KNOWLEDGE BASE FOR DATABASE DESIGN

J.J.Korczak, L.A.Maciaszek, G.J.Stafford

The University of Wollongong
Department of Computing Science
P.O.Box 1144, Wollongong, N.S.W. 2500, Australia

ABSTRACT

This paper describes the database design knowledge of a computer-assisted database design tool, called the Intelligent Database Design Kit (IDDK), used to develop database systems. The IDDK knowledge base is stored in the form of a database dictionary based on an extended entity-relationship-attribute model. The database dictionary is central to the knowledge engineering approach of the IDDK implementation strategy. The knowledge-based dictionary is a repository of meta-data (atomic facts, concepts, definitions, inference rules, heuristics) for a database system development. IDDK tools are developed using, to a large extent, semantic models and artificial intelligence techniques. The paper introduces computer-assisted extensions to Chen's entity-relationship model, in particular, recursive, subset, generic, and nested relationship concepts. An architecture and framework for intelligent database design is presented. The database design knowledge is divided into a declarative part (containing a description of entities, relationships and attributes) and a procedural part (defining design operations). While driving the IDDK tools, the knowledge base maintains basic consistency checks. In fact, the same methodology that supports the database design activities is used to implement the knowledge base. The IDDK interface to the knowledge base editor is based on a standard Macintosh interface extended with database design-oriented pallette icons and graphical editing tools.

Categories and Subject Descriptors: D.2 [Software Engineering]: D.2.1 Requirements/Specifications, D.2.10 Design; H.2 [Database Management]: Data dictionary/directory; I.2 [Artificial Intelligence]: I.2.4 Knowledge Representation Formalisms and Methods, I.2.5 Programming Languages and Software.

Additional Keywords and Phrases: Aggregation, Conceptualization, Data Dictionary, Declarative Knowledge, Entity-Relationship-Attribute Model, Expert System, Generalization, Inference Engine, Knowledge Base, Procedural Knowledge, User Interface.

1. INTRODUCTION

Retrospectively, database design has relied upon human experience and judgement rather than mechanistic algorithms. The design task is usually performed by experts who obtain information about the users' needs through interviewing, examining existing documentation, and other traditional means. To this extent, the current approach suffers from two weaknesses: (1) it requires the use of a scarce resource - the expert database designer, and (2) the designer's knowledge of the application is necessarily second-hand and some of its intricacies are likely to be overlooked. In our research, we respond to these problems by providing: (1) an expert system based set of tools, (2) a large knowledge base as a repository for information about the application and design activities.

The Intelligent Database Design Kit (IDDK) is a knowledge engineering project which integrates database technology, software engineering, and knowledge-based techniques. A key concept in this research is the use of a semantically extended conceptual model (entity-relationship (Chen, 1976)) as the underlying formalism of the database design knowledge base, which in turn drives the IDDK tools. The paper extends the forthcoming ANSI and ISO standards for Information Resource Dictionary System *IRDS* (Dolk and Kirsch II, 1987; Goldfine, 1985). The extension is threefold: (1) it applies a knowledge-based technology to computer-assisted design environments, (2) it uses explicit database design knowledge, (3) it relates to the life-cycle of system development.

IDDK is concerned with an entire database development life-cycle: requirements analysis, conceptual, logical and physical database modelling, application software design, maintenance and evolution. The methodology is process-driven; data semantics are derived from the semantics of business functions and any performance-motivated refinements and modifications of data structures are validated against the specifications of functions. In an overall IDDK approach, the design begins with the identification of business processes and data flows. IDDK includes a tool to draw data flow diagrams. The diagrams are used to derive a first-cut conceptual structure, which is then refined and converted to a relational logical structure. IDDK keeps track of completed transforms and ensures the coordination among steps and the integrity of the logical structure being derived.

The emphasis of this paper is on the database design knowledge of IDDK. The IDDK knowledge base is expressed as a semantic net that is an extension of Chen's entity-relationship model (Chen, 1976). IDDK knows the fundamental concepts such as entity, relationship, attribute, as well as the extensions such as recursive relationship, subset relationship, generic relationship, nested relationship, denormalized object, and modeling heuristics. An extended entity-relationship-attribute (ERA) model, built in IDDK, offers mechanisms for the representation and organization of knowledge. In contrast to application-oriented knowledge, database design knowledge - once customized - is not expected to undergo changes. The design knowledge for application development is unique to IDDK and relatively static.

The database design knowledge is obtained from database experts and encoded in IDDK. The design knowledge is divided into declarative and procedural knowledge (Frost, 1986; Keller, 1987). Declarative knowledge emphasizes the "static" aspect of database design - entities, relationships, attributes. The procedural knowledge emphasizes the "dynamic" aspect of database design and defines how to use the declarative knowledge in the design process. The declarative knowledge and procedural knowledge are integrated in a common framework and can be used for various phases of the system life cycle.

The paper is organized as follows. In the next section, the framework for intelligent database design is discussed. A detailed diagram of the IDDK tools for the database design life cycle is also introduced. In Section 3, our approach to database design knowledge is described. The description of the knowledge base editor is given and the structure of declarative and procedural knowledge for database design is presented. Simple examples are used to illustrate some fine points of a knowledge-based approach to database design.

2. FRAMEWORK FOR INTELLIGENT DATABASE DESIGN

2.1. Principles of IDDK Development

IDDK tools are being developed with the use of semantic models and artificial intelligence techniques. The semantic component is centered around the abstraction mechanisms (Smith and Smith, 1977). Recent advances in semantic modeling are taken advantage of (Brodie and Mylopoulos, 1986; Maciaszek, 1989; Stachowitz, 1985; Su, 1985). The applied artificial intelligence techniques concern mainly the knowledge representation formalism, reasoning and knowledge acquisition methods (Frost, 1986; Keller, 1987; Michalski *et al.*, 1983).

The IDDK design methodology covers all phases of the database development. We have identified seven phases of the database design life cycle:

1. *Requirements analysis and specification* (strategic planning, tactical modelling, document flows, defining data flow
2. *Conceptual modeling* (functions specifications, derivation of design ranks, view integration, clustering of attributes, defining entities and relationships, using abstractions and
3. *Design of the logical schema* (derivation of feasible logical structure, designing logical objects (records, sets, base and view tables, data items, etc.), verification and refinement procedures, logical schema definition);
4. *Design of the the physical schema* (derivation of feasible physical structure, gross and fine placement, access path optimization, space requirements, performance prediction, physical schema definition);
5. *Programming of user applications* (designing interactive and batch applications, enforcing semantic integrity of the database, deriving programs from database structures and business functions, screen (form) and report generation, programming in the host language environment);
6. *Maintenance of the database* (security, integrity, recovery, backup, authorization, auditing, tuning);
7. *Evolution* (restructuring, reorganization, application software conversion).

Of the seven phases, requirements analysis and specification and conceptual modeling are independent of the DBMS chosen for the database development. The remaining five phases apply separately for the relational, network, and micro environments.

The overall design process is iterative. Feedback is expected and complied with. As an illustration, the general design procedure at the conceptual level can be defined as follows:

```
...
set database_design_knowledge;
read application_knowledge_state;
S = application_knowledge_state;
...
procedure application_domain_design(S);
    var design_unfinished, stop_design: boolean;
    design_unfinished ← true;
    stop_design ← false;
    while design_unfinished
    begin
        identify a set T of applicable
        transitions on S
         T={T-entity, T-relationship, T-attribute,
            T-connection, T-editing}
        read selected_transition t(i);
        if t(i) ∈ T
            then
                begin
                    apply t(i) on S → new_S;
                    S ← new_S;
                end;
            else
                send an error_message;
        read stop_design;
        if stop_design then
            begin
                save S in
                application_knowledge_state;
                design_unfinished ← false;
            end application_domain_design;
    end;
```

Each step of the database design process can be characterized by a state and a set of applicable transitions. The are many ways in which one can select and apply transitions on a current state of a conceptual schema design. The simplest way is to evaluate the conditions in all rules with respect to the curent state of database design.

States and transitions are constrained. This means that the state which is obtained when transition t occurs has to satisfy the appropriate constraints. In general,

$$(S_i(s_i/c^s{}_i) \wedge F_i(t_i/c^t{}_i)) \rightarrow S_{i+1}(s_{i+1}/c^s{}_{i+1})$$

where S_i denotes a set of asseıtions in the database dictionary in step i, $F_i(t/c^t)$ is a set of formulas affected by the transition t_i, c^s and c^t are sets of constraints on states and transitions respectively.

2.2. IDDK Architecture

IDDK consists of five modules. Each module contains specific and independent information, data structures and procedures.

The modules are: Knowledge Acquisition, Database Design Knowledge, Application Domain Knowledge, Inference Engine, and User Interface (Figure 1).

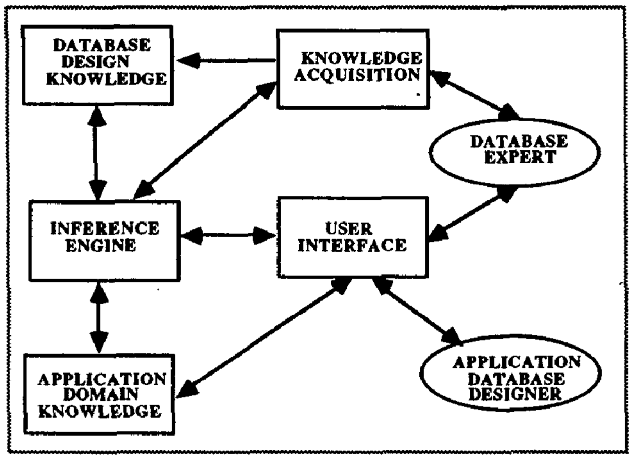

DATABASE DESIGN KNOWLEDGE
KNOWLEDGE ACQUISITION
DATABASE EXPERT
INFERENCE ENGINE
USER INTERFACE
APPLICATION DOMAIN KNOWLEDGE
APPLICATION DATABASE DESIGNER

Figure 1 IDDK Architecture.

The *knowledge acquisition* module is in charge of extracting database design knowledge and submitting it to the knowledge base module. The IDDK knowledge base consists of two types of knowledge which differ in content, in acquisition method and in usage. The first, database design knowledge is acquired by being programmed by a knowledge engineer (database expert). The design knowledge is defined as a set of design states and transitions; both restricted by a set of constraints. The second type of knowledge, application domain knowledge is acquired by being "drawn and typed". This is a form of learning from instruction (Michalski *et al.*, 1983). Acquiring domain knowledge from an application database designer, requires that the IDDK programs interpret the text typed or graphical symbols used, and transform them into an internal representation. The IDDK inference engine performs inferences on user-defined information, checks integrity constraints, and finally augments the existing application knowledge. The knowledge editor provides features to create, modify and document application knowledge. The editor aids the designer in organizing knowledge and supporting incremental acquisition. A database designer can instruct the system to change, validate or refute information it has been told previously. In IDDK, a set of readily understandable questions and help pages is implemented for eliciting knowledge from the system designer. The user interface of the knowledge editor is presented in Section 3 of the paper.

The IDDK *database design knowledge* is a collection of concepts, objects, integrity constraints, rules and operations that apply in the database design. The database design knowledge is divided into two parts: declarative (static) knowledge and procedural (behavioral) knowledge. The static knowledge is represented by means of an enhanced entity-relationship-attribute (ERA) model. The semantics of the knowledge base, in enhanced ERA representation, are declarative. However, as a semantically poor relational database technology is used to implement the knowledge base, some of these declarative semantics are implemented in procedural database triggers and demons (and moved to the inference engine). As a result, a limited and carefully monitored volume of deduction is traded for calculation (unavoidable phenomenon in any large knowledge-based system, for feasibility and performance reasons). The procedural knowledge refers to modeling of design actions and to keeping track of the database design processes. Especially, it incorporates the side effects of actions and the consistency checks. The knowledge editor (IDDK, 1988) applies to these two parts of database design knowledge. Another tool of IDDK is used to achieve a first-cut conversion of the domain knowledge from its conceptual model to a relational schema (Maciaszek *et al.*, 1988). The database design knowledge is discussed in the next section of the paper.

The *application domain knowledge* contains the descriptions of concepts, objects, integrity constraints, rules and operations concerning a given application domain. In other words, it contains the results of IDDK-controlled database design process. The application knowledge is also composed of declarative and procedural knowledge. That knowledge is application domain specific, i.e. a separate database is maintained for each database design project. There is only one body of design knowledge for a customized installation of IDDK. However, multiple domain-centered knowledge can be maintained by a designer who uses a customized IDDK (the customization will normally be necessary to provide the user with a required subset of IDDK tools, e.g. different subsets will be required for the designers of relational and network databases). Development of application domain knowledge is controlled by the inference engine using the database design knowledge. At the time of writing, a number experimental domain knowledge bases (e.g. bank customer services, inventory control, etc.) have been developed using IDDK and implemented under Oracle.

Before the presentation of the IDDK inference engine, let us consider structural and behavioral aspects of database design. The database design knowledge has relatively few rules and facts compared with the number of activities, rules and facts in an application domain. It would seem appropriate to consider the rules which effectively govern the way in which a database design state may be transformed into another. The database design knowledge can be regarded as a hierarchy of classes, where classes are defined as being: objects (entities and relationships), connections or attributes. These classes can be related to each other in various but definite ways to result in a semantic model of a given application. Note that the classification helps to improve the efficiency of reasoning by reducing the search space. For example, the operation of deleting the last attribute from a regular relationship re-classifies the relationship to weak. In database design, there is a need to store and manipulate a large amount of domain knowledge.

The IDDK *inference engine* uses the sets of axioms, contraints and functions in the database design knowledge to control the design process. As pointed out, each step of the database design process can be characterized by a state and a set of applicable transitions. In the IDDK project, the inference engine validates a transition with respect to the database design axioms and assertions in the database dictionary (this is the database design knowledge and the application knowledge). The inference engine is data-driven (forward-chaining) since the database design state is the sole identifier of applicable transitions. The inference engine can generate a tree of database design states by applying transitions, branching out from the input state and data. In a forward-chaining inference engine, it is difficult to control the direction in which the inference is conducted, because no explicit goals are defined. In many design tasks, IDDK makes temporary assumptions which allow pursuit of a set of possible solutions. Such assumptions may later be validated or invalidated. Non-monotonic reasoning is appropriate in the database design process, because the application domain is changing and incomplete. Some further information about axioms, constraints and operations of database design knowledge is given in the next section.

The IDDK *user interface* fully adheres to the Macintosh software development environment. This means an extensive use of windows, pull-down and pop-up menus, scrolling, scaling, mouse, etc. The user interface is oriented toward the database design process. It contains design-oriented pallette icons and graphical editing tools, tightly coupled with the data dictionary. This is a WYSIWYG user interface.

3. DATABASE DESIGN KNOWLEDGE

3.1. User Interface to Knowledge Base Editor

The knowledge base editor is an extension to the standard graphical Macintosh interface. The extensions concern database design-oriented pallette tools and menus. The dictionary stores definitions of attributes, entities and relationships, as well as cross-references between them. The changes to the data dictionary which affect the conceptual diagram are automatically reflected in the diagram (e.g. adding attributes to a weak relationship makes it regular). All changes in the diagram are recorded in the dictionary. The dictionary also enforces basic consistency checks (e.g. it does not allow duplicate names or direct connection of entities). The data dictionary has its own multiple-window user interface.

The user interface is tailored for ease of use (Figure 2). Picking the tool required, pointing at the desired position on the diagram screen, and clicking the mouse button, is all that is required to create an object. At the time of creation a meaningful name can be given to the object by simply typing the appropriate name.

The connection tools support establishing basic relationship connections. A connection can be made between a relationship oval and an entity rectangle or between two relationship ovals (the latter creates a nested relationship set). It is not possible to draw a connection line directly between two entity sets. Once established, the connection remains fixed for all editor operations, except for deleting a connected object or explicitly cutting the connection. A designer may define both partial and total membership of object sets in a relationship. The conectivity (1:1, 1:N, M:N) is indicated by means of a semicircle, rather than the conventional arrow-head.

A database designer may use two classes of abstraction. The aggregation connection tools support the aggregation/decomposition abstraction (black circles). The second abstraction, generalization, is implemented by means of black rectangles attached to subtype entity sets.

The editor provides an easy iconic way of manipulating objects in the diagram screen. The tools allow the user to relocate object boxes, slide the diagram on the paper, cut connection lines, delete objects, create and view sub-diagrams (user views). The show/hide tools permit the creation of multiple user views from a diagram. This feature is particularly useful for scheduling teamwork system development. With this feature, one can create sub-diagrams (sub-schemas), which can be further developed and used by other team members.

The IDDK knowledge editor uses seven menu bars. The first one is the standard Apple menu bar and it will not be described here. The File and Edit menu bars adhere to Apple requirements for such menus but are customized to serve editor purposes. The remaining four menu bars (DD, Project, Options, and Help) are provided to satisfy some typical knowledge editing functions.

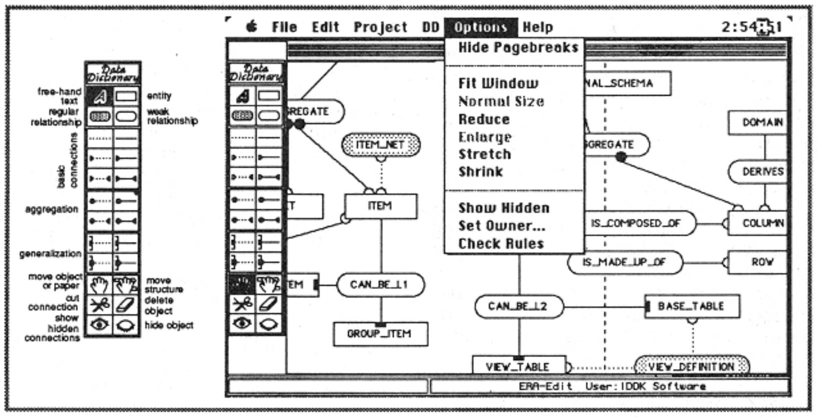


Figure 2 User interface of the knowledge editor.

The data dictionary tool provides quick access to any relationship or entity. The object creation tools allow creation and to some extent modification of entities and relationships. All editing tools (except those that establish connections) can be used against freehand text.

To create an object, one of the object editing tools has to be selected. Selecting the entity or relationship tool and clicking anywhere in the diagram screen, creates an entity or relationship set with a default name. The object remains selected to allow a new name to be typed in.

The Data Dictionary (DD) menu is a primary mechanism of entering the object definitions and the only mechanism of entering the attribute definitions in the data dictionary. From the DD menu, a multi-window editing environment is made available. It provides a means of entering and modifying definitions of attributes and of assigning attributes to entity sets and regular relationship sets (Figure 3).

The editor allows adition or deletion of attributes (simple or group) to/from an entity or relationship set. An attribute is added to an object by grabbing and moving it from the box of

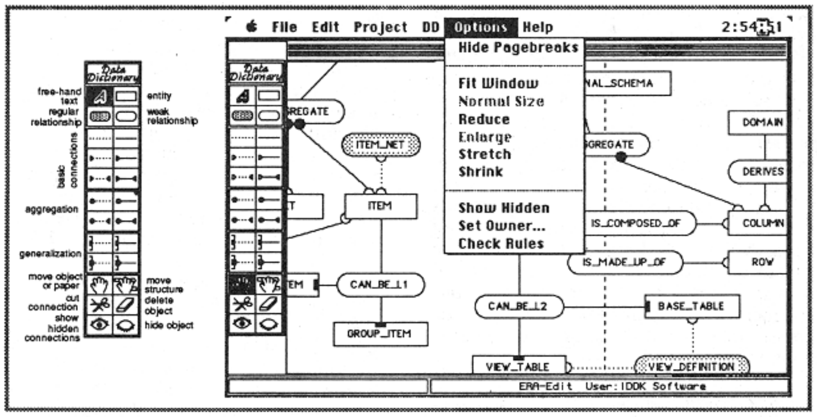


Figure 3 Example of object edition.

available attributes to the box of the object content. By moving in the opposite direction, one can remove an attribute from the object. The editor provides for a primary key and up to three candidate keys for an entity set (or a regular relationship set, if applicable). A professional-quality set of documentation of the database design knowledge base as well as application domain knowledge base can be automatically generated by the editor. Such documentation is delivered in the form of a "ready-to-bind" manual, with a title page, table of contents, etc.

3.2. Declarative Knowledge - Entities, Relationships and Attributes

The declarative knowledge of IDDK is expressed as a hypersemantic ERA model (Potter and Trueblood, 1988) and implemented as an ORACLE database. The representation of the declarative knowledge is a multiple-inheritance hierarchy of the objects connected by the *is-a* and *part-of* links (Figure 4). The *is-a* links denote the generalization/specialization hierachies. The *part-of* links express the aggregation/decomposition hierarchies.

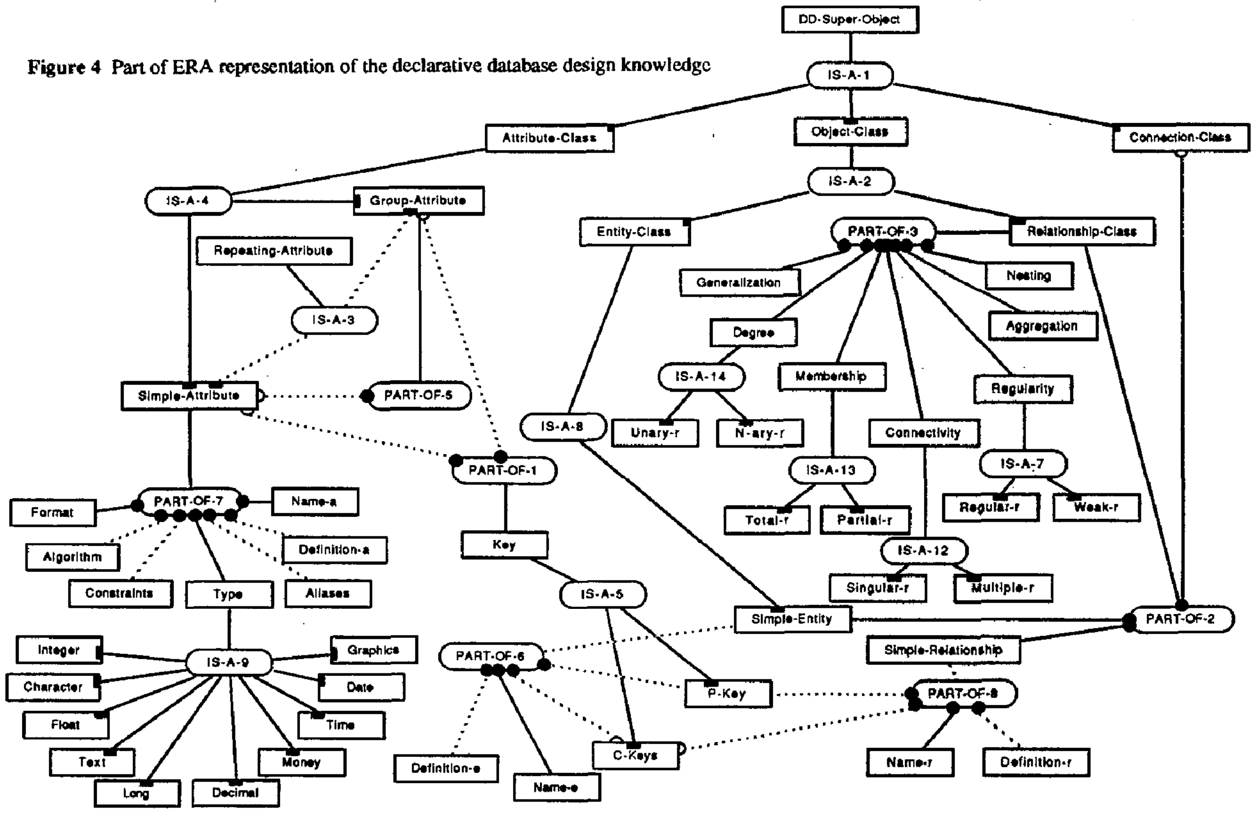


Figure 4 Part of ERA representation of the declarative database design knowledge

In Figure 4 ovals represent relationships, rectangles - entities. The total membership of an entity in a relationship is shown by a solid line that connects the two. A partial membership is represented by a dotted line. Plain lines are used to express 1 (singular) connectivity. To represent M (multiple) connectivity, the semicircle is attached to an M object.

Generalization turns a class of objects (usually entities) into a generic object (usually an entity). The reverse of this is called *specialization*. For instance, in Figure 4 the entity Object-Class is regarded as a generic *(supertype)* entity for the class of entities Entity-Class and Relationship-Class. We use such a generalization to ignore individual differences between *subtype* entities. A relationship that relates a supertype entity to subtype entities is called a *generic relationship*.

Black rectangles attached to subtype entities are used to denote a generic relationship. They also indicate that subtype

The *aggregation* transforms a relationship between objects (usually entities) into a higher level, aggregate *(superset)* object. The reverse of aggregation is called *decomposition*. For instance, in Figure 4 the subset entities Connection-Class, Simple-Entity and Simple-Relationship can be abstracted into a superset entity Relationship-Class. We can make such an aggregation to ignore details about the subset entities. For example, we want to think about a Relationship-Class without bringing to mind such details as what are the candidate keys, if any, of a relationship.

The existence of an aggregation is indicated by black circles attached to an aggregate relationship. All subset entities have connection lines that end with the black circle. This also represent an upward attribute inheritance mechanism of aggregation (i.e. from subset entities to a superset entity).

Clearly, there is an important difference between the inheritance mechanisms of aggregation and generalization. Contrary to aggregation, the inheritance of attributes in generalization is *downward* (top-down). Aggregation and generalization can be applied to composite objects to form aggregates and generics. The root of the knowledge hierarchy, called DD-Super-Object, is a generic object representing all objects, attributes and connections in the knowledge base. The root must exist in order for the axioms of database design knowledge to be hierarchically defined and satisfied.

As an example, a part of the declarative knowledge about objects and attributes is of the form:

*Axiom 1: There exists an entity with the name **E** which is found at the point **P** on the diagram.*

*Axiom 2: There is a connection of type **T** from relationship **R** to an entity **E**.*

*Axiom 3: There exists an attribute with the name **A**.*

*Axiom 4: The attribute **A** has the format **F** and constraints **C**.*

*Axiom 5: The attribute **A** is found in the entity **E**.*

etc.

The declarative knowledge provides the derivation of not explicitly stored theorems (data) from those stored in axioms. It assures the integrity-preserving knowledge base manipulations and it maintains the consistency of the database design knowledge. It also describes all potentially available operations which may be applied to transform the declarative knowledge, e.g. the specification of pre-conditions to add an attribute to an entity.

3.3. Procedural Knowledge - Database Design Transitions and Consistency Enforcements

Current methods and techniques of artificial intelligence and expert database systems do not allow for purely declarative construction and manipulation of complex knowledge bases. In the database design area, the deductive capabilities of the declarative knowledge are limited to those design aspects for which production rules (*if-then* clauses) can be stated. The procedural knowledge is used to fill the gaps in all these aspects of the database design process in which extensive calculation, follow-up integrity-enforcement operations, and human intervention are needed.

The models used in the design are: Business Model, Data Flow Diagrams Model, Entity Model or Entity-Relationship Model, Relational or Network Model, Application Design Model, Design Recovery and Reverse Engineering Model, and a few others.

Design process (major input-output transformations):

Problem Statement → Strategic Plan → Business Model

Business Model → (Document Flows, Implementation Plan, Data Flow Diagram)

Data Flow Diagram → (Entity Model, Functions Specs, Entity-Relationship Model)

Entity-Relationship Model → Normalized Entity-Relationship Model → Relational Logical Model → Prototype System

Entity-Relationship Model → Network Logical Model

Functions Specs → Structure Charts → Screen Painting → Program Generation

Logical (Network or Relational) Model → Physical (Network or Relational) Model

Application Design Model → Operational System → Design Recovery and Reverse Engineering Model

In order to exemplify the procedural aspect of the IDDK knowledge base, suppose that we are creating application domain knowledge (for a specific application). At the begining of the design (state 0), the database design knowledge contains:

(1) the declarative database design knowledge (axioms and integrity constraints), and

(2) the procedural database design knowledge (the set of allowed operations, e.g., *add_entity(E), add_relationship(R), add_attribute(A), delete_entity(E),*

Suppose that in a given application design state at the conceptual level, we have already defined the entity *e* with a group attribute *g* which in turn contains the attribute *s*, and a relationship *r* with the attribute *s*. In the next step, the designer wants to remove the attribute *s*. The diagram of the design situation is illustrated in Figure 5.

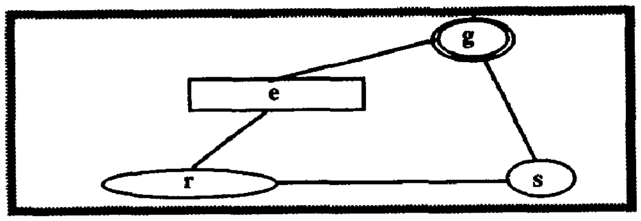


Figure 5 Example of an application design situation at the conceptual level.

To remove the attribute the system uses the procedural knowledge relating to the delete operation. This knowledge contains descriptions of operations and constraints which have to be satisfied to apply the *Delete_attribute* operation. The definition of the *Remove_attribute* procedure is as follows:

```
procedure Remove_attribute(a)
begin
     if Inused(a)
          then Delete_attribute(a)
          else
               begin
                    while g = Group_uses(a)
                         begin
                              if Last_reference(g)
                                   then
                                   Remove_attribute(g)
                                   else
                                   Delete_reference(g,a);
                         end;
                    while o = Object_uses(a)
                         begin
                              Delete_reference(o,a);
                         end;
                    Delete_attribute(a);
               end;
end.
```

The internal representation of our diagram (Figure 5) in the design state before *Remove_attribute(a)* is illustrated in Figure 6.

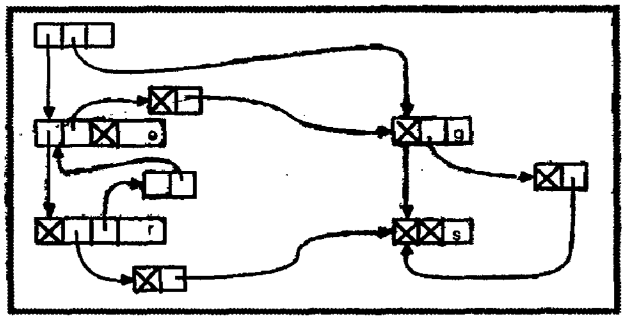
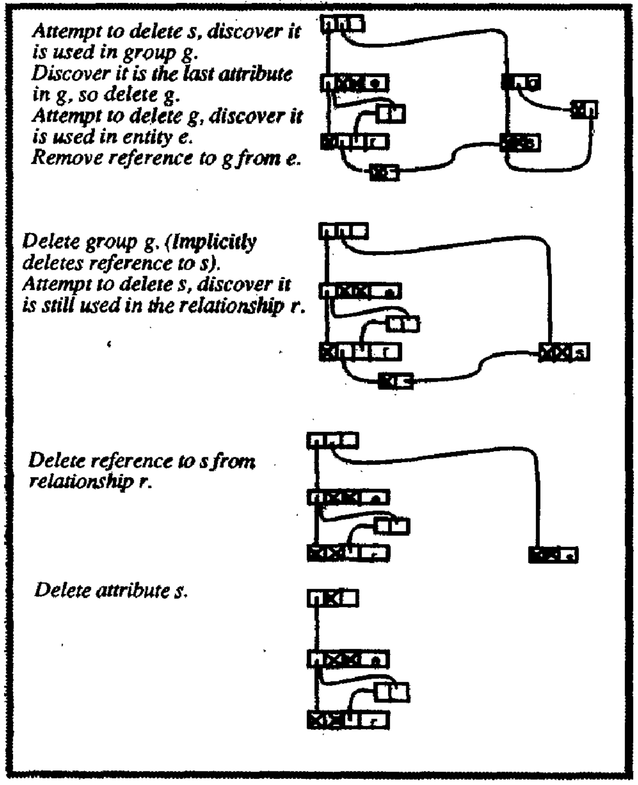


Figure 6 The internal representation of the design situation.

The process of removing the attribute *s* is illustrated diagramatically in Figure 7.

The database design methodology that underlies IDDK is process-driven. This means that the design begins with the specifications of user processing requirements. These specifications are then used to validate most intermediate design results. The seminal idea is that the database system should be closely tailored to specific user needs, rather than correspond to a vaguely understood concept of the "nature of data" (expressed in terms of various data dependencies). Although in IDDK the data dependencies do not drive the design, they are used to validate and enhance some data structuring decisions. The process-driven approach of IDDK has a direct impact on the procedural database design knowledge, as seen in the specific features:

* a methodology extending over the entire life-cycle of a database system;
* utilization of techniques intrinsic in data-oriented methodologies, such as normalization and abstraction, for design refinements;
* enhancements to entity-relationship model (e.g. relationship attributes, nested relationships, entity roles, generalization, aggregation);
* knowledge base which captures static and behavioral aspects of system design;
* strong design heuristics to eliminate, at early stages, paths of investigation that have little chance of success.
* various optimization tools to improve database performance (especially on the physical design level);



Attempt to delete s, discover it is used in group g.
Discover it is the last attribute in g, so delete g.
Attempt to delete g, discover it is used in entity e.
Remove reference to g from e.
Delete group g. (Implicitly deletes reference to s).
Attempt to delete s, discover it is still used in the relationship r.
Delete reference to s from relationship r.
Delete attribute s.

Figure 7 Using procedural knowledge in the *Remove_attribute* operation.

* adherence to ANSI'85 definitions for database language SQL (relational model) and Network Database Language NDL (network model) (Technical Committee X3H2, X3 Secretariat/CBEMA); applicabilty to the design of databases which provide relational user interface on top of network data structures (such as IDMS/R, IDMS/SQL, or RDMS-1100).
* Macintosh workstation graphics, Unix and relational DBMS interface; graphical and textual output of database schemas (conceptual, logical, physical); versatile reports; support for derivation of database programs; redesign support and significant propagation of design changes.

4. CONCLUSION

Due to the popularity and successes of computer-assisted software engineering tools, there is a growing need to provide a foundation for constructing a new category of knowledge-based CASE/CADE systems. We believe that our work is a step in this direction. We have described in this paper a framework for intelligent database design and a knowledge base for database design.

The development of a knowledge base to govern the database design process is a complex problem. The complexity is partly caused by the intrinsic requirement of describing the issues involved by means of themselves. This was evident in the paper: The knowledge editor serves not only to support the knowledge acquisition function of the IDDK (meta-data level), but it is also a stand-alone tool for entity-relationship-attribute modelling (data level). In the latter function, the tool is called *IDDK:ERA-Edit*™, is implemented in LightspeedC™ for the Macintosh™ and is now available commercially.

The bulk of the paper was concerned with the description of the declarative and procedural database design knowledge. A large part of the knowledge base has been implemented in a prototype form as an Oracle database and it is being experimentally used to drive existing tools of the Intelligent Database Design Kit. At the time of writing, *IDDK:ERA-Edit*™ is integrated at the "front-end" with *IDDK:DFD-Edit*™ (data flow diagramming tool) and at the "back-end" with *IDDK:LR-Derive*™ (converter from ERA design to relational database schema).

To achieve a full implementation of the knowledge base, further research and development work is needed to interface the knowledge base with the inference engine and knowledge acquisition methods. In particular, continuing work is being done on aspects of knowledge representation both in database design and application domain areas, including aspects of incomplete specification and non-monotonic logic. Some extensions and changes to the knowledge base schema are likely to be enforced by the development of successive IDDK tools.

REFERENCES

BRODIE, M.L. and MYLOPOULOS, J. (eds.) (1986): *On Knowledge Base Management Systems*, Springer-Verlag.

CHEN, P.P-S. (1976): The Entity-Relationship Model - Toward a Unified View of Data, *ACM Trans. Database Syst.*, 1, pp.9-36.

DOLK, D.R. and KIRSCH II, R.A. (1987): A Relational Information Resource Dictionary System, *Comm. of the ACM*, 1, pp.48-61.

FROST, R.A. (1986): *Introduction to Knowledge Base Systems*, Collins

GOLDFINE, A. (1985): The Information Resource Dictionary System, *Proc. 4th Int. Conf. on E-R Approach*, Chicago, USA,

IDDK (1988): *IDDK:ERA-Edit Fundamentals*, Version 1.0, IDDK Software, p.33.

KELLER, R. (1987): *Expert System Technology, Development and Application*, Yourdon Press Computing Series

MACIASZEK, L.A. STAFFORD, G.J. HAYWARD, J.J. HAYWARD, M.R. and KRAV, S.I. (1988c): A CADE Tool to Derive a Relational Database Structure from an Enhanced Conceptual Design, *Proceedings Australian Software Engineering Conference - ASWEC'88*, Canberra, Australia, pp.137-154.

MACIASZEK, L.A. (1989): *Database Design and Implementation*, Prentice-Hall (to appear).

MICHALSKI, R.S, CARBONELL, J.G. and MITCHELL, T.M. (1983): *Machine Learning*, Tioga Publ.

POTTER, W.D. and TRUEBLOOD, R.P. (1988): Traditional, Semantic, and Hyper-Semantic Approaches to Data Modeling, *Comp.*, June, pp.53-63.

SMITH, J.M. and SMITH, D.C.P. (1977): Database Abstractions: Aggregation and Generalization, *ACM Trans. Database Syst.*, 2, pp.105-133.

STACHOWITZ, R.A. (1985): A Formal Framework for Describing and Classifying Semantic Data Models, *Inform. Syst.*, 1, pp.77-96.

SU, S.Y.W. and RASCHID, L. (1985): Incorporating Knowledge Rules in a Semantic Data Model: An Approach to Integrated Knowledge Management, *Artificial Intelligence Applications, The Engineering of Knowledge-Based Systems, Proc. 2nd Conf.*, ed. C.R.Weisbin, IEEE CS Press/North-Holland, pp.250-256.