirschburg, "Scalar Quantity Implicatures: A strategy for processing scalar utterances," *Department of Computer aand Informaation Science MS-CIS-83-10*, University of Pennsylvania, May 1983.

38. Webber, B.L. and Mays, E., "Varieties of User Misconception: Detection and Correction," *Proc. Eighth IJCAI, Karlsruhe, West Germany*, Aug. 8–13, 1983.

39. R.E. Fikes and G.G Hendrix, "A Network-Based Knowledge Representation and its Natural Deduction System," *Proc. Fifth IJCAI*, , 1977, pp. 235–246.