1993

Design and construction of maintainable knowledge bases through effective use of entity-relationship modeling techniques

William Yancey Pike
bill_pike@bellsouth.net

DESIGN AND CONSTRUCTION OF MAINTAINABLE KNOWLEDGE BASES THROUGH EFFECTIVE USE OF ENTITY-RELATIONSHIP MODELING TECHNIQUES

by

WILLIAM YANCEY PIKE
B.S., University of West Florida, 1987

THESIS

Submitted in partial fulfillment of the requirements
for the degree of
Master of Science
College of Engineering
University of Central Florida
Orlando, Florida

Summer Term
1993

ABSTRACT

The use of an accepted logical database design tool, Entity-Relationship Diagrams (E-RD), is explored as a method by which conceptual and pseudo-conceptual knowledge bases may be designed. Extensions to Peter Chen's classic E-RD method which can model knowledge structures used by knowledge-based applications are explored.

The use of E-RDs to design knowledge bases is proposed as a two-stage process. In the first stage, an E-RD, termed the Essential E-RD, is developed of the realm of the problem or enterprise being modeled. The Essential E-RD is completely independent of any knowledge representation model (KRM) and is intended for the understanding of the underlying conceptual entities and relationships in the domain of interest. The second stage of the proposed design process consists of expanding the Essential E-RD. The resulting E-RD, termed the Implementation E-RD, is a network of E-RD-modeled KRM constructs and will provide a method by which the proper KRM may be chosen and the knowledge base may be maintained. In some cases, the constructs of the Implementation E-RD may be mapped directly to a physical knowledge base.

Using the proposed design tool will aid in both the development of the knowledge base and its maintenance. The need for building maintainable knowledge bases and problems often encountered during knowledge base construction will be explored.

A case study is presented in which this tool is used to design a knowledge base. Problems avoided by the use of this method are highlighted, as are advantages the method presents to the maintenance of the knowledge base. Finally, a critique of the ramifications of this research is presented, as well as needs for future research.

ACKNOWLEDGEMENTS

The author would like to thank his wife, Tracey, for inspiring, motivating, and occassionally forcing him to continue his education. The author offers no small amount of gratitude to Dr. Soheil Khajenoori for displaying tremendous amounts of patience; to Drs. Tom Peeples and Allan Lang for their assistance and support; and to his parents, Billy and Eola Pike, for their love and guidance through the years. Finally, but most importantly, the author would like to thank the Lord, without whose supreme assistance this work would be impossible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Although the topic of knowledge base and database integration has recently been an area of considerable research from both from academia and industry, for the most part this research has failed to integrate conceptual database design principles into the design of knowledge bases.

The need for such a design methodology in the knowledge base system world is inarguably a real one. The design of knowledge-based systems (KBS) and their underlying knowledge base management systems (KBMS) suffers from a lack of a de facto standard methodology (**Gonzalez and Dankel 1993**). This lack of a methodology can lead to a *paradigm shift*, in which, during the development of the KBS, the developer must shift to a new technology. (**Gonzalez and Dankel 1993**) This paradigm shift is caused when the initial selection of knowledge representation model (KRM) can not adequately perform its intended function. This represents perhaps the most serious problem in KBS development. However, there are other inherent problems, as described in (**Gonzalez and Dankel 1993**). One problem lies in the difference between solving traditional information-system problems and heuristic-oriented problems. The data needed for algorithmic problems can be determined fairly easily, while in the case of knowledge-based systems, sometimes the "nature and quantity" of the knowledge isn't known even by the experts. The process of knowledge acquisition can thus prove to be fairly frustrating. One of the underlying reasons for this, claims Earl Cox, a columnist for *AI Expert*, is a common perception that AI is commonly defined in "terms of ever more advanced knowledge representation schemes devoid and divorced from fundamental architectural and design considerations." **(Cox 1993)** The lack of any recognized correlation between AI and conventional systems has lead to "confusion in aims and directions" of AI in the marketplace **(Cox 1993)**. Clearly, there is a need to apply sound, established traditional software development principles to AI system development.

The attitudes and mindsets of KBS developers are perhaps part of the problem. The roots of database research lie primarily in the "commercial sector's need for efficient and secure data processing systems." **(Jelly and Gray 1992)** Free from this requirement which would restrict research to mostly commercial applications, early KBS researchers developed an almost "renegade" approach to application development. Indeed, as Cox has pointed out, "there does seem to be a general consensus among knowledge engineers that AI is somehow completely removed from computer science, systems design, and functional decomposition." **(Cox 1993)**

Another viewpoint of this problem is stated in **(Cohen 1990)**. Paul Cohen blames much of the problem on the lack of qualitative vice quantitative research in AI. He states, "Much work is unevaluated and most evaluations are limited to measures of performance. System design appears arbitrary and, when justifications do appear, they are informal...Evaluation tends to be limited to performance evaluation, instead of tests of hypotheses of how behavior arises from the interaction of agents' architectures and their environments." Cohen goes on to describe what he terms the "strip mining" view of AI research. "AI researchers trash the space of questions about intelligence in much the same way that slash-and-burn cultures trash the rain forest. Both make very inefficient use of resources." As an example of "strip mining," Cohen points out the following: "The statement 'X is sufficient to produce Y' alleges but does not model or explain the alleged causal relationship between X and Y . . . Demonstrating that X is sufficient to produce Y does not show that X is a particularly good way to produce Y, or that X is necessary to produce Y." This problem is very similar to the "nature and quantity" dilemma discussed above.

Quality has become somewhat of a buzzword in industry (e.g., "Total Quality Management".) As knowledge-based systems in specific and AI systems in general come out of the research lab and into the mainstream of the marketplace, the quality of these systems must be taken into consideration. It is pointed out in **(Fenn and Veren 1991)** that "adherence to a software engineering methodology and development lifecycle can significantly improve the quality of a completed system." Earl Cox has stated that "successful AI projects combine quality with concerns for economical solutions." **(Cox 1993)**

As the maintenance portion of any software project life cycle has historically been the costliest, a design technique should provide for maintenance in order to supply quality to the project **(Ignizio 1991)**, **(Parsaye and Chignell 1988)**, **(Debenham 1992)**.

To a great extent, these same problems or similar concerns can be seen in traditional database application development efforts. Semantic data models have been used as a design tool to solve these problems. Of all the semantic data models, Peter Chen's entity-relationship (E-R) model has become the most popular, due to a great extent to the popularity of the E-R diagram (E-RD), a graphical companion to the E-R model. **(Date 1990)**

Applying Peter Chen's classic E-R diagramming technique, or some variation thereof, to the design of a knowledge base (regardless of the knowledge representation technique used by the KBS) provides the developer with a proven methodology to ensure a more intelligent design. By developing E-RDs early in the development life-cycle of the KBS, designers can avoid the knowledge representation paradigm shift by determining the proper representation *a priori* implementation. Having a well-defined E-RD of the knowledge base can also aid in maintaining the KBS. The effects of adding new knowledge or modifying existing knowledge can quickly be determined by consulting an E-RD.

This thesis proposes the use of entity-relationship diagrams as a design tool for the development of knowledge base systems. More specifically, a two-stage process is proposed in which a traditional E-RD, called the Essential E-RD, is developed based on the conceptual knowledge base as the first stage. This E-RD serves to identify the conceptual entities and relationships of the knowledge realm. In the second stage, the first E-RD is expanded to model the knowledge structures via extended E-RD structures. The resulting E-RD is called the Implementation E-RD. These E-RD structures, for the most part, have already been proposed in earlier bodies of research as development aids for DBS applications, although additional structures are proposed herein to better model knowledge concepts.

The use of a semantic data model is defended as a combination of the latter two levels of the three-level of integration of databases and knowledge bases. The first level, the physical layer, involves utilization of database management systems (DBMSes) to physically store the knowledge of a KBS, and the integration of traditionally KBS-oriented features into DBMSes. The second level, termed the pseudo-conceptual layer, starts to apply conceptual DBMS design methodologies into the design of a knowledge base. In this layer, the design is presented for a certain KRM only. At the conceptual layer, database design techniques are proposed for the design of the conceptual knowledge base, independently of a specific KRM.

As knowledge bases continue to grow, they will undoubtedly require a great deal of support from databases. Many expert-system shells now offer front-ends to popular database engines. Likewise, as database applications become more complex, they will require intelligent features from knowledge based systems. An example of this is ongoing research in the database community of implementing *business rules* into databases and database applications. These rules, defined as "constraint(s) placed upon the business" (**Moriarty 1993**), have a five-stage design process very similar to the design process of KBSes. A reason expert systems fail is that they aren't integrated into the corporate computing architecture. "A high percentage of expert system programs result in a successful prototype from a technical point of view but fail to produce a system which is integrated into an organization's mainstream operational environment." (**Fenn and Veren 1991**). The corollary of this statement may also well be true; that is, the knowledge bases of intelligent systems are not being utilized by the "mainstream" corporate applications. This work serves as an important step in bringing the two camps together.

LITERATURE REVIEW

Research for this thesis was necessarily performed from two separate but complimentary viewpoints. Experts from both the database realm and the knowledge base realm have written extensively on issues similar to the ideas proposed herein.

Databases and knowledge bases share many similarities. Both serve to store the data necessary to make their respective systems perform. Both have established physical structures designed to optimize the retrieval of that data. Both have certain relationships between their logical design and their physical design. The union of databases and knowledge bases can be divided into three levels: the *physical* level, the *pseudo-conceptual* level, and the *conceptual* level. While the conceptual and the pseudo-conceptual levels are the primary concern of this paper, a review of all three levels will help establish a better baseline for the main premise to be presented later.

The Physical Level

The physical level represents the lowest level of abstraction in the integration of databases and knowledge bases. At the physical level, research has focused on many areas. Those areas discussed here will consist of:

- Storing/retrieving knowledge in/from a database management system (DBMS),
- Adding traditional expert system features to DBMSes, and
- Interfacing database systems (DBSes) and KBSes.

Frank Anger, Rita Rodriguez and Douglas Dankel have co-authored a series of papers on organizing expert systems' knowledge bases using databases and database design techniques. They have proposed utilizing a commercial relational DBMS

(RDBMS) to store the rules of an expert system's knowledge base **(Rodriguez et. al. 1989)**. Their proposal calls for three RDBMS relations, or tables, to implement the knowledge base. The first, named **IF**, consists of the fields *rule#*, *assrt#*, and *assrtdescr*. The second table, **THEN**, is also made up of the fields *rule#*, *assrt#*, and *assrtdescr*. To track confidence, an integral part of rule-based systems, the table **RULE-CONF** is defined to consist of the fields *rule#* and *conf*. These fields are described as such:

rule# - a unique identifier of the rule
assrt# - a unique identifier of an assertion
assrtdescr - the textual description of the assertion
conf - a number which represents the confidence in the deduction

In this design, both the **IF** and **THEN** tables have a composite primary key consisting of the fields *rule#* and *assrt#*, while **RULE-CONF** uses *rule#* as its primary key **(Rodriguez et. al. 1989)**.

The same paper also details the addition of procedural knowledge via a *trigger* relation. This table, called **TRIGGERS**, includes as a foreign key the field *assrt#*. When the inference engine fires a rule which involves assertion N, the system queries **TRIGGERS** to determine whether any procedures are to be invoked. An additional table, **PROCEDURES** (whose primary key *pname* is also a foreign key in **TRIGGERS**), contains the action to be performed **(Rodriguez et. al. 1989)** **(Anger et. al. 1988)**.

An additional step in this direction has been proposed in **(Ito 1991)**. Ito proposes a coupling of KBSes and DBMSes. Since the reconstruction of an existing database to perform the task of knowledge base manager is "burdensome", Ito suggests the knowledge representation scheme (KRS) provide the mechanisms required for coupling. Called IKD (Interface for integrating a Knowledge-based system and a Database system), the system serves as the interface between a KRS called KBUS and a relational database.

KBUS is composed of a frame-based system called FKBUS and a production system called PKBUS, in addition to IKD. FKBUS consists of several frames and sub-frames which include, among other items, actual SQL (Structured Query Language) code to retrieve knowledge from the database. In summary, Ito's paper proposes a knowledge-based system which uses a frame-based subsystem to retrieve knowledge from an SQL-compliant relational database.

Levent Orman of Cornell University proposed in **(Orman 1992)** that a three-layer abstraction ("external", "conceptual" and "internal" layers) of knowledge bases be developed, with each layer targeted to a specific user type. At what Orman calls the "internal level", targeted to system implementers, rules are to be "viewed as data." An interesting point of Orman's proposal is the case he makes for hierarchical databases to store rules, as opposed to the relational database approach championed in (**Anger et. al. 1988**), **(Rodriguez et. al. 1989)** and (**Ito 1991**). As a discussion of which database model is most suited for the storage, retrieval and management of knowledge constructs is beyond the scope of this paper, the point is simply made that **(Orman 1992)** provides a strong case for the physical level of abstraction of database/knowledge base integration.

Industry has also contributed to the physical level of DBMS/KBS unions. Many relational databases now supply *triggers*, which supply a primitive method of supplying rule-based processing. A trigger is defined to be invoked on a certain action or condition (cf., *trigger relations*, **(Rodriguez et. al. 1989)** (**Anger et. al. 1988**)). Unfortunately, triggers generally must be written in SQL, which doesn't provide the flexibility required to add true intelligence to a database. Sybase, Inc., an innovator in client-server RDBMS engines, has included the capability for "stored procedures" which can add a further level of intelligence to a database by defining certain processing to occur based on user-defined events. These stored procedures, which are compiled and execute on the server side of database applications, allow more efficient processing than triggers. The influx of client-server database engines has provided another opportunity for DBMS/KBS unions. A query can be passed through a KBS on the client side before issuing the SQL code to the server side. Ingress, the relational DBMS from Ask Computer Systems, has improved this process by supplying a knowledge management module as an add-on. This module allows for the incorporation of rules into applications which use the database **(Jenkins and Grygo 1991)**.

The Pseudo-Conceptual Level

The layer of abstraction referred to here as "pseudo-conceptual" is somewhat harder to define. In this work, the pseudo-conceptual layer will refer to a level of integration of knowledge base design and database design in which one particular knowledge representation scheme is modeled via traditional logical database design techniques. At this level, the semantic model becomes of more importance than the syntactic model.

In addition to the physical layer examined above, both **(Anger et. al. 1988)** and **(Rodriguez et. al. 1989)** contain a certain amount of work in the pseudo-conceptual layer. KBS developers can use E-R diagrams to model rule bases in much the same way as databases are modeled. More specifically, their proposal states that "simple assertions of the rule base are viewed as one entity type and the rules as another, with *IF* and *THEN* being relationships between these types." **(Rodriguez et. al. 1989)** Using this method will capture "the information contained within the rules." **(Rodriguez et. al. 1989)**

At Orman's "external level", targeted to end-users of KBSes, rules are depicted graphically **(Orman 1992)**. Orman proposes the use of labeled arcs to represent relationships between data items represented by points. Cardinality concepts (e.g., SOME, UNIQUE, EACH) are given graphical constructs as well. As in the previous references, though, the graphical representations are limited to applications to rules, thus fitting the definition of the pseudo-conceptual level.

The differences between the physical and the pseudo-conceptual layers cited in the same works are significant. The first set of references to **(Anger et. al. 1988)**, **(Rodriguez et. al. 1989)** and **(Orman 1992)** examined the proposal to take actual rules and store them in a database. In the second set of references to these same three papers, emphasis is placed on taking an existing knowledge base (in all three cases, a rule base) and modeling its semantics via some graphical methodology. Thus, it is the pseudo-conceptual level of database/knowledge base integration at which one can first see an attempt to integrate semantic principles of the two techniques.

The Conceptual Level

At the level of abstraction of KBS-DBS integration referred to as the conceptual level, the particular inferencing technique becomes of secondary importance to the conceptual knowledge schema, in much the same way as the physical database model is of less importance than the logical database schema during the logical design phase of database design. Although previous work has failed to hone in on this level to the extent it has the other two levels, recent literature has seen a trend of research on this level. One example is **(Mattos 1989)**, in which semantic data models and knowledge representation models are characterized as being composed of several abstraction concepts, including classification, generalization, inheritance, element and set association, and element and component aggregation. Mattos further argues that each of these main concepts (classification, generalization, association and aggregation) has inherent reasoning facilities. Additionally, **(Debenham 1992)** presents an argument for building a "maintainable" knowledge base around Horn clause logic (essentially, a rule-based system) which would, by definition, place his methodology at the pseudo-conceptual level. However, he does defend his approach as being independent of KRS by pointing out that "as long as the knowledge has been modeled rigorously and...this model of the knowledge has been normalized," it "really doesn't matter what language is used to actually implement the knowledge." **(Debenham 1992)** In **(Feldman and Fitzgerald 1985)** the point is made that, while knowledge based systems represent a newer discipline than more traditional information systems, both share common problems in the area of "knowledge representation and acquisition", more than in "technical aspects of programming methods." This common area of concern clearly points to a high level of abstraction in the marriage of the two areas.

Again, the difference between the conceptual level and the pseudo-conceptual level is significant: at this higher level of abstraction, any restriction on inference technique is removed, and the problem becomes one of actually modeling a conceptual base of knowledge with a semantic data model. In **(Borgida 1991)**, the point is made that in the database world, more emphasis is placed on "modeling the human conceptualization" of the knowledge domain, while the knowledge base world has just now begun to investigate modeling the conceptual schema vice "modeling the physical storage structures."

In summary, previous examinations of the union of DBSes and KBSes can be separated into three layers of abstraction: physical, pseudo-conceptual and conceptual. The physical layer is the layer at which databases are used to physically manage knowledge, and at which intelligent features are added to DBMSes. The pseudo-conceptual layer begins to examine the use of database design techniques, but generally limits their use to one specific KRM. The most abstract layer, the conceptual layer, suggests the use of database design techniques for any and all KRMs. A combination of the pseudo-conceptual and conceptual layers will provide the basis for the proposal of this thesis.

Divergence from the KADS Methodology

The KADS methodology has become the most notable KBS design methodology since its origin in 1983 as an ESPIRIT project. Many of the same concerns expressed in this work are also expressed in **(Hickman et. al. 1989)**, which is probably the definitive English-language text on the methodology. One such concern is based on the traditional KBS development method, that of rapid prototyping. "(Rapid prototyping) provides very little in the way of support for management issues, which are crucial to successful project control." The authors go on to point to the "deliberate confusion between process and data" as a deficiency in conventional software development methodologies for KBS development. The text claims that entity modeling is not appropriate for KBS development because the process of assigning entities to the real world problem is difficult and the process of assigning attributes to those entities is "very difficult indeed." One item that truly separates the KADS methodology and the other references cited here is that the KADS methodology makes no attempt to integrate knowledge bases and databases, nor does it attempt to separate the knowledge base from the KBS at the logical level.

Knowledge Base Maintenance Using Database Techniques

Additionally, recent research has centered on the area of knowledge base maintenance, and how database design principles can assist. The importance of normalizing knowledge and applying constraints, including the referential integrity constraint, has been discussed in **(Debenham 1992)**. (The concepts will be discussed in detail later.) Debenham's work presents three models: the data model, the information model, and the knowledge model. Basically, the data model is based on the real world realization of the problem, and corresponds roughly to a semantic data model. The information model is analogous to the metadata of a relational database schema, while the knowledge model consists of details about the knowledge representation structure. The data model drives the information model, which in turn drives the knowledge model. Debenham suggests normalization be performed at the data model as it is the easiest to normalize. In addition, non-normalized entities at the data model level can cause a "proliferation" of non-normalized entities at the higher levels. Knowledge base maintenance becomes more manageable with a normalized model, Debenham argues, since all inter-relationships between the component items can be determined more quickly. In a similar manner, Debenham defends applying constraints to the knowledge base (on all three models) as a means to ensure efficiency in the maintenance process.

KNOWLEDGE REPRESENTATION

Just as there are several database models (e.g., relational, network, hierarchical), likewise are there several different knowledge representation models. The most common knowledge representation models are

- Rules
- Frames
- Semantic, or Associative, Networks
- Object Orientation

The inclusion of object orientation as a knowledge representation model could be somewhat debatable; however, when examined at the very basic level, one can see similarities between a frame-representation scheme and an object oriented approach. In addition, object orientation is seen as a means by which intelligence can be added to databases; thus it is included herein as a separate model. Each of these models will be examined in detail to determine what features a modeling tool must provide in order to model their structures.

Rule-based systems are the most commonly known of all KRMs. A rule consists of two parts, a *premise* and a *conclusion*. Rules are generally expressed either as an "IF-THEN" relationship (e.g., **IF** it is August, **THEN** we will have a thunderstorm) or vice versa (We will have a thunderstorm **IF** it is August.) Any number of ANDs, ORs or NOTs can be appended to the premise (**IF** it is August **AND** we are in Central Florida **OR NOT** (I have mowed my yard), **THEN** we will have a thunderstorm.)

A frame-based system collects related knowledge into sets of attribute-value (or slot-filler) pairs called *frames*. The fillers are often subdivided into facets, each of which has its own value (**Gonzalez and Dankel 1993**). Facets may include *range*, *default value*, and *daemons*, procedures which execute upon a pre-defined condition. Frames are ordered in the knowledge base into a hierarchy with IS-A links between the nodes (**Hodgson 1991**). Inheritance plays a major role in frame-based systems as children frames tend to inherit values from parent frames. Using the structure set forth in (**Gonzalez and Dankel 1993**), a frame detailing storm types could be depicted as:

```
Generic STORM Frame
        Specialization-of: WEATHER
        Generalization-of: (THUNDERSTORM, HAILSTORM, SNOWSTORM)
        Precipitation:
                Range: (NONE, RAIN, ICE, SLEET, SNOW, HAIL)
                Default: (RAIN)
        Wind-Speed:
                Range: (0-150)
        Warning-Type:
                Range: (NONE, WATCH, WARNING)
                If-Needed: (WATCH-WEATHER-CHANNEL)
                If-Modified: (ALERT-MEDIA)
        Lightning-Presence:
                Range: (NONE, LIGHT, MEDIUM, HEAVY)
                If-Modified: (CHECK-FOR-THUNDERSTORM)
```

This example illustrates classification (*Specialization-of* and *Generalization-of*), from which inheritance generally arises, ranges and defaults, and daemons (*If-Modified*, *If-Needed*). The STORM frame will inherit properties of the WEATHER frame, while THUNDERSTORM, HAILSTORM and SNOWSTORM will inherit properties of the STORM frame.

Associative networks, originally termed semantic networks, were developed to represent knowledge in natural language sentences. Their use has grown beyond semantics to encompass physical and causal associations (**Gonzalez and Dankel 1993**).

Associative networks are basically directed graphs whose nodes represent concepts and whose links represent associations between the concepts. These associations can take on many different meanings; classification (instance-of), generalization (is-a) and aggregation (part-of) are three of the more common and important association types (**Mattos 1991**).

Object-orientation (OO) can arguably be presented as a knowledge representation scheme. Its inclusion here is an acknowledgment of the capability of OO to add intelligence to databases. The world of objects has grown to include object-oriented *programming* (OOP), object-oriented *analysis* and *design* (OOA and OOD), and object-oriented *databases management systems* (OODBMS). While each of the three has its own features which are not crucial to this thesis (e.g., the concept of dynamic binding in OOP), all object-oriented approaches share common features, including inheritance, polymorphism and encapsulation. Inheritance in OO is identical to inheritance in frame-based systems. Polymorphism is similar to the concept of generalization. Encapsulation, perhaps the cornerstone of the object world is the process by which data structures and the processes performed upon them (methods) are encapsulated, or combined, into one entity, called a package, class or object type.

In summary, a design methodology for KBSes must meet the requirements of several different knowledge representation schemes. These schemes utilize the following features:

- If-Then relationships between premises and conclusions
- Inheritance
- Generalization/Specialization
- Classification
- Aggregation
- Encapsulation

E-R DIAGRAMS AS A KBS/KBMS DESIGN TOOL

So far, this paper has established the need for a structured design methodology for knowledge-based systems, presented arguments for the integration of knowledge-base and database systems, and examined various knowledge representation models. Building on previous work on the integration of KBSes and DBSes, this chapter will present a design methodology for KBSes which will satisfy the needs of the various KRMs and overcome common problems inherent with KBS design and development. It is the primary intent of this thesis to introduce the use of E-R diagramming as a knowledge base system design tool, and to defend its use by presenting its advantages to various stages of the knowledge base lifecycle.

Semantic modeling has been defined as "the overall activity of attempting to represent meaning." **(Date 1990)** This definition compliments the view of a KRM as a scheme to represent knowledge. It has been argued in **(Borgida 1991)** that semantic data models and KRMs share many similarities, while their differences tend to revolve around the "differing goals to which they subscribe." These similarities include:

- **Object Identity** - both KRMs and semantic models subscribe to the notion that an instance of knowledge or data has its own identity independent of its attribute values or participating relationships.
- **Binary Relationships among Objects** - both support binary (vs. n-ary) relationships among objects (e.g., attributes, slots, properties).
- **Grouping of Individuals into Classes** - Chapter II discussed generalization; the concepts of grouping individuals into classes and generalization are practically identical.

- **Decomposition of Classes into Subclasses** - Chapter II discussed specialization; the concepts of decomposition of classes into subclasses and specialization are practically identical.
- **Constraints** - Both KRMs and semantic models provide means of expressing conditions of validity for attributes.
- **Derived Classes/Relationships** - KRMs and semantic models both have methods defined to control redundant information and enforce its consistency. **(Borgida 1991)**

Drawing upon this list, it is safe to say there is a definite parallel between knowledge representation and semantic modeling. For this reason, this chapter will promote the concept of semantically modeling the knowledge of a KBS as a design aid for KBS development.

Peter Chen's classic entity-relationship modeling and diagramming technique **(Chen 1976)** is arguably the de facto standard for database design in general, and relational database design in particular. As databases have become more intelligent in nature, so too have E-R modeling and diagramming techniques been extended to help developers better keep track of the inherent intelligence of the database. This research will demonstrate how the classic E-R diagram, with extensions, can adequately model the knowledge base of any KBS, regardless of knowledge representation scheme. It will also bring to light some advantages of performing this modeling.

In the decade and a half since Chen presented his very valuable tool, the E-RD methodology has undergone many adaptations. Researchers have proposed extensions to the original model to allow it to model many different types of data and knowledge. The proceedings of the annual Entity-Relationship Approach conferences provide a wealth of new E-RD extensions. There are object-oriented E-RDs (**Navathe and Pillalamarri 1989**), action-modeling E-RDs (**Feldman and Fitzgerald 1985**), and E-RDs which model both transactional information and conceptual knowledge (**Lazimy 1988**), to name but a few. Elements of many of these "E-RD flavors" will be selected to develop a case for this paper's proposal: entity-relationship modeling and (in particular) diagramming can be used to model the conceptual knowledge base of a knowledge-based system in much the same way as they presently model the logical database of a traditional information system.

Entity-Relationship Basics

A short review of basic E-R modeling reveals three main concepts: *entities*, *attributes* and *relationships*. Peter Chen, who originated both the concept of the entity-relationship model and its graphical partner, the entity-relationship diagram, defines an entity as "a thing which can be distinctly identified." An attribute is a piece of information that describes an entity. Finally, a relationship is defined as "an association among entities." (**Chen 1976**) An example to illustrate these basics is that of a personnel system. The entities of concern are ***EMPLOYEE*** and ***OFFICE***. In this example, employees are assumed to work for one and only one office. The attributes are as follows:

Table 1

EMPLOYEE and OFFICE entities and attributes

EMPLOYEE	**OFFICE**
EMPLOYEE_ID	**ORGANIZATION_CODE**
EMPLOYEE_NAME	**OFFICE_TITLE**
JOB_CLASS_CODE	**MANAGER_ID**
DATE_REPORTED	

Figure 1 shows this example in E-R diagram form.

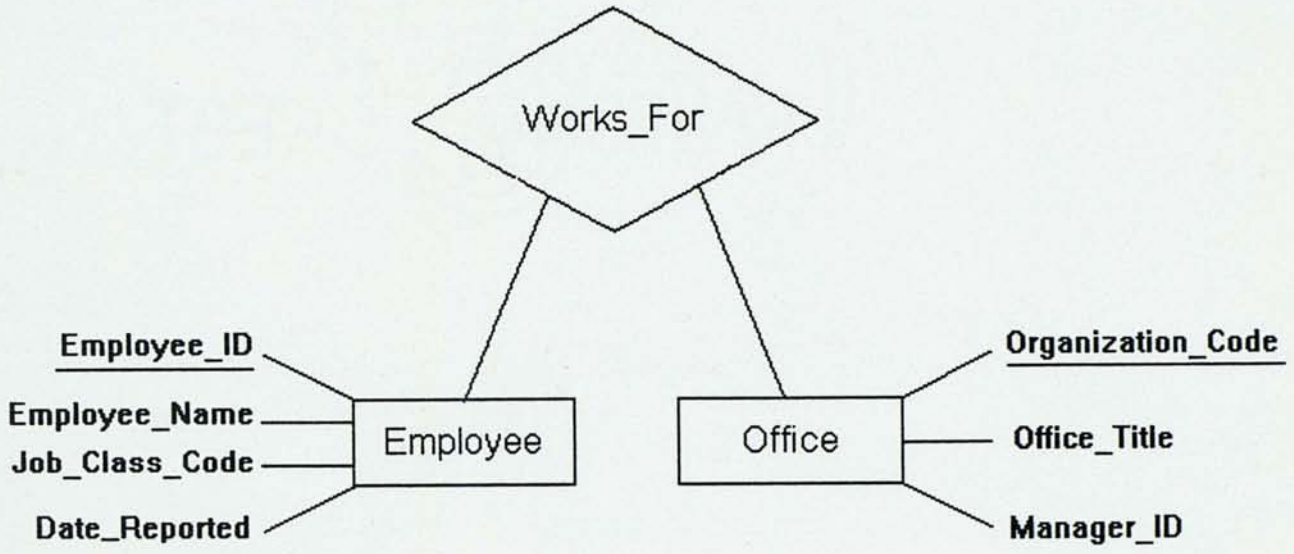

Figure 1 - Sample E-R Diagram

The underlined attributes (**EMPLOYEE_ID**, **ORGANIZATION_CODE**) represent the primary keys of their respective entities. The cardinality of the relationship between the entities is denoted by the "**M**" and the "**1**"; in this example, there is a one-to-many relationship between offices and employees. Although Chen introduced several other features in his essay, these features constitute the bulk of E-RD basics. Appendix A presents a more detailed review of E-R concepts.

Knowledge-modeling Extensions to E-RDs

The first requirement of a knowledge-modeling tool is to provide a model for if-then rules between premises and conclusions. In **(Rodriguez et. al. 1989)**, the following diagram is given as an example of how this can be accomplished with standard E-RDs.

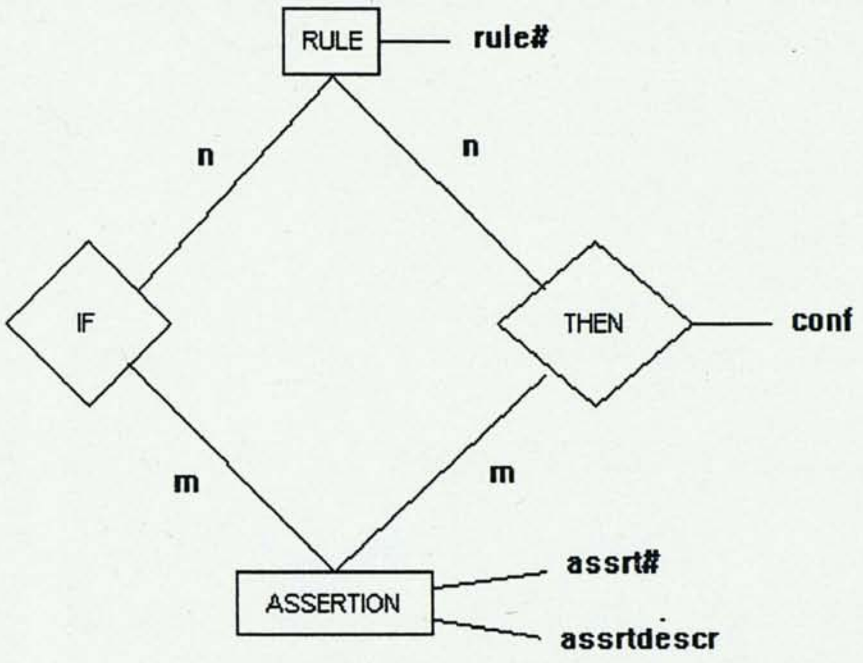

Figure 2 - If-Then E-RD from (Rodriguez et. al. 1989)

This E-RD depicts a many-to-many relationship occurring between the entity **RULE** and the entity **ASSERTION.** This approach differs from more traditional E-R modeling by viewing the rule base as the real world. In traditional database applications, the subset of the real world involved in the problem is modeled as the real world.

A more conceptually-oriented approach to semantically model rules is discussed in **(Feldman and Fitzgerald 1985)**. In that work, the use of "action modeling" is presented. They propose this action model to be "constructed in analysis after an entity model has been built," a two-stage approach to knowledge base design similar to the approach espoused in this work. The fact that some sort of behavior modeling must be provided in order to successfully model a rule excludes the static structure of the entity-relationship model; however, rules do perform their actions on entities, thus some method of depicting them must be provided.

A rule can be considered as an action which occurs as the result of some state of a relationship between one or more entities. As such, a rule should be considered to be an attribute of that relationship. If the rule applies to only one entity, a weak entity and relationship may be created, although this adds an unnecessary step. In this case, the rule may be depicted as an attribute of the given entity. The term "attribute" as used here should not be readily compared to an attribute in a typical database E-RD. Attributes in database E-RDs will become domains, fields or columns in the physical database, while an attribute depicting a rule will see a different mapping in the physical knowledge base. This attribute should be some sort of implementation-independent description of the rule (a "pseudo-rule", comparable to pseudo-code.) The pseudo-rule should either be attached to the relationship as written or identified by a unique identifier and written out elsewhere. This ensures that the relationship between the rule and the entity(ies) the rule references can be determined quickly by visual inspection of the E-RD.

Figure 3 depicts a rule in an E-RD.

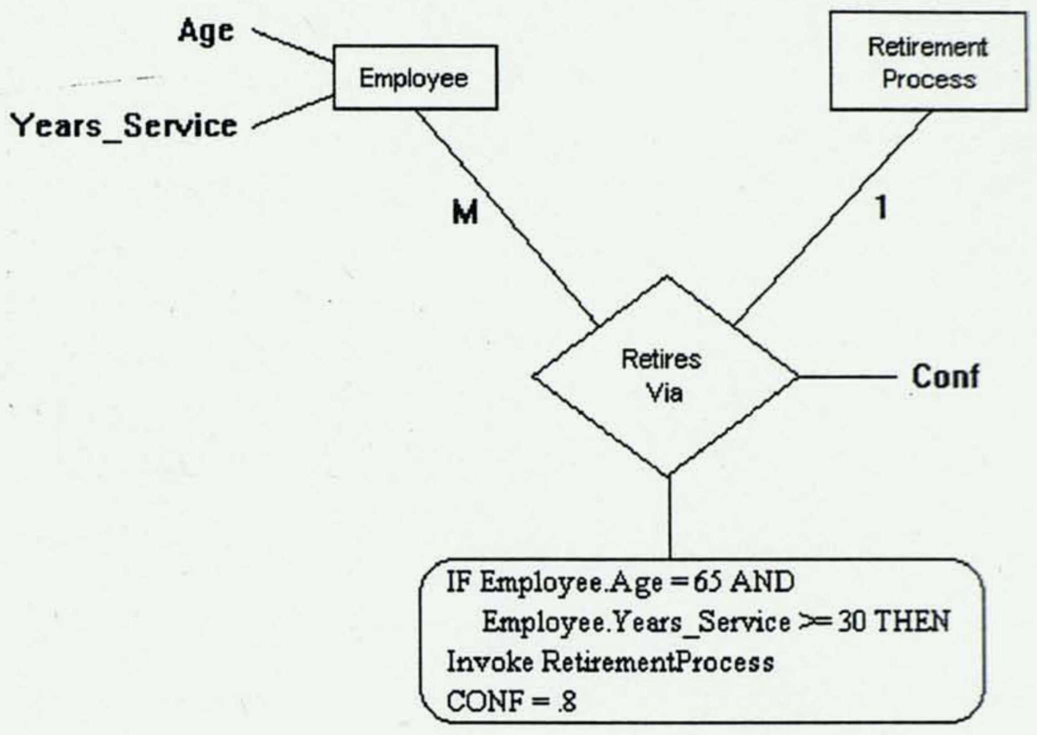

Figure 3 - Depicting If-Then (Rule) relationship in E-RD

The point is made in (**Debenham 1992**) that rules do not always follow the traditional "if-then" format of Figure 2. A semantic model should thus not be limited to if-then relationships simply because the underlying KRM is a rule. However, providing a single diagramming construct to capture all possible rule relationships isn't practical. The method illustrated above allows the designer flexibility in establishing rules.

Classification, generalization, specialization and inheritance all rely on sub- and super-classes. These classes represent a hierarchy from the general (superclass) to the specific (subclass). An entity type which is defined as a superclass will, in an E-RD, be connected to its subclass with a triangle. Multiple subclass entity types each connect to the triangle, which then connects to the superclass entity type. Figure 4 presents an example in which the EMPLOYEE superclass consists of ENGINEER, SECRETARY and SUPERVISOR subclasses.

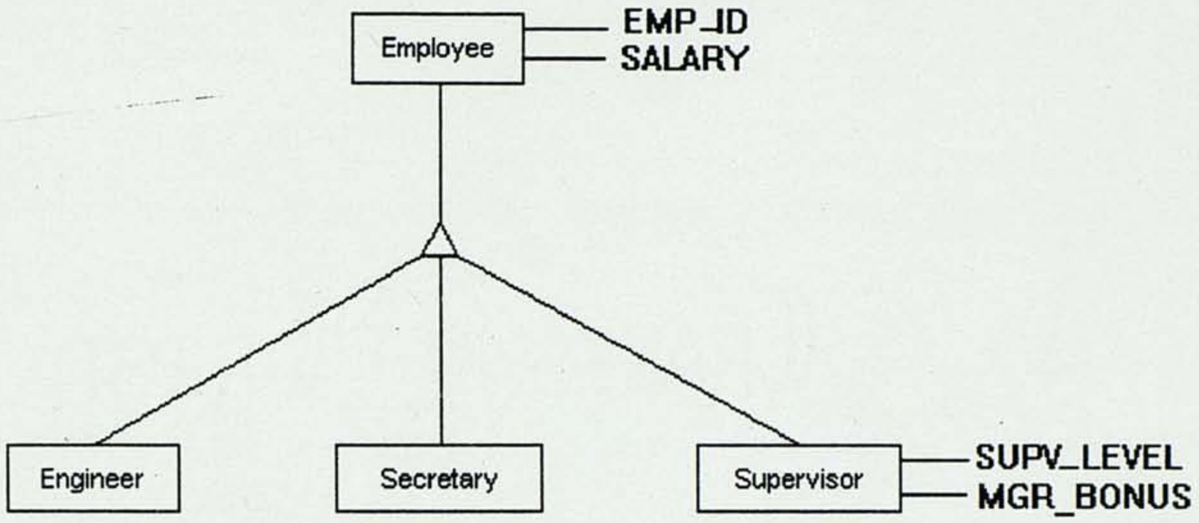

Figure 4 - E-RD illustrating subclasses, superclasses

The presence of a subclass symbol (triangle) represents subclasses; a subclass is assumed to inherit any and all attributes from its parent superclass. Sibling subclasses are not assumed to share additionally defined attributes; if two or more subclass entity types are to share an attribute, that attribute must be explicitly assigned to each entity type. Thus, in Figure 4, all three subclass entity types inherit the attributes EMP_ID and SALARY, while only the SUPERVISOR entity type has SUPV_LEVEL and MGR_BONUS defined.

Generalization and specialization are complimentary concepts, with specialization defined as "the process of defining a *set of subclasses* of an entity type." **(Elmasri and Navathe 1989)** The process of specialization produces subclasses; likewise, generalization produces superclasses. There are several constraints on generalization and specialization which show up in the extended entity-relationship (EER) diagrams defined in **(Elmasri and Navathe 1989)**. These include:

- Predicate definition
- Disjointness

- Completeness

Predicate definition refers to the method by which membership in subclasses is determined. The attribute-value condition is called the **defining predicate**; all entities in the superclass which meet the defining predicate condition belong to a certain subclass. If all the subclasses in a particular specialization are defined to have the same predicate (i.e., the same attribute is used to determine the membership constraint for each subclass), the specialization is called an **attribute-defined specialization**. When the defining condition is not the same across all members of the specialization (i.e., different attribute-value pairs are used to determine the membership constraint for subclasses), the subclass is considered **user-defined**. For an attribute-defined specialization, the defining attribute of the superclass is listed on the line between the superclass entity type and the superclass-denoting triangle, and the values are listed on the line between the triangle and the respective subclass entity types.

The second constraint defines to how many subclasses of the specialization an entity type can belong. If an entity type can belong to no more than one subclass, the specialization is called **disjoint**. If an entity type may belong to more than one subclass, the specialization is considered to **overlap**. Disjoint specializations are denoted by a "d" in the triangle; specializations which overlap have an "o" in the triangle.

The final constraint is called the **completeness constraint**. A specialization may be either a **total** or a **partial** specialization. In a total specialization, every entity type in the superclass must belong to a subclass. All entity types need not be a member of a subclass in a partial specialization. A total specialization has a double line connecting the superclass entity type and the triangle, while a partial specialization has a single line.

Figure 5 illustrates the concepts of generalization and specialization in an E-RD. The diagram tells that the specialization is total (the double line from EMPLOYEE to the triangle) and overlaps (the "o" inside the triangle). Furthermore, EMP_TYPE_CODE is the defining attribute of the attribute-defined specialization. Defining predicates are EMP_TYPE_CODE = E for the subclass ENGINEER, EMP_TYPE_CODE = S for the subclass SECRETARY, and EMP_TYPE_CODE = V for the SUPERVISOR subclass .

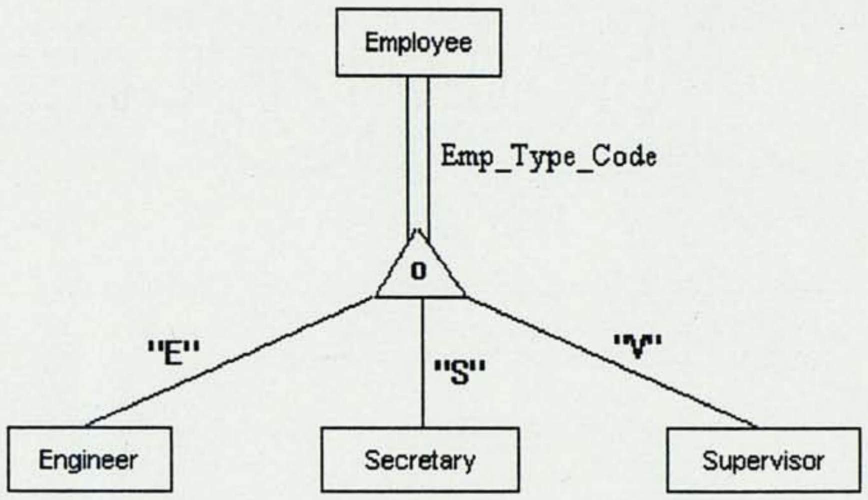

Figure 5 - E-RD illustrating generalization/specialization

In a knowledge-based system, the defining attribute may be a rule. In this case, the rule should be shown as an attribute of the classification triangle.

Aggregation can be represented quite easily in an E-RD. If an object class is defined as an aggregate of multiple entities and one or more relationships, that class can be diagrammed as a single entity in an E-RD. The aggregate entity must be labeled as such, and the components must be so noted. Figure 6 shows an example of the aggregation of entities and relationships to produce the Retirement Process entity of Figure 3. The box surrounding the constituent entities and relationships is in bold to show that it is an aggregate entity. If an entity which is not part of the aggregate entity must be shown in the same box, it should be shown to be separate by double vertical lines. In this case, CorporateHeadquarters is not part of the aggregate entity RetirementProcess, but is shown in the box for clarity.

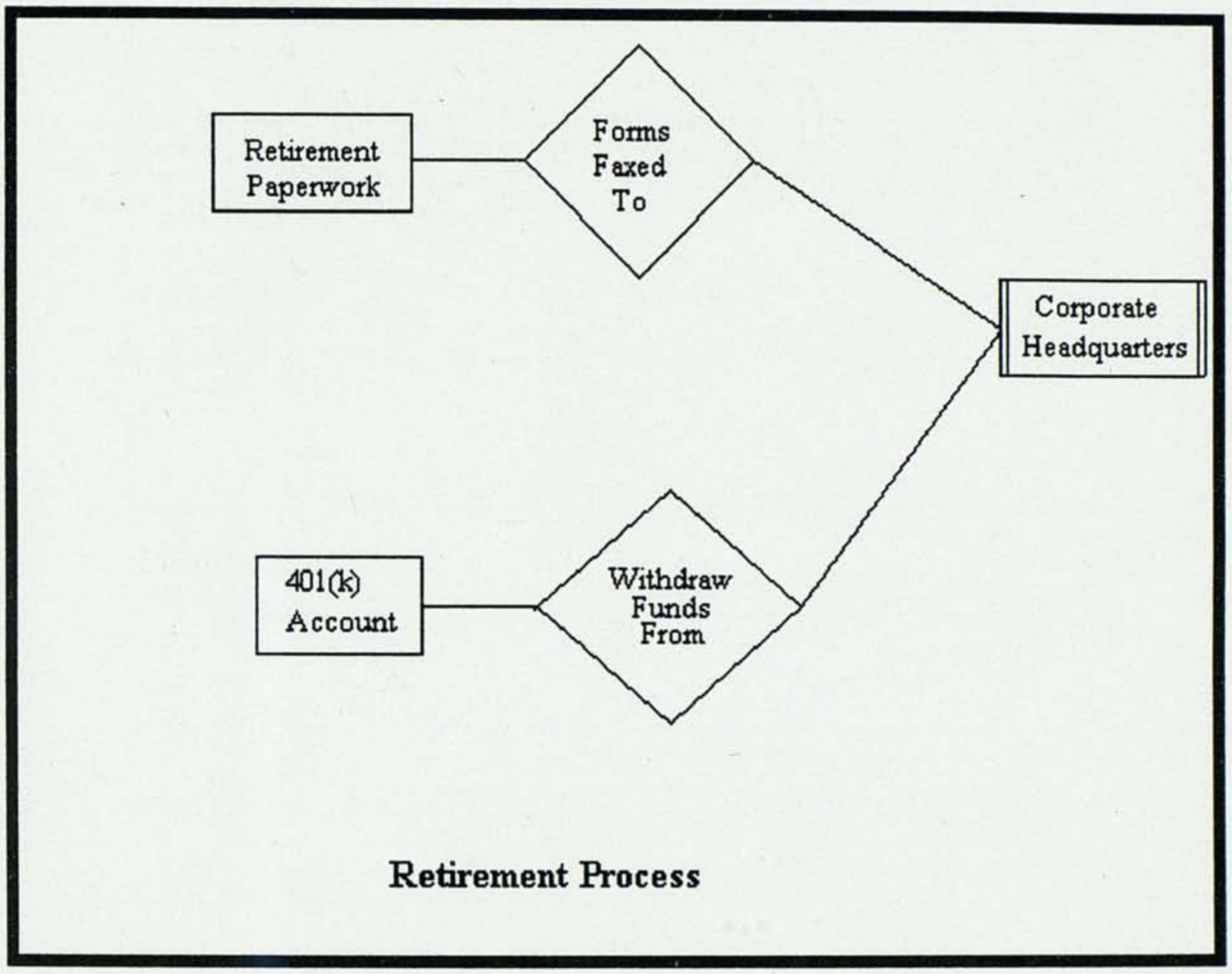

Figure 6 - E-RD illustrating aggregation

Encapsulation, the process of storing data items and the methods which are performed upon the data items into one package, requires a bit of care when being represented by an E-RD construct. By definition, an E-RD is intended to model only data entities and relationships between these entities. Modeling an encapsulated package requires that application code be modeled, to some extent, along with data. In a conceptual model of a knowledge base, this application could should logically not appear. However, as the definition of an encapsulated package dictates that methods and data are tightly bundled in a package, some diagramming method must be provided. If encapsulated packages are considered to behave as a special entity type (with a different symbol from "normal" entities), they could reside in the same diagram without causing a conflict with the rest of the conceptual knowledge base. The assumption must be made that the code resident in the package will affect (i.e., be allowed to modify) the data resident in the package only, although it should certainly be allowed to read other data. The proposed symbol for an encapsulated package entity is the logical OR-gate. Figure 7 presents an alternate view of the Retirement Process of Figures 3 & 6.

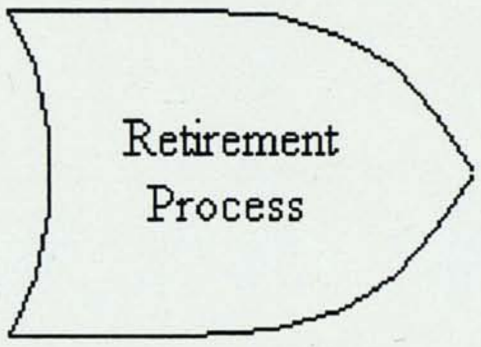

Figure 7 - Encapsulated Package Entity

Note that this package has grouped all the data it needs into one object. The assumption is made that all the data items this package needs are contained within the package, and that only this package will be making changes to these data items.

One obvious question which may arise concerns how this approach can work for an entire knowledge base. So far, the examples in this chapter are limited to single knowledge constructs. However, they can easily be pulled together into a network. Figure 8 shows an expanded E-RD which combines several of the examples of this chapter.

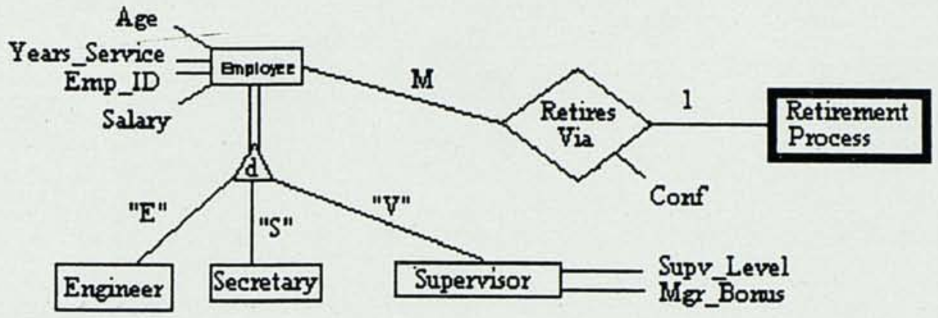

Figure 8 - Network of E-RD knowledge structures

The Retirement Process entity is shown in a bold box, signifying the aggregate entity, although the OR-gate entity of Figure 7, depicting an encapsulated package, could have just as easily been used.

The Essential E-RD

The process by which a knowledge base is designed using this technique will be a two-step process. In the first step, an Essential E-RD is developed using only Chen's original model. This is done to treat the knowledge similarly to data collected for a data-based E-RD. In this way, the knowledge engineer should be free from any pre-conceived biases toward a certain KRM. The second step involves taking the E-RD developed from the first step and transforming the basic entities and relationships into structures which more closely resemble KRM structures. The knowledge base can then be implemented from this second diagram. In some cases the E-RD can be directly mapped to a KRM construct.

In order to develop the first E-RD, the knowledge engineer should approach the knowledge acquisition in much the same way a database developer gleans information from the targeted users of the system to be developed. The primary concern is to identify all the "things" (entities) of the real world and how they each relate to one another. From this information, an E-RD which adheres to Chen's original definition may be developed. Again, the primary intent of this E-RD is to provide a baseline semantic model of the knowledge base, free from implementation- (and thus KRM-) specific structures. The benefit of developing this first model is that it may be more easily tested for *referential integrity* and *normalized entities*, two concepts from the relational database realm which help insure the soundness of the database schema.

In the relational database realm, one of the primary advantages of producing E-RDs lies in their quick mapping into database relations. Mapping to database relations can produce a sound database design only when the E-RD has been normalized and referential integrity is maintained. Normalization is the process by which relations are reduced to a normal form. Although there are many normal forms, only first, second, and third normal forms (1NF, 2NF and 3NF, respectively) are commonly used. Each of the three normal forms relies, in succession, upon the relation adhering to the previous normal form (e.g., a relation in 2NF must, by definition, also be in 1NF). A relation in 3NF is one in which every attribute depends fully upon each and every constituent attribute of the primary key of the relation. Appendix B illustrates how a relation is transformed into 3NF. Referential integrity is a constraint which requires that a tuple (row) in one relation which refers to another relation must refer to an existing tuple (**Elmasri and Navathe 1989**). The basic advantages of having data in third normal form is to insure that attributes are grouped together in the proper entities. The primary benefit of having data adhere to the referential integrity constraint is to insure that data items which depend on other data items will never be left orphaned. The concept of a weak entity type was originally implemented to aid in ensuring referential integrity. Referential integrity is normally not a concern when the E-RD contains only one-to-one relationships. Having a database schema adhere to these constraints helps to insure changes to the data will not result in spurious, unattached data. It also provides an aid when the schema itself must change. No alteration to the schema should be allowed to cause any existing entity to break these constraints. This can be easily checked by consulting the E-RD.

To a great extent, the same benefits can be realized by a knowledge base designed to these standards. Although the schema will not be mapped into relations, normalization and referential integrity are still concerns.

The Implementation E-RD

The second stage of the design methodology involves converting the first E-RD into one in which the knowledge constructs are modeled using the extensions presented earlier in this chapter. The knowledge engineer will use both the first-stage E-RD and knowledge gathered during knowledge acquisition sessions to convert the first-stage (traditional) E-RD into one in which knowledge constructs become more apparent. This E-RD, the Implementation E-RD, can then be used to develop and maintain the knowledge base. In some cases, the physical knowledge base structures can be mapped directly from the Implementation E-RD.

Mapping KRMs from Implementation E-RD Constructs

Frames can be mapped directly from the Implementation E-RD. A seven-step methodology is presented in **(Elmasri and Navathe 1989)** by which an E-RD may be mapped directly to a relational database. As there is little difference at the conceptual level between a RDBMS relation (table) and a frame, a similar methodology will apply to mapping frames. Step 6 of the original methodology, which dealt with multi-valued attributes, is omitted. Although multi-valued attributes are not permitted in relational databases, arrays and lists are valid in many frame management systems.

STEP 1: For each regular (non-weak) entity in the Implementation E-RD, create a frame type. Assign a primary key composed of one or more attributes whose values ensure each instance of that frame will be unique.

STEP 2: For each weak entity type in the Implementation E-RD, create a frame type. Assign as a foreign key the attribute(s) composing the primary key of the owning entity/frame. The primary key of this frame type will be composed of the foreign key and any attribute(s) whose values ensure each instance of that frame will be unique.

STEP 3: For each 1:1 binary relationship in the Implementation E-RD, choose one frame type to contain as a foreign key the primary key of the other frame type. If one entity always participates in the relationship, that entity should receive the foreign key.

STEP 4: For each regular (non-weak) binary 1:M relationship in the Implementation E-RD, place as a foreign key in the frame type on the many (M) side the attribute(s) composing the primary key of the frame type on the one (1) side.

STEP 5: For each binary N:M relationship, create a frame type. This frame type's primary key will be composed of all the attributes composing the primary keys of the frame types on both sides of the relationship.

STEP 6: For each n-ary relationship (n > 2), create a frame type. This frame type's primary key will be composed of all the attributes composing the primary keys of the frame types on all sides of the relationship.

When mapping to frames, it is important to remember that relationships may not need to be mapped to frames. Relationships may be included to show the presence of a rule (see below.)

Rules are harder to map directly from the Implementation E-RD to a knowledge construct. A rule may be defined as some action which occurs based on the validity of one or more conditions. A rule may involve the relationship between two or more entities, but it just as likely will not. In cases where a rule involves only one entity, a weak entity and a weak relationship can be created to show such a relationship; however, creating such a structure adds another step and complicates the Implementation E-RD. If a rule involves only one entity, it should be shown as an attribute of that entity. If a rule involves two or more entities, those entities should be related, and the rule should be diagrammed as an attribute of the relationship. The If-Then relationship should either be written out as a pseudo-rule on the diagram or identified by a unique code and written out elsewhere. Mapping thus becomes an exercise of converting the pseudo-rule to implementation-specific code.

How aggregation is mapped depends on the KRM selected. The "part-of" relationship inherent to aggregation may be modeled in a frame-based system by an attribute-value slot, or in a semantic network by an appropriately titled link.

Inheritance is a feature of classification. An entity which is the subclass of a superclass entity will inherit values from that superclass entity unless specified explicitly. If a particular attribute is to be inherited, it should appear on the superclass entity only. If the subclass entity overrides the superclass, then the attribute should appear on both entities.

Encapsulation, the blending of data and methods into an object or package, depends on the KRM chosen for its mapping. What is important in mapping such an object is that operations on the data are "hidden", i.e., performed only within the object. This is not to say that data from an entity which participates in such an encapsulated object can be used only by that object, but that the specific operation is performed only by that object.

Advantages to Applying Implementation E-RD Methodology

The advantages to using such a methodology in the design of a knowledge base may be seen during both the implementation of the knowledge base and the maintenance portion of the KBS lifecycle. During implementation, both the first-stage E-RD and the Implementation E-RD can be consulted to determine how different portions of the knowledge base inter-relate. This helps to solve the "nature and quantity" problem mentioned in **(Gonzalez and Dankel 1993)**. By developing a graphical portrayal of the knowledge base, knowledge engineers can more quickly develop an understanding of the realm. The most important advantage during the implementation phase lies in avoiding the paradigm shift, also mentioned in **(Gonzalez and Dankel 1993)**. The Implementation E-RD can be consulted to determine the most appropriate choice of the KRM(s) to be used.

Maintenance, historically the costliest phase of any software project lifecycle, can also be aided by the development of an E-RD and Implementation E-RD. When new knowledge must be added, the Implementation E-RD can be consulted to determine how it will affect the existing knowledge base. Extensions to the original design (e.g., new entities) can be added to both E-RDs to determine their impact before physically altering the knowledge base.

CASE STUDY

This chapter will present a case study to illustrate the concepts discussed in this thesis. A full knowledge base will be designed using the methodology presented earlier, and advantages of using this methodology will be presented.

Problem Domain

The case study revolves around the design of a knowledge which is to be used in the physical design of a relational database using Rdb/VMS, a relational database management system (RDBMS) from Digital Electronics Corporation (DEC). Rdb/VMS, or Rdb, is a powerful RDBMS tightly coupled with VMS, the predominate operating system on DEC's popular line of VAX minicomputers. The physical design of an Rdb database is at least as important as the logical design; a poorly designed Rdb database can cause serious performance problems.

One of Rdb's main strengths lies in its ability to spread data across multiple physical storage devices to allow simultaneous input/output operations (I/Os) on the database. Another strength is the ability to optimize the physical structure for particular access patterns. For example, a parent-child (or one-to-many) relationship with a small number of children records can be physically implemented to retrieve the parent and all children records with a single I/O. A parent-child relationship with a larger number of children records can be optimized for two I/Os.

The knowledge base produced from this case study will allow a knowledge-based system to translate a 3NF logical database design and user-supplied parameters into a physical Rdb database. For this case study, the 3NF design is necessary; however, how the knowledge is generally supplied is application-dependent. All applications will provide a body of knowledge from which a knowledge base is created; in this case study, the knowledge comes from the 3NF, user-supplied input, and knowledge acquisition from an expert in the problem area.

Knowledge Domain

There are a number of database-wide parameters which can greatly affect the performance of the Rdb database. These include:

- **Number of Users** - The number of simultaneous users allowed on the database. If this parameter is set too low, deadlocks will prevent users from attaching to the database.
- **Number of VAXcluster Nodes** - VAX nodes (CPU or multiple CPU computers) may be physically linked together in a DECNET (DEC's proprietary transport protocol) cluster, allowing machines to share resources such as printers, disks, etc.
- **Number of Buffers** - Together with Buffer Size (below), this parameter determines the amount of virtual memory reserved for database users' buffer pools. By default, each database user has a buffer into which database pages are read.
- **Buffer Size** - See Number of Buffers (above).
- **Global Buffers** - As mentioned above, buffers are established by default for individual users. Enabling global buffers will establish one buffer pool per VAXcluster node using the database. Establishing global buffers can improve performance by reducing I/O and freeing up memory, if many processes frequently use the same database pages.
- **Fast Commit Processing** - By default, modified pages are flushed to disk when a

COMMIT statement is execute. If Fast Commit Processing is enabled, modified pages are only flushed to disk at user-defined checkpoints. This option can greatly increase throughput for update-intensive databases.

There are several other database-wide parameters which can be specified; however, most require a detailed knowledge not only of Rdb, but also of how the database is performing in its environment. For this reason, other parameters will be discussed only as needed.

The primary storage entity in Rdb is the *storage area.* Each storage area maps to a file where the data physically resides, and, optionally, a *snapshot* file, where data resides temporarily for read-only database users. The use of a snapshot file allows users to read data while another user has a non-exclusive write lock on the same data. Spreading the storage file and the snapshot file on different disk drives allows database administrators (DBAs) to balance I/O operations across multiple disks.

There are a number of parameters which a DBA can set for a storage area. These include:

- **Filename** - The name of the storage file, including the device and directory.
- **Allocation** - The number of *pages* initially allocated to the storage area.
- **Page Size** - The size, in 512-byte blocks, of each page in the storage area.
- **Thresholds** - One, two or three values which represent three possible ranges of free space available on any given page.
- **Interval** - The number of data pages between space area management (SPAM) pages. Rdb uses SPAM pages to govern the placement of data. They are also used to locate the page where an index will be found.
- **Extent** - The number of pages by which the storage area will grow when it must be extended.

- **Extension options** - Provides for greater extension control by allowing the DBA to specify percentage of growth as well as minimum and maximum number of pages to extend the storage area.
- **Snapshot Filename** - The snapshot file is an optional separate which contains data to be used by read-only users of the storage area. Snapshot allocation, extent, and extension options may be specified as well.
- **Write Once** - If the storage area is to reside on a write-once, read-many (WORM) device (for large amounts of static data), this parameter may be specified.

Once a storage area is created, tables may be stored in it by the use of the CREATE STORAGE MAP clause. Storage maps, as their name implies, map tables to storage areas.

Rdb offers a robust set of options for indexing. Parameters include the following:

- **Unique** - A keyword which specifies whether each value of the index must be unique.
- **Column Parameters:**
  - **Asc/Desc** - Describes whether the index is ascending (default) or descending.
  - **Size is *n*** - Compression clause used to limit the number of characters used to retrieve data.
  - **Mapping Values *l* To *h*** - Used to translate numeric columns into a more compact form.
- **Type is Sorted** - Range retrievals tend to work best with sorted (B-tree) indices. If the

  index is defined as sorted, the following parameters may be used:
  - **Node Size** - Size in bytes of each index node.
  - **Percent Fill** -The initial fullness percentage for each node.

- **Usage Update/Query** - USAGE UPDATE sets PERCENT FILL to 70% (the default

  if neither parameter is given); USAGE QUERY sets PERCENT FILL to 100%.
- **Type is Hashed** - Hashed indices are preferable for exact-match queries. They require

  mixed format storage areas to implement.
- **Index Store Clause** - A clause which specifies how the index is to be stored (in one storage area, spread randomly across multiple areas, or spread by some value across multiple areas.) In addition, threshold values similar to the threshold values of the storage area parameters may be specified.

The following data is site-specific and will require user input.

Environment Information

Disks:

- Disk name
- Disk size - Blocks (512 bytes) free
- Disk speed - Seek time
- WORM device - Yes/No

Shadow Sets: (In a shadow set, information written to one drive is copied to another; if the primary drive fails, the secondary drive becomes the primary drive. Although this configuration degrades update-intensive applications to some extent, read-only applications see an improvement since either drive can be used by read-only users. However, shadowing data intended for read-only usage isn't cost effective. Shadowing is primarily intended to ensure uninterrupted data access.)

- Primary disk

- Secondary disk

Node:

- Node name
- Node type (e.g., 6510, MicroVAX 3100)
- Available RAM

Database Information

Relations (Tables):

- Name
- Primary key length - In bytes
- Primary key unique - Yes/No
- Row size - In bytes
- Number of rows
- Primary access - (Insert, update, query, delete)
- Growth rate - Records per day

Attributes:

- Table name
- Attribute name
- Attribute type - Data type (e.g., character, numeric, BLOB (Binary Large OBject))
- Attribute length

Relationships:

- Table 1
- Table 2
- Table 1 cardinality - Number of rows
- Table 2 cardinality - Number of rows
- Table 1 key - Attributes composing "join" key

- Table 2 key - Attributes composing "join" key

Indices:

- Table name
- Field name(s)

At a top-level view, the Rdb-specific portion of the knowledge base can be described in narrative form as follows:

An Rdb database is composed of a root file, one or more storage areas, and, optionally, one or more snapshot files. Parameters can be specified for the database to control usage of memory, number of users and accessible nodes, and the flushing of committed data to disk.

Storage areas consist of data pages and SPAM pages. A storage area has an initial allocation of pages, whose size is also defined when the storage area is created. One set of parameters control the extension of storage areas. The interval between SPAM pages and data pages is controllable.

Indices may be sorted or hashed. Sorted, or B-tree indices, are more appropriate for range retrieval, while hashed indices offer performance benefits for exact-match retrievals. Node size and fullness percentages may be specified for sorted indices.

The remainder of the body of knowledge comes from knowledge acquisition sessions with an expert in Rdb database design.

> To design a physical Rdb database, each table and its relationships to other tables has to be examined. Any conflicts (e.g., optimizing for the update-intensive operations on one table vice read-only operations on another) have to be resolved by assigning some sort of priority. This might be as simple as starting with the tables and relationships used by the most important transactions and working down from

there. One factor that must stay in the DBA's mind at all times is the number of disks and amount of storage available. Given an inexhaustible supply of disks, an almost perfect Rdb database can be designed. Unfortunately, this is not an option for most organizations. Thus, it becomes important to spread data across the available drives. This can be done in a couple of different manners. First, a table can be spread across multiple drives. The other option is to keep each table on one drive, and spread the various tables across different drives. The first option offers the benefit of spreading I/O on a heavily-hit table across multiple devices. The second option provides for easier maintenance, and works better if no one table is hit more heavily than any other. Snapshot files should be put on a device other than the device on which the main storage file resides. This allows a more even distribution; read-only users don't have to wait for read-write transactions to finish before they can access the data. Parent-child relationships may be set up in two different ways. First, if their is a known upper limit to the number of children records, the parent record and all its children can be placed on one page, thus allowing the parent and all children records to be retrieved by a single I/O. This is known as "optimizing for one I/O." The way to do this is to create a mixed-format storage area (or areas, if the tables are to be spread across multiple devices.) Size the pages large enough to contain the parent record and a hashed index, along with enough space to hold all children records and their hashed indices. The *Guide to Database Design and Definition* contains the appropriate formulae for this. Keep in mind there will be duplicate children records; the formula for hashed index size must make use of this fact. When the relationship is optimized for two I/Os, the parent record, its hashed index, *and* the hashed index for the children record are stored in one storage area, while the children record are stored in another. One I/O retrieves the parent record and the index to the children records; the second I/O retrieves the children records. This setup is more appropriate when the parent record is quite large or there isn't an upper limit on the number of children records.

The proper design of indices is another important point. Hashed indices are wonderful for exact-match queries, but they can actually degrade performance for range retrieval queries. By far, their best usage is for the parent-child relationships discussed above. Sorted indices can be combined with hashed indices to allow both exact-math and range retrievals, but this can be quite tricky to set up. Knowledge of how the index and its table will most often be used (query or query/update) is required to set the percent fill parameter. An index used mostly for querying should be set to close to 100% full, while an index used for update should be set around 70% full. The node size can be calculated by the formula:

3 * (*key size* + *number of columns* + 11) + 32

The "3" ensures that three entries will fit into a node, which helps to keep the structure more of a B-tree than a pure binary tree. *Key size* is the total size, in bytes, of all the constituent columns of the index, plus one byte per column for the null indicator. *Number of columns* is, obviously, the number of columns comprising the index. The "11" is the maximum overhead per index key within a node. The "32" is the node's overhead.

Care must also be taken in establishing storage areas. One easy rule is to use (at least) one storage area per table. In this way, changes can be more easily made "after the fact", after the database has been established and the data loaded. Again, a storage area and its snapshot file should be on separate devices. The page size should allow for some comfortable number of data rows and all the indices defined for the table. Try to store the indices with the data. The *Guide to Database*

Performance and Tuning provides guidelines for setting the fullness threshold values. For a storage area containing only one table, the values should be set to always allow one more row to fit fully on the page. In other words, the third parameter should be set so that the difference between the page size and the third fullness parameter is less than the uncompressed row size. For example, if a page has been set to 1000 bytes, and the row size is 100 bytes, the third parameter should be set to at least 91%, thus indicating the page is full at 910 bytes. When this threshold is reached, no more data rows will attempt to be inserted, thus eliminating fragmented data rows. The difference between the page size and the third fullness setting can be considered as free space for future growth to existing rows. The other two parameters are used mostly for storage areas housing multiple table types. The parent-child relationship type mentioned previously should make use of the other two parameters. The first one should be set to ensure that a page which has not yet met that parameter can store the largest row type once more. Likewise, the second parameter should ensure the same for the smallest row type. The SPAM interval parameter should be allowed to take the default and modified only after an extensive investigation of the database's performance, noting disk I/O vs. SPAM page locking. Consult the *Guide to Database Performance and Tuning* and the *Guide to Database Maintenance* for more details. The extent size parameter should be set to allow minimal disruption to users when the storage area must extend. Detailed knowledge of how the storage area will grow and over what time period growth can be expected is required. For example, an application which adds another 10% to a storage area every month during a heavy insertion cycle should have its extent set to 10% (or slightly more) to ensure that only one extent will be made during the input cycle. The extension options allow for a finer granularity of control over extensions by allowing percentage growth, with a guaranteed minimum and maximum number of pages the area will grow. Finally, large amounts of static text, bit-mapped images, or other BLOBs (Binary Large OBjects) are ideal candidates for storage areas residing on WORM devices.

The database-wide parameters can require knowledge about the nodes on which the database will be accessed. The first parameter which can cause trouble is number of users. Certain fourth-generation languages (4GLs) have users declare a read-only transaction to query a record, then a read-write transaction to update it. This results in two users, as far as the database is concerned. Since no harm is done by setting this parameter too high, it is a good idea to set it to 200 to start.

Buffer parameters, which affect Rdb's usage of virtual memory, are considerably more tricky to set. The first question is whether database users access the same pages often, in which case global buffering would be useful. Otherwise, local buffering is the best option. Unfortunately, these parameters apply to every node on the cluster. Thus, the machine with the smallest amount of RAM on the cluster becomes the driving factor. For local buffering, the buffer pool per user can be calculated by multiplying the number of buffers by the buffer size. To prevent VMS from paging, the sum of the buffer pools for all users should not exceed the amount of physical memory. However, if the buffer pool is too small, Rdb will be forced to continually swap pages to and from memory, which would also result in a paging problem. Setting the buffer size high brings in more data, thus allowing related rows to be brought into memory together, and also improves sequential searches. Specifying a high number of buffers makes it more likely that rows previously used will still be in memory, which improves performance of transactions which use the same data several times. Generally speaking, however, having many small buffers results in less paging than having fewer large buffers.

In global buffering, Rdb establishes a global buffer pool on each node. Users then map that global section to their own virtual memory. The advantages global buffering brings are for applications in which users access the same database pages fairly often. Detailed knowledge of the application's performance and the number of database pages used by multiple applications is required to effectively institute global buffering, so global buffering should be done after the database is created and information has been gathered to determine how large a global buffer pool should be.

Development of Essential E-RD

The Essential E-RD is created from knowledge from the database parameters and user-supplied information. In this case study, the expert knowledge is used only by the Implementation E-RD. In general, the knowledge required for the Essential E-RD will come not from an expert's analysis of the problem milieu, but from a general understanding of the problem realm, while the knowledge required for the Implementation E-RD will come from a typical knowledge acquisition phase. However, this may vary from application to application. The following E-RDs comprise the Essential E-RD.

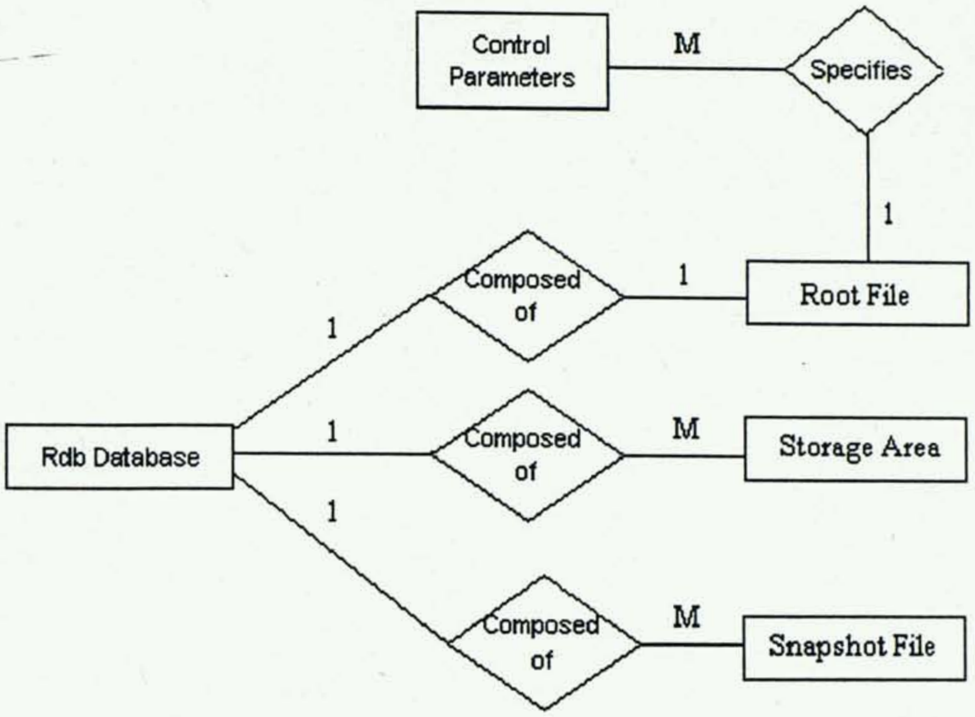

Figure 9 - Top-level E-RD of Rdb Database

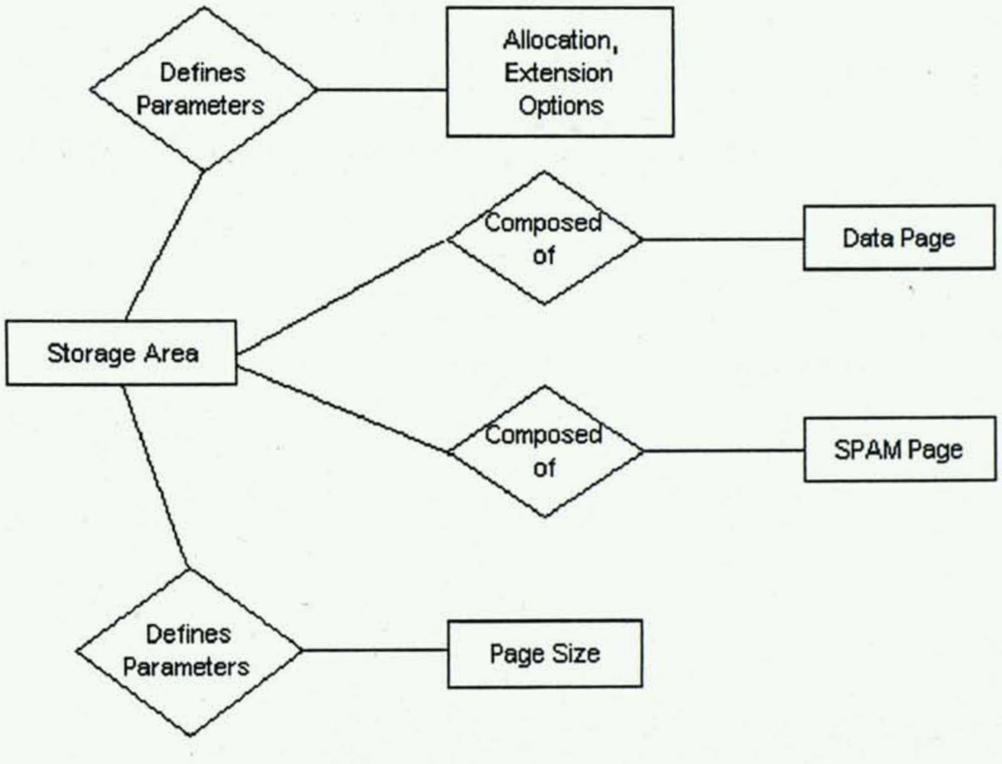

Figure 10 - Expanded E-RD of Storage Area

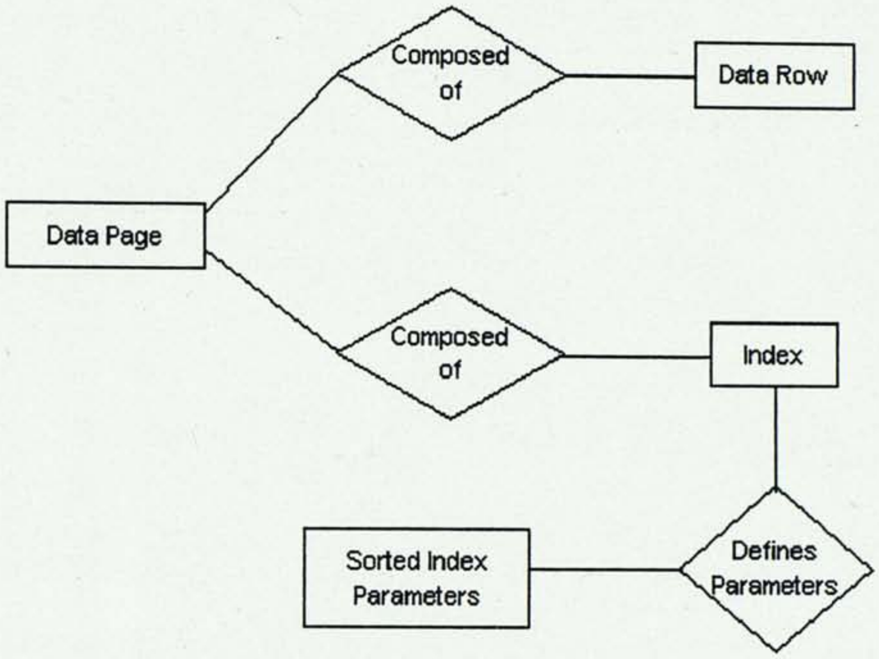

Figure 11 -Expanded E-RD of Data Page

The above diagrams are meant to be interconnected; to make them more readable, they are separated by area of context. Similarly, environment information may be depicted as such:

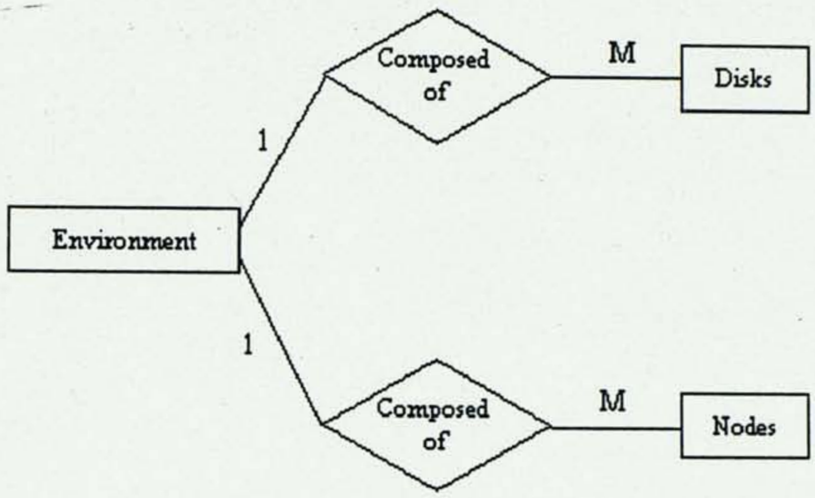

Figure 12 - E-RD of Environment

The E-RD for the database (i.e., the 3NF logical database to be converted into a physical Rdb database) might look as follows:

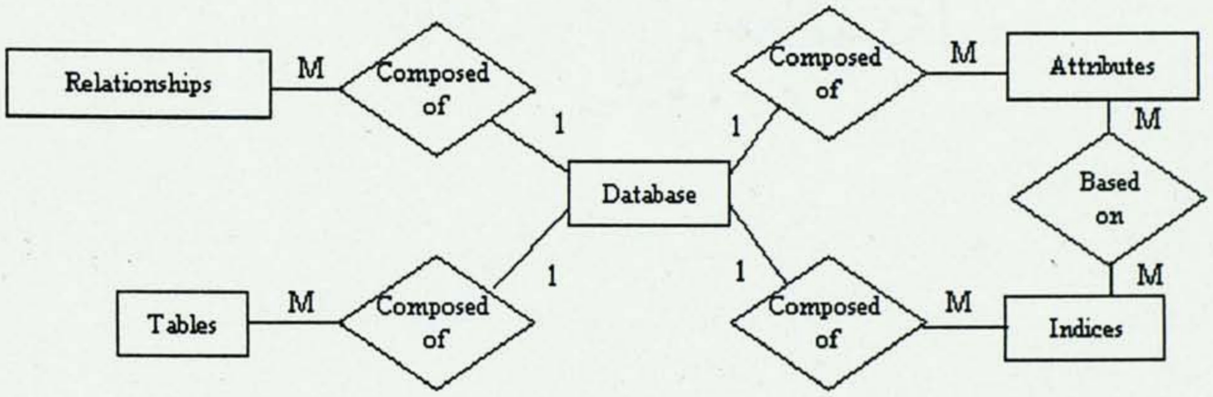

Figure 13 - E-RD of 3NF Logical Database

These diagrams provide for the first step of the two-step process by providing standard E-RDs of the knowledge domain. These diagrams are intended only to provide developers with insight into the relationships between the various entities of the domain.

Development of Implementation E-RD

The expert-supplied knowledge can be depicted graphically by using the extensions detailed in chapter 4. Again, attributes are not shown and diagrams are divided into sub-diagrams to improve readability. Also, rules are depicted by numbers and described at the end of the diagrams.

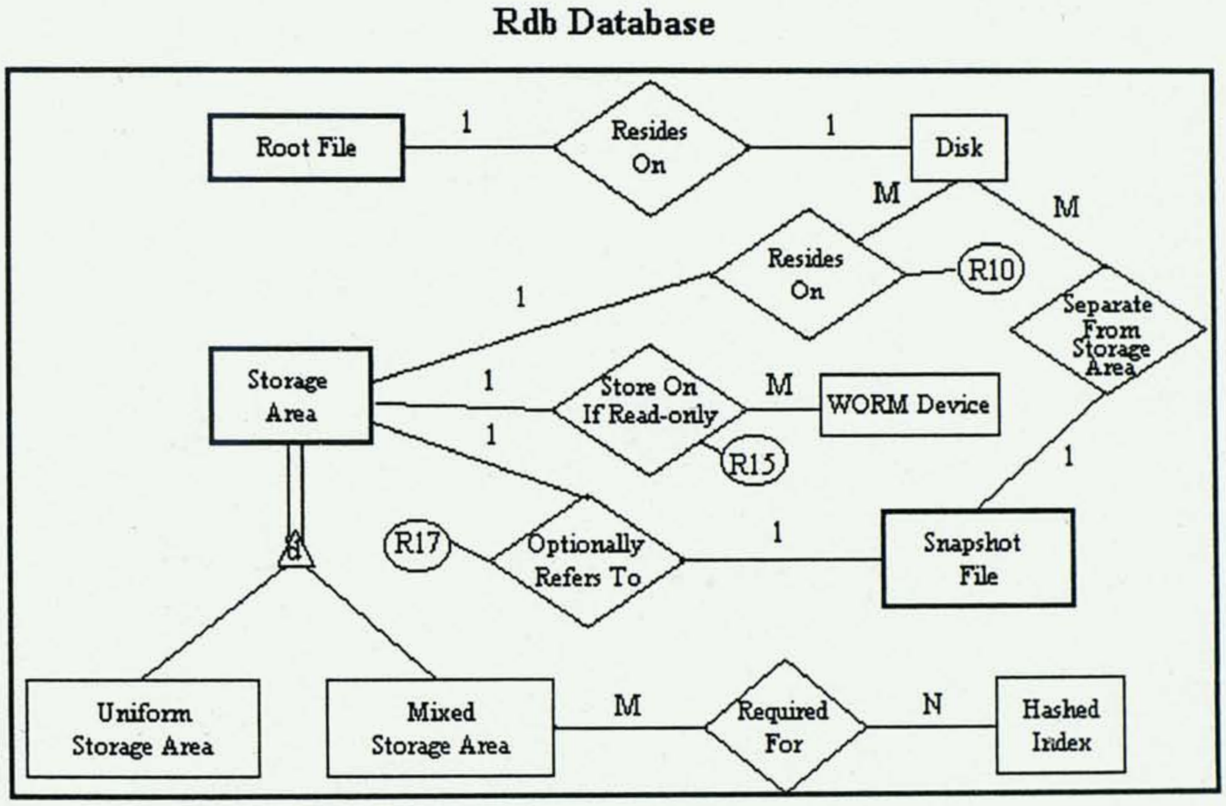

Figure 14 - Top-level E-RD of Rdb Database with Knowledge Structures

Figure 14 illustrates several knowledge constructs in E-RD form. The entire structure represents an aggregate entity (**Rdb Database**), as noted by the heavy lines around the entire diagram. The entities **Root File**, **Storage Area** and **Snapshot File** are also aggregate entities, and each will be expanded in later diagrams. The **Storage Area** entity represents a superclass of the entity types

Uniform and **Mixed Storage Area** (or, optionally, **Uniform** and **Mixed Storage Area** are sub-classes of the **Storage Area** entity type.) Although two of the relationships involving the **Disk** entity also involve a **Resides On** relationship, there must be three separate relationships. The **Optionally Refers To** and **Store On If Read-only**, by their names, imply some sort of If-Then relationship.

Root File

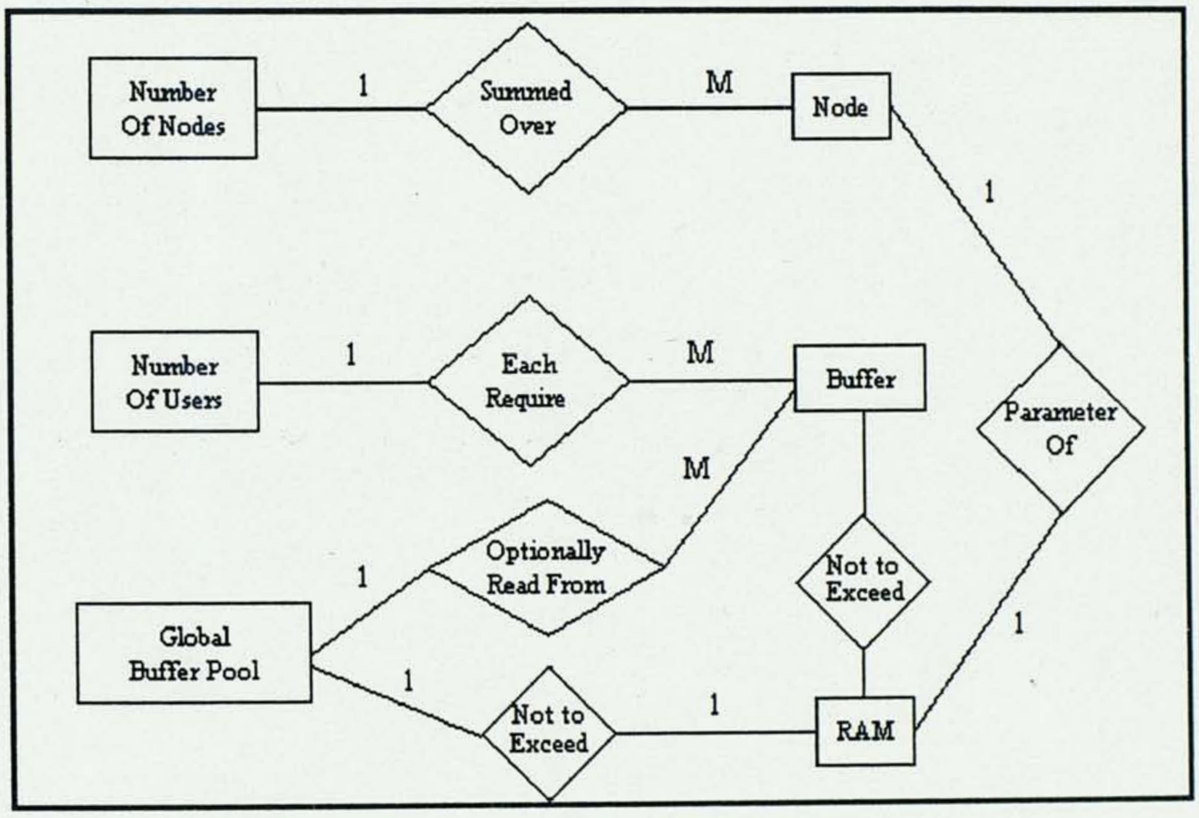

Figure 15 - Detailed E-RD of Root File

Like the **Optionally Refers To** relationship of Figure 14, Figure 15's **Optionally Read From** implies an If-Then relationship.

Snapshot File

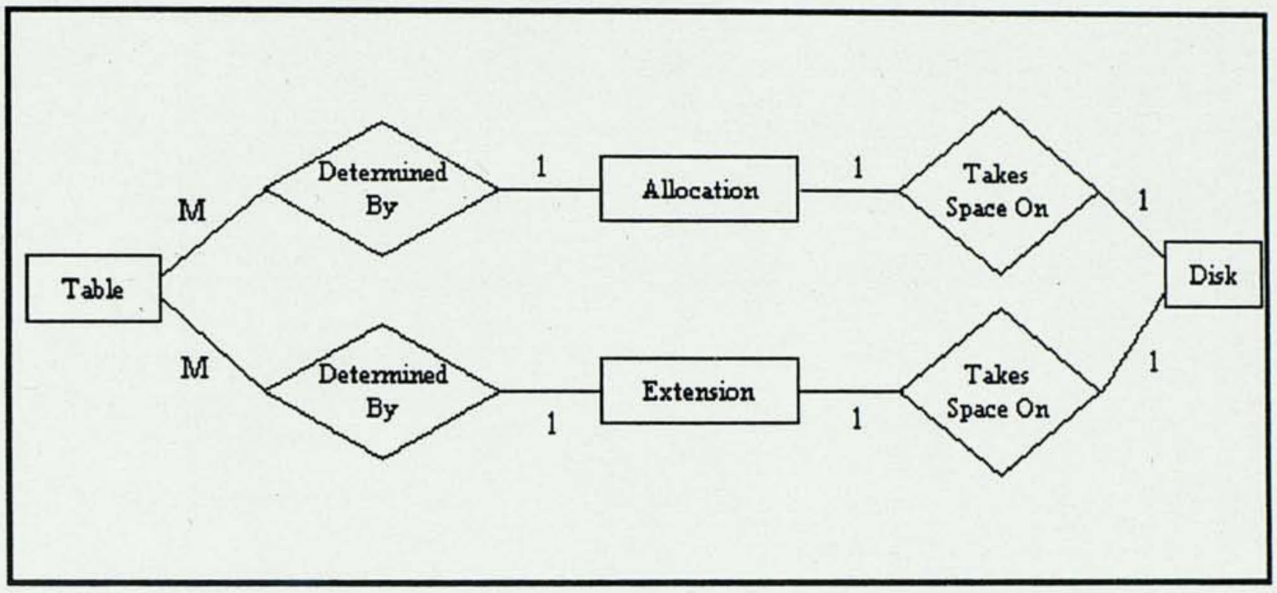

Figure 16 - Detailed E-RD of Snapshot File

Figures 16 and 17 (below) do not contain any knowledge structures (other than aggregation) as they appear here; however, a later diagram will illustrate a way around the binary relationship restriction of the classic data-modeling E-RD. (i.e., the **Determined By** and **Takes Space On** relationships appearing twice and relating the same entities.)

Storage Area

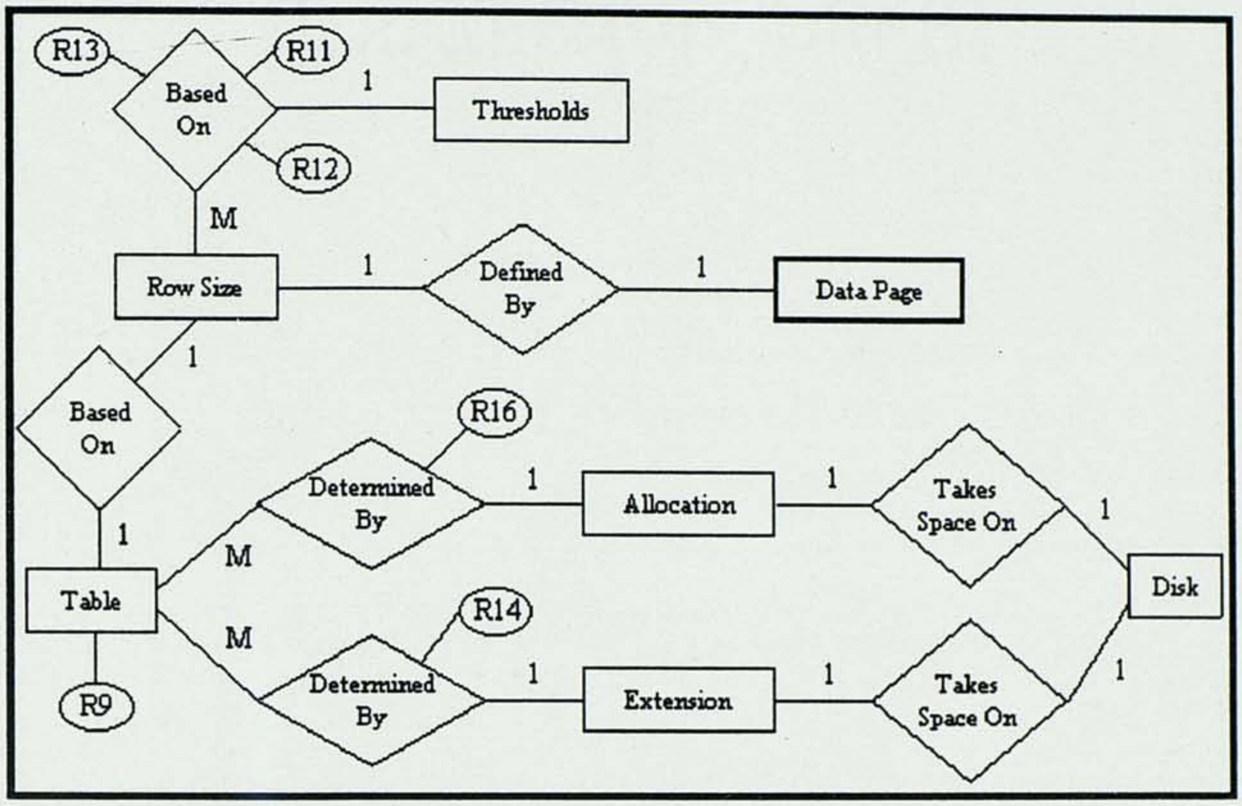

Figure 17 - Detailed E-RD of Storage Area

Data Page

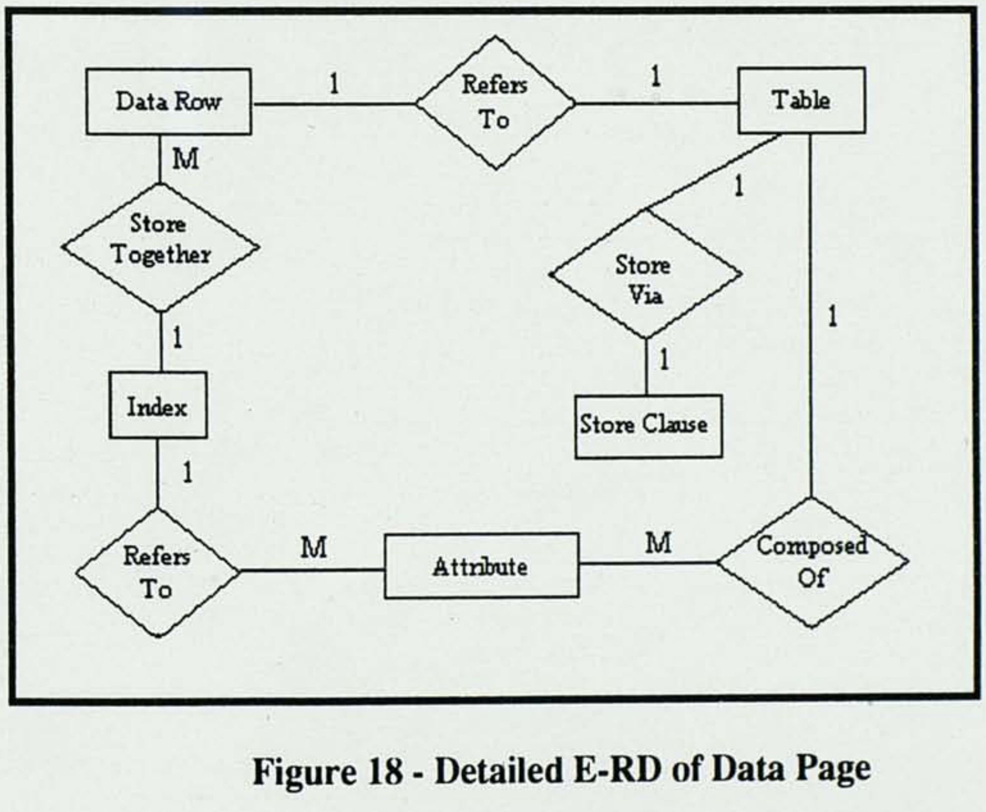

Figure 18 - Detailed E-RD of Data Page

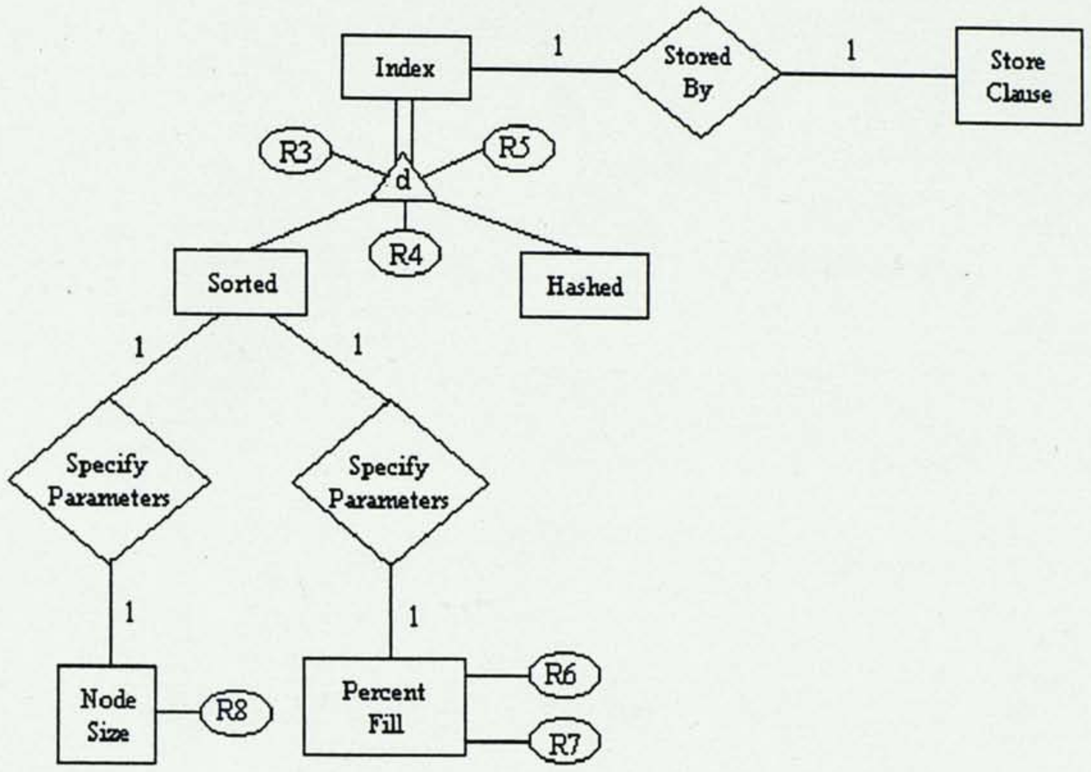

Figure 19 - Detailed E-RD of Index

Figure 19 contains a super-class/sub-class relationship between **Index** and **Sorted** and **Hashed**, two types of indices.

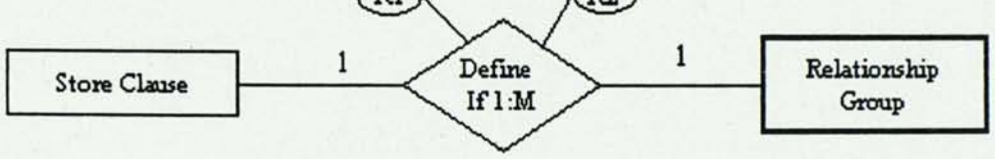

Figure 20 - E-RD of Store Clause

Figure 20 contains an If-Then relationship and an aggregate entity type.

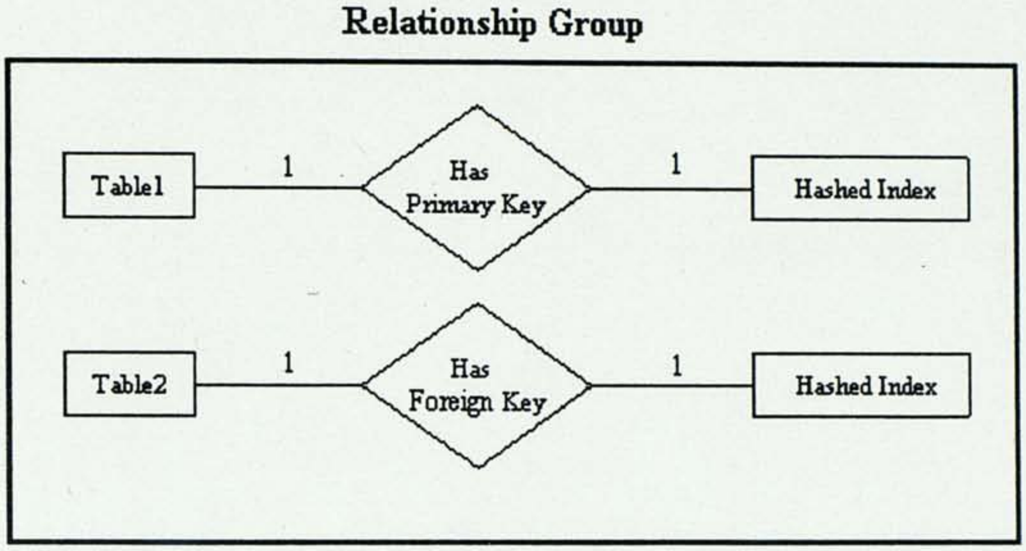

Figure 21 - E-RD of Relationship Group

The requirement that relationships be binary in nature becomes quite restrictive on an E-RD intended to diagram knowledge. In many of these diagrams, the same relationship is used multiple times, often relating the same entities. Encapsulation provides one way around this restriction. The **Determined By** and **Takes Space On** relationships of Figures 16 and 17 provide a good example of this. Figure 16 (and the similar portion of Figure 17) may be re-drawn as such:

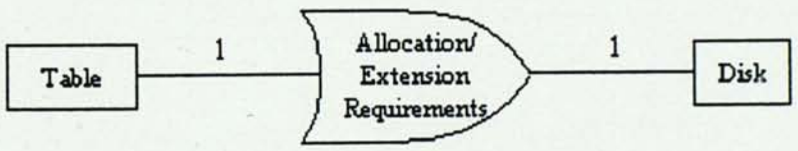

Figure 22 - Encapsulation Snapshot File/Storage Area E-RD

Here, the encapsulated package takes the form of a relationship while the example in chapter 4 used a similar construct as an entity. The relationship is perhaps a more natural form, as such a package provides for some sort of operation on data related to it.

The If-Then relationships, identified as attributes in the preceding diagrams, are listed below.

Table 2

Rules from Implementation E-RD

R1	IF (1:M relationship) AND ((upper limit of children known) AND (small amount of child data)) THEN (optimize for 1 I/O)
R2	IF (1:M relationship) AND ((upper limit of children unknown) OR (large amount of child data)) THEN (optimize for 2 I/Os)
R3	IF (range retrieval) THEN (sorted index)
R4	IF (exact match) THEN (hashed index)

R5	IF (1:M relationship) THEN (hashed index)
R6	IF (query) THEN (percent fill) = 100
R7	IF (update) THEN (percent fill) = 70
R8	IF (sorted index) THEN (node size = 3 * (key size + # of columns + 11) + 32)
R9	IF (new table) THEN (new storage area)
R10	IF (new storage area) THEN (snapshot on separate disk)
R11	IF (new storage area) THEN (1st fullness parameter < ((page size - largest row size) / page size) * 100)
R12	IF (new storage area) THEN (2nd fullness parameter < ((page size - smallest row size) / page size) * 100)
R13	IF (new storage area) THEN (3rd fullness parameter = ((page size - row size) / page size) * 100)

R14	IF (new storage area) THEN (extent = growth rate* row size)
R15	IF (table has BLOBS) THEN (store on WORM device)
R16	IF (new table) THEN (allocation = row size * num rows * 1.1)
R17	IF (application not read-only) THEN (use snapshot)

Advantages of Applying Implementation E-RD Methodology to Case Study

The use of this methodology offers several advantages to the design and maintenance of this system. The "paradigm shift" problem of **(Gonzalez and Dankel 1993)** can be avoided by selecting the proper KRM. A perusal of the Implementation E-RD shows classification and aggregation as well as rules. Several KRMs can provide for classification and aggregation. However, the selected KRM must be able to provide knowledge to rules. Frames can provide classification and aggregation constructs and can store knowledge for rules. Therefore, a hybrid system combining frames and rules would be the best KRM for this system.

The "nature and quantity" problem also discussed in **(Gonzalez and Dankel 1993)** can likewise be eliminated by the proper use of this methodology. The knowledge base has been reduced to a set of diagrams, thus providing a graphical depiction of the problem area. The nature and quantity of the data in the problem realm may be determined by visual inspection as opposed to reading through unstructured interview notes. It is important to remember that, as popular as entity-relationship modeling is, it owes its popularity to the accompanying entity-relationship diagramming technique.

The methodology provides a powerful benefit to the (usually costly) maintenance portion of the software lifecycle as well. Debugging problems in the knowledge base becomes mush easier when the knowledge base has been depicted graphically. If, for example, the StorageAreaExtent is not being computed properly, the developer can determine by inspection that the rule R14 and the entities Table and Extension provide the data for the computation. The impact of adding new knowledge can be ascertained quickly with an Implementation E-RD as well. As any new knowledge added to the knowledge base will in some way touch existing knowledge, adding the new structure to the Implementation E-RD will allow the developer to determine the impact of the new knowledge by visual inspection.

Mapping Implementation E-RD to Knowledge Constructs

Using the mapping procedure discussed earlier on the Implementation E-RD of Figure 14 will produce frames for RootFile and Disk. One of the two would require a foreign key which would be the primary key of the other. In this case, the RootFile frame would be the more logical choice to contain the foreign key, since all RootFiles exist on a Disk, while not all Disks contain a RootFile. SnapshotFile and Disk would be treated the same way, as would StorageArea and WORMDevice. UniformStorageArea and MixedStorageArea are subclasses of StorageArea, and would inherit applicable characteristics of that frame. Finally, MixedStorageArea and HashedIndex, because of their M:N relationship, would require the creation of a third frame, which would contain as a primary key the combination of the primary keys of MixedStorageArea and HashedIndex.

The following table presents frame types mapped from the Implementation E-RD, their primary keys, their foreign keys and frames and keys referenced, and aggregation and classification information. These frames come from Figure 14.

Table 3

Frames Mapped from Implementation E-RD

Frame Name	Primary Key	Foreign Key References
RootFile	**RootFileName**	**Disk.DiskName**
Disk	**DiskName**	**None**
StorageArea	**StorageAreaName**	**Disk.DiskName**
		WORMDevice.DeviceName
		SnapshotFile.FileName
WORMDevice	**DeviceName**	**None**
SnapshotFile	**FileName**	**Disk.DiskName**
UniformStorageArea	**AreaName**	**None**
MixedStorageArea	**AreaName**	**None**
HashedIndex	**IndexName**	**None**
HashedStorage	**IndexName, AreaName**	**None**

Aggregation can be depicted in frames by the inclusion of a "part-of" attribute. For example, the frame GlobalBufferPool would have the slot:

part-of : RootFile

Similarly, classification can be shown using the format of **(Gonzalez and Dankel 1993)**. The MixedStorageArea frame would have the slot:

specialization-of : StorageArea

while StorageArea would have the slot:

generalization-of : (MixedStorageArea, UniformStorageArea)

CONCLUSIONS

The need for a structured methodology for knowledge base design is inarguably a real requirement. Unlike their counterparts in information system development who view database design and development and application software design and development as separate issues, developers of knowledge-based systems too often view the knowledge base and the application program as one package. At the physical level, the two may be combined in one package, but at the conceptual level, the knowledge base should be considered separately. This becomes more of a concern as knowledge based systems come out of the research laboratory and into the marketplace. As this happens, knowledge bases and databases will require some sort of integration. Knowledge bases are requiring larger and larger amounts of data, and databases are requiring intelligent features traditionally associated with knowledge based systems. For the integration of knowledge bases and databases to come to a successful fruition, an attempt must be made to apply the same design principles to knowledge bases as have been applied to databases for many years.

Knowledge bases and databases share many similarities. As has been pointed out earlier, semantic models can be used to model both. Peter Chen's entity-relationship model, the best known of any semantic database model, offers to designers of knowledge bases a tool by which most knowledge representation structures may be modeled.

The two-stage methodology espoused herein serves a dual purpose. First, it allows knowledge engineers to become more familiar with the inter-relationships at a highly conceptual level. Second, it offers a valuable tool to prevent the inherent problems of knowledge base design and to provide for easier knowledge base maintenance. In some cases, the physical knowledge base may be mapped directly from the Implementation E-RD. Even if the physical knowledge base is not mapped directly from the Implementation E-RD, the advantages to the design and maintenance of the knowledge base make the development of these E-RDs a worthwhile task for knowledge engineers. The two stages of software development unique to knowledge-based systems, knowledge acquisition and knowledge engineering, are thus aided by this methodology, as is the maintenance portion of the application's lifecycle. Knowledge acquisition is made easier by providing developers a graphical representation of not only the problem domain (the Essential E-RD), but also the conceptual and physical knowledge base (the Implementation E-RD.) Knowledge engineering becomes easier by avoiding the problem of selecting an improper knowledge representation model, as well as providing a conceptual knowledge base which can be mapped directly to some physical knowledge structures.

Applying the methodology to a test case illustrated that it works very well for frame-based systems. Indeed, frames can be mapped directly from the knowledge E-RD. Aggregation and other similar relationships used in semantic, or associative, networks are also easy to depict, as is classification. Unfortunately, the static nature of E-RDs makes them too inflexible to depict the behavioral structure of rules at a conceptual level. However, including references to rules and encapsulated packages on the Implementation E-RD makes the maintenance task easier, since the relationship between the entities of the knowledge base and the rules and packages can be determined visually.

For this methodology to truly work for knowledge bases, the knowledge engineer has to first view the problem as one would view a typical information system problem. Only by divorcing himself or herself from the typical rapid-prototyping mindset of most knowledge-based system developers can the knowledge engineer produce an E-RD detailing the inter-relationships of the problem area at the conceptual level. Furthermore, this first-stage E-RD becomes the backbone of the Implementation E-RD from which the knowledge base can be developed and maintained.

As new knowledge representation models are developed, so too must structures be created or adapted to model these new KRMs in an E-RD. For example, the emergence of blackboard systems may well require a new knowledge representation model. Likewise, the E-RD is an evolving tool. As new E-RD structures, concepts and methodologies are developed, they should be incorporated into the design methodology. One noteworthy example of this is research into action-modeling E-RDs, which may well provide an answer to the dilemma of modeling rules via E-RD constructs. In either case, whether newly-developed KRMs require E-RD constructs or newly-developed E-RD constructs can model existing KRMs, the goal of the researcher expanding this tool should reflect the goal of this thesis: to efficiently model KRMs for the design and construction of maintainable knowledge bases.

APPENDIX A
The Entity-Relationship Model

As mentioned previously, Peter Chen defined entities and relationships as the major constructs of his entity-relationship model. This section will define these and other constructs more closely and focus on their diagramming techniques.

Recall that an entity was defined as "a 'thing' which can be distinctly identified," while a relationship is "an association between entities." **(Chen 1976)** The information about entities and relationships exists as attribute-value pairs. For example, the entity Employee may be composed of the attributes Emp_Name, Emp_ID, Birthdate, and Salary. An instance of this entity might be Emp_Name = John Doe, Emp_ID = 12345, Birthdate = 1 January 1950, and Salary = \$50,000. Chen also defined the concepts of *regular* and *weak* entities and relationships. A regular entity is one which does not require a relationship with another entity to exist; a weak entity requires such a relationship. A regular relationship is one in which all participating entities are defined by own attributes, while a weak relationship has at least one entity which is identified by another relationship. A common example of these concepts is that of employees and dependents. The entity Employee is a regular entity, since it does not require any relationships in order to exist. Dependents is a weak entity, since it depends on its relationship with the Employee entity to exist. Likewise, any relationship between Dependents and another entity will be a weak relationship, since Dependent is a weak entity.

Chen also defined *mapping ratios* (often called *cardinality ratios*) between the participating entities of a relationship. Relationships may be one-to-one (1:1), one-to-many (1:m), or many-to-many (m:n). This ratio refers to how many instances of each of the entities in the binary relationship can exist. For example, the Employee-Dependent relationship is said to be 1:m since each employee may have many dependents, but each dependent will have only one sponsoring employee.

The diagramming constructs Chen proposed are quite simple. Entities are drawn as rectangles. A weak entity is shown as a rectangle within a rectangle. Relationships are drawn as diamonds, connected to their participating entities by lines. Weak relationships are denoted by a diamond within a diamond. The cardinality ratio is shown by a "1", "M", or "N" on the line between the relationship and the entity. Although Chen did not propose attributes to be included in the E-RD, they commonly are, either enclosed in ovals or simply named and attached to their respective entity or relationship. The *primary key* (the attribute(s) which uniquely identifies an instance of the entity) is usually underlined.

Figure A-1 illustrates the Employee-Dependent relationship. Reading this E-RD gives the information that Dependent is a weak entity and relies on Employee for its existence. Emp_Dep is a weak relationship since it involves the weak Dependent entity. Emp_ID is the primary key of the Employee entity, and a *composite key* (two or more attributes which together form a key) of Emp_ID and Dep_ID is the primary key of the Dependent entity.

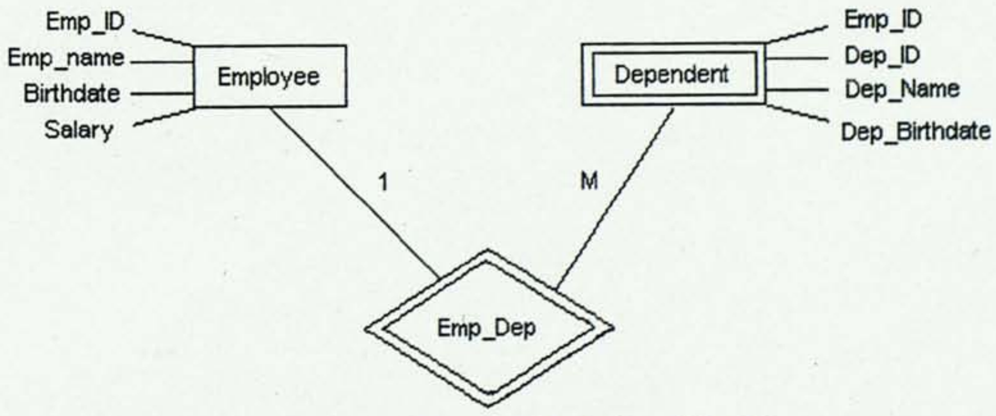

Figure A-1 - Example Entity Relationship Diagram

APPENDIX B
Transforming a Relation into Third Normal Form (3NF)

Normalizing an entity to 3NF guarantees its constituent attributes "belong" to it; thus, an operation on an attribute will affect only the proper entity and its attributes. Transforming a denormalized relation into one of 3NF must start by forcing it into 1NF. Date **(Date 1990)** defines 1NF as such:

"A relation is in *first normal form* (1NF) if and only if all underlying simple domains contain atomic values only."

This constraint has been called the "no-repeating" constraint. Basically, it says that the attributes (domains, in Date's terminology) of a relation cannot have repeating values. For example, an attribute called "EmpName" can have only one occurrence of an employee name; it must not be an array of employee names.

Second normal form is defined in **(Elmasri and Navathe 1989)** as follows:

"A relation...R is in **second normal form (2NF)** if every...attribute A in R is not partially dependent on *any* key of R. This definition can be restated as follows: A relation...R is in **2NF** if every...attribute A in R is fully functional on *every key* of R."

Date defines 2NF slightly differently:

"A relation is in *second normal form* (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key."

The difference between these two definitions is not as great is it might first appear. Assuming one selected the proper choice of primary keys, a relation not in 2NF would fail

Date's criteria by inspection. If the wrong primary key were chosen, the Elmasri/Navathe test might be in order.

As an example of the first two normal forms, consider employees and dependents. To make an employees relation adhere to 1NF, an attribute called "DepID", the identification number of a dependent, would reside in every tuple of the Employee relation. DepID would contain only atomic values, and an Employee relation, consisting of EmpID (the primary key), EmpName, DepID, and DepName, would be in 1NF. However, DepName is not fully dependent on EmpID. To force the Employee relation into 2NF, a separate Dependent relation should be created, consisting of DepID (the primary key) and DepName.

The transition to 3NF is more subtle. Going back to Date:

> "A relation is in *third normal form* (3NF) if and only if, for all time, each tuple consists of a primary key value that identifies some entity, together with a set of zero or more mutually independent attribute values that describe that entity in some way."

To extend the Employee relation example to show the difference between 2NF and 3NF, suppose the Employee example contained the attributes EmpID, EmpName, DepID, JobCode, and Salary. Furthermore, suppose Salary is dependent on JobCode, which has been assigned the status of a *foreign* key in the relation Employee, as it is also the primary key of a relation called Job. Employee could be said to pass the Elmasri/Navathe test for 2NF. However, it does not pass the 3NF criteria. Removing Salary from the Employee relation and placing it in the Job relation would transform Employee into a 3NF relation.

BIBLIOGRAPHY

Anger, Frank D. , Rita V. Rodriguez and Douglas D. Dankel II. "Organizing an Expert System's Knowledge Base Using Relational Database Techniques", Proceedings of the First Florida Artificial Intelligence Research Symposium, The Florida AI Research Symposium, 1988

Borgida, Alexander. "Knowledge Representation, Semantic Modeling: Similarities and Differences", Entity-Relationship Approach: The Core of Conceptual Modeling, North-Holland, 1991

Briand, H., J-F. Hue and Y. Simon. "Expert System for Translating an E-R Diagram Into Databases", IEEE International Conference on Entity-Relationship Approach, IEEE Computer Society Press, 1985

Chen, Peter. "The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database Systems 1, Number 1, March 1976

Codd, E.F. "A Relational Model of Data for Large Shared Data Banks", Communications of the ACM 13, Number 6, June 1970

Cohen, Paul R. "Methodological Problems, a Model-based Design and Analysis Methodology, and an Example", Methodologies for Intelligent Systems 5, 1990, North-Holland

Cox, Earl. "How a Machine Reasons: Part 7", AI Expert, February 1993

Date, C. J. An Introduction to Database Systems, Volume I, Addison-Wesley, 1990

De Antonellis, Valeria. "Databases and Knowledge-bases: Which Approach is Good for What?", Entity-Relationship Approach: A Bridge to the User, North-Holland, 1989

Debenham, John. "The Construction of Maintainable Knowledge Bases", The Next Generation of Information Systems: From Data to Knowledge, Springer-Verlag, 1992

Digital Equipment Corporation. VAX Rdb/VMS Guide to Database Design and

Definition, 1991, Digital Equipment Corporation

Digital Equipment Corporation. VAX Rdb/VMS Guide to Database Maintenance, 1991, Digital Equipment Corporation

Digital Equipment Corporation. VAX Rdb/VMS Guide to Database Performance and Tuning, 1991, Digital Equipment Corporation

Digital Equipment Corporation. VAX Rdb/VMS SQL Reference Manual, 1991, Digital Equipment Corporation

Elmasri, Ramez and Shamkant Navathe. Fundamentals of Database Systems, Benjamin/Cummings, 1989

Entsminger, Gary. The Tao of Objects: A Beginner's Guide to Object-Oriented Programming, M & T Books, 1990

Feldman, Paul and Guy Fitzgerald. "Representing Rules Through Modelling Entity Behavior", IEEE International Conference on Entity-Relationship Approach, IEEE Computer Society Press, 1985

Fenn, J.A. and Veren, L.C. "Expert System Development Methodologies in Theory and Practice", Proceedings of the IEEE/ACM International Conference on Developing and Managing Expert System Programs, 1991, IEEE Computer Society Press

Gonzalez, Avelino and Douglas D. Dankel II. Engineering of Knowledge-Based Systems: Theory and Practice, Prentice-Hall, 1993

Held, James P. and John V. Carlis. "Conceptual Data Modelling of an Expert System", IEEE International Conference on Entity-Relationship Approach, IEEE Computer Society Press, 1985

Hickman, Frank R. , Jonathan L. Killin, Lise Land, Tim Mulhall, David Porter, Robert M. Taylor. Analysis for Knowledge-Based Systems - A Practical Guide to the KADS Methodology, Ellis Horwood, 1989

Hodgson, J. P. E. Knowledge Representation and Language in AI, Ellis Horwood, 1991

Ignizio, James P. Introduction toExpert Systems, 1991, McGraw-Hill

Ito, Hideaki. "Interface for Integrating a Knowledge-based System and a Database

Management System using Frame-based Knowledge Representation", The World Congress on Expert Systems Proceedings 1991, Vol. 2, Pergamon Press, 1991

Jelly, I. E. and J. P. Gray. "Common Architecture for Databases and Knowledge-Based Systems", The Next Generation of Information Systems: From Data to Knowledge, Springer-Verlag, 1992

Jenkins, Avery L. and Gene Grygo. "Expert Systems Pave Way for Knowledge Sharing", Digital Review, March 4, 1991

Kan, Sangki and Jung Wan Cho. "KPSP: A Knowledge Programming System based on Prolog", IEEE International Conference on Entity-Relationship Approach, IEEE Computer Society Press, 1985

Lazimy, Rafael. "Knowledge Representation and Modeling Support in Knowledge-Based Systems", Entity-Relationship Approach, North-Holland, 1988

Mannino, Michael V. "Data Bases Versus Knowledge Bases: Which Approach is Good for What?", Entity-Relationship Approach: A Bridge to the User, North-Holland, 1989

Mattos, Nelson M. "Abstraction Concepts: The Basis for Data and Knowledge Modeling", Entity-Relationship Approach: A Bridge to the User, North-Holland, 1989

Mattos, Nelson M. An Approach to Knowledge Base Management, Springer-Verlag, 1991

Moriarty, Terry. "The Next Paradigm", Database Programming & Design, February 1993

Navathe, Shamkant B. and Mohan K. Pillalamarri. "OOER: Toward making the E-R Approach Object-Oriented", Entity-Relationship Approach: A Bridge to the User, North-Holland, 1989

Orman, Levent V. "Multilevel Design Architecture for Knowledge-Base Management Systems", Applied Artificial Intelligence, April-June 1992

Parsaye, Kamram and Chignell, Mark. Expert Systems for Experts, 1988, John Wiley & Sons

Rodriguez, Rita V., Frank D. Anger and Douglas D. Dankel II. "Efficient Expert-

System Rule-Based Management via Relational Database Techniques", Advances in Artificial Intelligence, Vol. 1, JAI Press, 1989

Sandifer, Alice and von Halle, Barbara. "Linking Rules to Models", Database Programming & Design, March 1991

Vrtacnik, M., D. Dolnicar, A. Cizerle, P. Cok, S. A. Glazar, and R. Olbina. "Design of an Expert System for Water Pollution Determination/Prevention", The World Congress on Expert Systems Proceedings 1991, Vol. 2, Pergamon Press, 1991