

a quarterly bulletin
of the IEEE computer society
technical committee
on

Database Engineering

Contents

Cooperation Between IEEE TC on DBE and ACM SIGMOD	1	IRMA: An Automated Logical Data Base Design and Structured Analysis Tool	40
Liaison Person Needed: ACM SIGMOD—IEEE Technical Committee on DBE	2	R.M. Curtice	
Letter from the Editor	3	An Overview of Research in the Design of Distributed Databases	46
Information System Design at the Conceptual Level—The Taxis Project	4	S. Ceri, B. Pernici, and G. Wiederhold	
J. Mylopoulos, A. Borgida, S. Greenspan, and H.K.T. Wong		Current Research in Database Design at the University of Minnesota	52
The Database Design and Evaluation Workbench (DDEW) Project at CCA	10	S. March, S. Mendu, P. Palvia, M. Prietula, D. Ridjanovic, J.V. Carlis, D. Beyer, and K.L. Ryan	
D. Reiner, M. Brodie, G. Brown, M. Friedell, D. Kramlich, J. Lehman, and A. Rosenthal		Research on Form Driven Database Design and Global View Design	58
Database Design Activities Within the DATAID Project	16	M.V. Mannino and J. Choobineh	
C. Batini, V. De Antonellis, and A. Di Leva		A Prototyping Approach to Database Applications Development	64
A Realistic Look at Data	22	A. Albano and R. Orsini	
W. Kent		A Causal Approach to Dynamics Modeling	70
Tools for View Integration	28	V. De Antonellis and B. Zonta	
R. Elmasri, J.A. Larson, S. Navathe, and T. Sashidar		Designing Database Updates	76
RED1: A Database Design Tool for the Relational Model of Data	34	S. Salveter and D.E. Stumberger	
A. Bjornerstedt and C. Hulten		Calls for Papers	82

Special Issue on Database Design Aids, Methods, and Environments

**Chairperson, Technical Committee
on Database Engineering**

Prof. Gio Wiederhold
Medicine and Computer Science
Stanford University
Stanford, CA 94305
(415) 497-0685
ARPANET: Wiederhold@SRI-AI

**Associate Editors,
Database Engineering**

Dr. Haran Boral
Microelectronics and Computer
Technology Corporation (MCC)
9430 Research Blvd.
Austin, TX 78759
(512) 834-3469

Prof. Fred Lochovsky
Department of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S1A1
(416) 978-7441

Dr. C. Mohan
IBM Research Laboratory
K55-281
5600 Cottle Road
San Jose, CA 95193
(408) 256-6251

Prof. Yannis Vassiliou
Graduate School of
Business Administration
New York University
90 Trinity Place
New York, NY
(212) 598-7536

**Editor-in-Chief,
Database Engineering**

Dr. David Reiner
Computer Corporation of America
Four Cambridge Center
Cambridge, MA 02142
(617) 492-8860
ARPANET: Reiner@CCA
UUCP: decvax!cca!reiner

Database Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Database Engineering. Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Database Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in the Database Engineering Technical Committee is open to individuals who demonstrate willingness to actively participate in the various activities of the TC. A member of the IEEE Computer Society may join the TC as a full member. A non-member of the Computer Society may join as a participating member, with approval from at least one officer of the TC. Both full members and participating members of the TC are entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Cooperation between IEEE TC on DBE and ACM SIGMOD

The IEEE Technical Committee on Database Engineering puts out this publication; ACM SIGMOD (Special Interest Group on Management of Data) publishes the SIGMOD Record. In orientation, goals, and membership, there is much in common between the two groups, and both publications are aimed at researchers and practitioners in the database area. To foster cooperation and cross-fertilization between these groups, SIGMOD has agreed to fund distribution of this issue of Database Engineering (DBE) to its members. The IEEE TC hopes to reciprocate in 1985 by distributing an issue of SIGMOD Record to its members.

A SIGMOD member wishing to receive future issues of Database Engineering may join the IEEE Computer Society, which allows him to join any of a number of TCs. Alternatively, he may join just the TC on DBE, as a correspondent, at no cost (currently). (Write to: IEEE Computer Society, 1109 Spring St., Suite 300, Silver Spring, MD 20910.)

A TC on DBE member wishing to receive future issues of SIGMOD Record may join ACM and any of a number of Special Interest Groups (SIGs), including SIGMOD. (Write to: ACM, 11 West 42nd St., New York, NY 10036.)

If you already belong to both organizations, please: (1) pass your extra copy of this issue along to a colleague, and (2) notify David Reiner (see address on inside front cover) by mail (or netmail), using the tear-out form on the next page. This will help us keep down costs of duplicate mailings in the future.

Briefly, here are the differences between the two publications. DBE, published quarterly, focuses on a particular theme with each issue, and tends to contain mainly invited papers on current research and development efforts. Although submissions are not subject to a formal review process, the editors generally read articles very carefully, and work with the authors to achieve both clarity and brevity. Upcoming 1985 issues will treat DBMS Performance, Concurrency Control and Recovery in DBMSs, Natural Languages and Databases, and Object Oriented Systems and DBMSs.

The Record, published from two to four times a year, accepts submissions on a broad range of database-related topics, and prefers to see somewhat unusual articles which may not fit other forums such as the ACM SIGMOD Conference, the IEEE Data Engineering Conference (CompDEC), or VLDB. When possible, issues have a unifying theme. There is no formal review process.

We hope in the future to cooperate on one or more joint issues of these two publications, and plan to refer papers back and forth where appropriate. In the meantime, enjoy this issue whatever your affiliation.

David Reiner, Editor-in-Chief, Database Engineering

Jon Clark, Editor, SIGMOD Record

Liason Person Needed:
ACM SIGMOD - IEEE Technical Committee on DBE

We would like to follow up on the direction of greater inter-society cooperation announced by the ACM and the IEEE Computer Society. To make something actually happen, we are looking for an individual willing to be a focal point in this task. Such a person would be appointed to the ACM SIGMOD and the IEEE CS TC DBE as coordinator, or whatever reasonable title can be invented and supported.

Concerns will be shared publication efforts, conference sponsorship and schedules, and anything else which appears to be a positive step. Bea Yormark, chair of the ACM SIGMOD, has indicated her support. Please contact either one of us.

Gio Wiederhold, Chairperson,
Technical Committee on Database Engineering

[Gio's address and phone # are on the inside front cover; Bea's are 7503 Lynn Drive, Chevy Chase, MD 20815, (703)-836-2696.]

If you are or plan to become a member of both ACM SIGMOD and IEEE TC on DBE, please send a copy of this form to:

David Reiner
Computer Corporation of America
Four Cambridge Center
Cambridge, MA 02142

or notify Dave via netmail:

Arpanet: reiner@cca
UUCP: decvax!cca!reiner

Our purpose is to keep down costs of duplicate mailings.

Name _____

Address _____

Net address (if any) _____

Check all that apply:

	ACM SIGMOD	IEEE TC on DBE	IEEE Computer Soc
Currently a member	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Plan to join	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Letter from the Editor

Designing a database is a complex iterative process. It requires familiarity with design techniques, methods, heuristics, and tools, and with the nature of the data and its intended uses. In the last few years, database design research has been on the upswing. Researchers are concentrating not just on improved design tools, but also on the overall methodological framework of the design process, and on integrated environments to support it.

There seems to be general agreement that database design is best separated into a succession of related but independent steps, moving from the more abstract levels of requirements analysis and conceptual design, to the logical level where the target data model is introduced, to the more concrete levels of distributed and physical design (though the exact nature and numbers of these steps may vary slightly from one methodology to another). There is increasing interest in design systems which support many stages of database design, and which connect database design to the broader lifecycle of system design and evolution, including application development.

Current trends and areas of concentration in database design include: expanded data model semantics (including more emphasis on constraints and improved techniques for view integration), more attention to dynamic (transaction-oriented) aspects of designs, graphics workstation-based design environments (with interactive design tools), and AI-related approaches (expert systems technology, designing for natural language interfaces to databases). Productive work continues in rapid database prototyping, form-based design, and physical and distributed design. Commercial database design products are beginning to be available.

The first three papers in this special issue describe fairly comprehensive design environments now under development: the Taxis project at Toronto, CCA's DDEW project, and the Italian DATAID effort. The next four cover various aspects of conceptual and logical design: the "fact-based" approach, view integration, the RED1 logical design tool from Sweden's SYSLAB, and ADL's logical designer. Several university research projects come next, on distributed design, physical design, and form-based design. The last three papers discuss early prototyping, modeling dynamic aspects of databases, and designing database updates in a natural language environment.

My thanks to the contributors to this issue, who have invested considerable effort to produce concise but readable summaries of their current research. ACM SIGMOD's support for printing and distributing extra copies of this issue for its members is also gratefully acknowledged.

David Reiner

David Reiner
Cambridge, Massachusetts
December, 1984

INFORMATION SYSTEM DESIGN AT THE CONCEPTUAL LEVEL -- THE TAXIS PROJECT

John Mylopoulos, University of Toronto¹
Alexander Borgida, Rutgers University
Sol Greenspan, Schlumberger-Doll Research, Conn.
Harry K.T. Wong, Lawrence Berkeley Labs

Abstract

This is a brief overview of the Taxis project, concerned with the development of languages, tools and methodologies for the design of interactive information systems (ISS) such as reservations, inventory control and credit card verification. We describe the major novel ideas explored in this project, the tools and techniques supporting them, and their source in significant ideas of Artificial Intelligence and Software Engineering.

1. INTRODUCTION

The development of database design techniques and tools is based on philosophical considerations concerning the nature of databases, their role and the major source of problems in their development. (Such philosophical underpinnings are of course not always made explicit.) In our case, there appear to be three fundamental observations whose logical conclusions have shaped the nature of our design aids and approach in general.

First, we consider the design of databases an integral and inseparable part of the design of Information Systems (ISs), which include transactions, user interfaces, etc. and hence have broadened the scope of our research to cover all aspects of Information System design, not just those dealing with the storage of data.

Secondly, as others, we view an IS as a *model* of some relevant portion of the "real world", or more accurately a model of the *end-user's (conceptual) perception of the world*. As with models in other fields, an IS is then useful to the extent that it reflects reality accurately, and to the extent that the information in it is easily accessible. One consequence of this axiom is that IS design is now viewed as *model development* or *modeling*. A second consequence is that IS development should then be much easier and more successful if it begins at the *conceptual level*, rather than the logical level. For this reason, the Taxis project has drawn inspiration from Knowledge Representation schemes current in Artificial Intelligence, and has adapted them to the specific problems of IS design. An underlying assumption here is that the mapping from the conceptual design to the more "physical"/"machine dependent" levels can be automated.

Finally, we view ISs as software, and hence believe that important precepts of Software Engineering are applicable to IS engineering. However, there is much to be gained by taking advantage of the fact that ISs form a strict subclass of all possible software, one subject to many restrictions and limitations.

Among the consequences of these views we list

- the applicability of the software life cycle to ISs. In particular, we advocate separate requirements specification and design phases, but emphasize the advantages of a *uniform philosophy, conceptual modeling*, underlying both;
- the importance of *abstraction* followed by *gradual refinement* as a fundamental tool for coping with the sea of minutiae which need to be captured in a model. ISs however provide an opportunity to explore new abstraction principles, which may not apply as successfully in the case of general software.
- the utility of *prototyping* as a technique for obtaining quick feedback in determining what the user really wants.
- the need for a *methodology of software engineering* to provide at least three things for each level

¹Address correspondence to Department of Computer Science, University of Toronto, Toronto, Ontario, Canada. (416) 978 5180. Netmail to jm@toronto.csnet, borgida@rutgers.arpa, greenspan@rutgers.arpa, or wong@lbl-csam.arpa.

of specification:

- o *languages* -- precise notations for expressing the relevant information.
- o *techniques* -- procedures for constructing, manipulating, and validating specifications.
- o *tools* -- automated aids designed to support the above.

2. THE TAXIS LANGUAGE

The focus of the Taxis Project is the Taxis programming language, which supports the description of ISs at the conceptual level ([MYLO80a, 80b], [WONG83]). Taxis incorporates a so-called "semantic data model" which provides for the description of the entities in the world and their inter-relationships through the notions of *objects*, related by *properties/attributes*. A major advantage of this object-centered framework over traditional record-based approaches is the direct and natural correspondence between the model and the world (e.g., no reliance on keys), which facilitates both the design and access of the IS. Individual objects are organized into classes, which describe commonalities of their instances in the form of constraints -- e.g., properties applicable to them, the valid ranges of values of such properties. In Taxis, classes themselves are objects, and hence can be members of *meta-classes*; therefore classes can have their own properties (e.g., aggregate information). Furthermore, classes are organized into a hierarchy with general classes located above their specializations. If one class (e.g. employee) is defined to be a specialization, or *subclass* of another class (e.g., person), then at all times every instance of the first is considered to be an instance of the second. An important consequence of this organization is that properties can be *inherited* from superclass to subclass, e.g., the class of employees inherits properties such as name, address, and so on, from the class of persons.

In addition to modeling data, Taxis supports the development of Information Systems by providing language features to model the *activities* in the world. For short-term activities, Taxis provides the notion of *transaction* familiar in databases as the basic unit of integrity and recovery maintenance. A transaction consists of an initial group of *preconditions* which check the applicability of the operation at this point, followed by a sequence of actions described in traditional procedural notation (e.g., assignments, loops, conditionals, data manipulations), and concludes with *post-conditions*. Taxis attempts to maintain uniformity and parsimony by casting transactions in the same mold as entities: a transaction is viewed as an object, with its parameters, conditions and actions becoming its properties; procedure definitions become class descriptions (hence procedure invocations are instances of these classes); and most innovatively, transactions are also organized into specialization hierarchies (e.g., admitting a surgical patient is a specialization of admitting a general patient).

In order to deal with special cases, Taxis incorporates a procedure-oriented exception handling mechanism, and in a recent extension, provides the ability to store information which does not fit the class-schema (see Section 6).

To model *persistent activities* -- i.e., activities with prolonged duration such as participating in a clinical trial or attending university -- Taxis also supports the notion of *scripts* ([BARR80,82], [PILO83a,83b], [CHUN84], [TAXI84]). A script is built around a Petri-net skeleton of states connected by transition arcs, which are augmented by condition-action rule pairs (viz. [ZISM78]). The rules are described in Taxis, but also allow reference to the passage of time, and permit the transmission of messages following Hoare's CSP mechanism. Scripts are integrated completely into the Taxis framework, so that script classes are organized into a subclass hierarchy according to their generality/specificity, have their states and transitions defined in terms of properties, and their instances can be accessed through the same facilities used to access instances of entity classes. This allows among others queries concerning the currently executing set of scripts.

Finally, an extension of Taxis allows designers to describe *user interfaces to ISs*, e.g., the query or interaction language, in the same uniform framework of objects with properties in classes. Such an extension allows a direct link to be established between the "referring expressions" used in the query language and their "referents" in the database. The extension described in [PILO83a,83b],[TAXI84] also provides for modelling tools to be used for the specification of a grammar and a lexicon which are integral components of any user interface.

3. THE RML LANGUAGE FOR REQUIREMENTS

A *requirements model* is a description of some portion of the world that encompasses potential information systems and is used to communicate and analyze the problem situation. (A model at this level corresponds to "Corporate Requirements".) In our case, it also provides a starting point for information system design using Taxis.

We have defined a language called the Requirements Modeling Language (RML) for the purpose

of describing requirements models ([GREE82,84], [TAXI84]).

A fundamental premise of the Taxis project is that one conceptual framework (such as the object oriented framework described above) can be used both at the design *and* the requirements specification level. What does change as a designer moves from one level to the other are the kinds of classes at his disposal as he constructs his specification, and the kinds of information which needs to be captured. For the design level these are data, transaction, exception and script classes. For the requirements level, on the other hand, they are classes of *individuals*, *activities* and *assertions*. Informally, classes of individuals correspond fairly directly to Taxis data classes. Activity classes are intended to model both instantaneous actions and long-term events, corresponding to transactions and scripts in Taxis. Finally, assertions are logical formulas making statements about the world, the rest of the requirements specification, or the relation between the two.

Abiding by the maxim that "the requirements should express WHAT the system does but not HOW", RML does not have the notion of "control flow"; instead, RML supports a *temporal view of the world*: properties of objects, and their existence are all related to a time line, and the designer must specify temporal constraints in order to ensure the desired sequencing of events, for example. RML does however provide abbreviation techniques such as "property categories" (e.g., "*initially*", "*always*") which shorten descriptions by removing the clutter of temporal indices. Furthermore, RML provides a mechanism for introducing and defining new property categories, allowing RML to be customized to a specific task.

In addition to higher-level descriptions of activities, RML provides the opportunity to model objects from both the world and the proposed IS, thereby allowing the specifier to express definitions of terms, such as units of measurements, which would not normally appear in the IS, and to specify constraints on the performance and accuracy of the IS.

4. METHODOLOGIES OF DESIGN

4.1. Dimensions of Abstraction

An abstraction mechanism is a conceptual or linguistic mechanism that allows certain information to be highlighted while suppressing other information. In software engineering, abstraction is usually equated with the suppression of design decisions or implementation detail. However, within a given level, the Taxis framework offers a set of complementary abstraction facilities based on the notions of *aggregation*, *classification*, and *generalization* [SMIT77].

If we define a *property* to be a directed relationship between two objects, aggregation allows one to view an object as a composite of the objects to which it is related by properties. For example, a person has a name, an address, and so on. The "abstraction" here is that one may talk about an object while choosing to ignore its components for the moment.

Along an orthogonal dimension, the classification abstraction allows individuals to be grouped into classes (and classes into *metaclasses*) that share common properties. By describing class objects, one abstracts away the the detailed differences of the class instances.

Generalization allows the *common properties of several classes to be factored out into the definition of a single, more general, class*. For example, the class of persons can be represented as a generalization of the classes representing males, females, managers, engineers, female engineers, and so on. The taxonomic organization provided by generalization hierarchies can lead to models that are understandable and consistent, because the more that classes have in common with each other, the "closer" they are located to each other in the hierarchy. Also, generalization hierarchies can lead to more concise models, since it is sufficient to associate properties to the most general, applicable class and let inheritance imply the rest.

4.2. Taxonomic Programming

A methodology for specification/modeling should provide guidance to its users. At the heart of many software development methodologies lies one or more abstraction mechanisms, which allow us to ignore details at some level, plus a *refinement principle* which provides for the guided and gradual introduction of details across the abstraction dimension.

We have explored the utility of the generalization abstraction as the basis of a methodology for building descriptions which we call *taxonomic programming* or *stepwise refinement by specialization* [BORG82]. Its main idea is that a model should be constructed by modeling first the most general classes, and then proceeding to more specialized classes. For example, in modeling a hospital world, one might consider first the concepts of patient, doctor, admission, treatment, etc. Later, the modeler can differentiate between child patients, heart patients, internists and surgeons, surgical and medical treatments, etc. At each step, only the information (properties) appropriate to that level are specified, and because of inheritance, only new facts need to be stated.

Generalization is the appropriate principle to exploit when the difficulty of modeling is due to a large number of details rather than due to algorithmic complexity; a hierarchy of classes organized along this dimension guides the attention of the designer, and provides a convenient structure for distributing information and associating it where it most naturally belongs. We emphasize that in Taxis, specialization is applicable not only to data objects but also to the description of activities, transactions, exceptions, and scripts.

5. DESIGN AIDS

5.1. Compilers and interpreters

Brian Nixon has implemented a compiler for Taxis programs [NIXO83]. The target language for the compiler is Pascal, augmented with relational database facilities such as those provided by Pascal/R [SCHM80]. Taxis data hierarchies are translated into relational schemata, and hierarchies of transactions are translated into the block structure of Pascal. The output of the compiler is a program containing definitions of all classes and transactions, routines to enforce constraints, a database interface (as provided by Pascal/R), and a near-empty database.

In developing such a compiler, several interesting implementation problems needed to be resolved, including the distribution of information about an object in multiple relations, the possibility of conflicting inherited properties, and the operation of inheritance for procedures. The implementation of the Taxis compiler has been extended to handle the execution of scripts [CHUN84]. The major problem in implementing scripts is the development of efficient algorithms for testing whether a state transition is ready to fire or whether the invariants associated with a script state (and expressed in terms of a logical assertion) are not being violated at a particular time.

Turning to prototyping facilities, an interactive environment for creating and trying out Taxis programs has been designed and implemented by Pat O'Brien [O'BRI82]. It includes a class-oriented editor, whose commands and functionality are centered on Taxis classes; a semantic consistency verifier which ensures that Taxis programs conform to the semantic rules of Taxis; and an interpreter and debugger for prototyping. The editor provides the information system designer with facilities to construct, inspect, and modify a Taxis program. The consistency verifier performs various checks to ensure the correctness of the conceptual model being specified. The interpreter simulates execution of Taxis programs, and the debugger assists the designer in validating the model. The design environment also provides various other aids to the user, such as an online help facility, a documentation generator, and a way of keeping track of multiple versions of models. The Taxis design environment is also being expanded to handle scripts [PARK84].

5.2. The connection of RML to SADTTM

The difficulty of building a high-level requirements specification as in RML should not be understated. In the initial stages of requirements definition, all of the parties involved are faced with the problem of deciding what concepts and phenomena are relevant to the situation at hand, agreeing on terminology, and conveying their "mental models" of the situation to each other. We propose, therefore, that requirements be defined in two steps:

- The first would use a language for *structured analysis* such as SADTTM [ROSS77], in which terms are introduced in an organized way.
- The second would use RML for *semantic modeling*, which gives definitions of the semantics of the concepts introduced in the first step.

In [GREE84], the connection between SADT and RML is made. SADT provides a way of introducing concepts/terms into the requirements specification by a process of stepwise decomposition (expanding a concept "box" into a "diagram" containing several interconnected boxes). The result is a hierarchically organized structure of interrelated terms, which provide a "structured lexicon", a sort of road map to guide the RML modeling process. --

RML is then used to express more formally the information usually expressed by natural language labels and comments on SADT arrows and boxes. In the process, the semantic relationships expressed in the RML model are constrained by the connectivity of the SADT diagram from which it is derived -- e.g., arrows connecting boxes become properties relating the corresponding classes.

In an attempt to validate our design methodology, Taxis was used for describing a medical information system for the Pacemaker Center at the Toronto General Hospital [DIMA83], which keeps track of patients who have received a cardiac pacemaker. It was also used to design a medical information system for managing clinical trials [BUCH82], which are controlled experiments for investigating the cause/effect relationship of new treatments. In both cases, Taxis' facilities for

describing scripts, exceptions and organizing classes in specialization hierarchies proved to be very important.

6. CONCLUSIONS AND CURRENT WORK

The Taxis Project has designed and implemented a variety of languages and tools for requirements and design. Although they draw on ideas that are popular in Artificial Intelligence and Data Base Management, as well as in some programming languages, they are all based on the same object-oriented framework, which uses three fundamental abstraction mechanisms to structure and organize information.

Software engineering is viewed as the construction of a series of models, starting with a world-oriented requirements model (SADT plus RML), then a Taxis design model, and ultimately a completely implemented system. The task of requirements modeling is likened to the task of knowledge representation in Artificial Intelligence, and the Taxis framework applies concepts that are popular in Artificial Intelligence (as well as in semantic data models) to both RML and the Taxis language.

Current work is proceeding both on RML and Taxis. The RML language is being extended in several directions: the uniform treatment of properties as objects; linguistic mechanisms for relating objects in the world and their images in the IS; allowing contradictory information to be introduced during specialization, thereby supporting a new abstraction principle: *normalization*. By providing a translation of RML into logic, and by connecting this to a specialized theorem-prover, we hope to allow reasoning about specifications, such as checking for consistency. We are also co-operating with a software house in adapting RML for "practical use".

The exception-handling facilities of Taxis have also been greatly extended to allowing *dynamic* exceptions -- i.e., allowing information to be stored which violates the schema of classes used during compilation. This mechanism allows an IS to be much more flexible in the face of variability in the world, especially unexpected occurrences, and can also be used to deal with such thorny problems as null values, conversions of measurements, estimates, etc. (see [BORG84], [TAXI84]).

Acknowledgments The following members of the Taxis project have contributed significantly to the advancement of the research reported here: Dr. P. Bernstein, J. Barron, B. Nixon, Dr. M. Pilote, P. O'Brien, I. Buchan, C. DiMarco, S. Park, L. Chung.

REFERENCES

- [BARR80] John Barron, *Dialogue Organization and Structure for Interactive Information Systems*. Technical Report CSRG-108, Computer Systems Research Group, University of Toronto, January 1980.
- [BARR82] John Barron, Dialogue and Process Design for Interactive Information Systems Using Taxis. *Proceedings, SIGOA Conference on Office Information Systems*, Philadelphia, PA, June 1982. *SIGOA Newsletter*, 3(1,2), pp. 12-20.
- [BORG82] A. Borgida, J. Mylopoulos, H.K.T. Wong. "Generalization as a Basis for Software Specification" in M.Brodie, J.Mylopoulos, J.Schmidt(eds.) *On Conceptual Modeling: Perspectives from AI, Databases and Programming Languages*, Springer Verlag, 1984.
- [BORG84] A. Borgida, "Language Features for Flexible Handling of Exceptions in Information Systems", Technical Note, Department of Computer Science, Rutgers University; submitted for publication.
- [BUCH82] I. Buchan, H. D. Covvey, J. Mylopoulos, C. DiMarco, and E. D. Wigle, "Taxis: A Language for the Development of Clinical Trial Management Systems," *Proc. Sixth Annual Symposium on Computer Applications in Medical Care*, October 1982. (also [TAXI84])
- [CHUN84] K. Lawrence Chung, *An Extended Taxis Compiler*. M.Sc. thesis, Dept. of Computer Science, University of Toronto, 1984.
- [DIMA83] Chrysanne DiMarco, *Using TAXIS to Design a Medical Information System*. (M.Sc. Thesis) Tech. note #31, Department of Computer Science, University of Toronto, 1983.
- [GREE82] S. Greenspan, J. Mylopoulos, and A. Borgida, "Capturing More World Knowledge in the Requirements Specification," *Proc. 6th International Conference on Software Engineering*, Tokyo, 1982. (also [TAXI84])
- [GREE84] S. Greenspan, *Requirements Modeling: A Knowledge Representation Approach to Requirements Specifications*, Ph.D. thesis, University of Toronto, 1984.

- [MYLO80a] J. Mylopoulos, P. A. Bernstein, and H. K. T. Wong, "A Language Facility for Designing Interactive Database-Intensive Applications," *ACM Transactions on Database Systems*, Volume 5, Number 2, June 1980, pp. 185-207.
- [MYLO80b] J. Mylopoulos and H. K. T. Wong, "Some Features of the TAXIS Model," *Sixth International Conference on Very Large Data Bases*, 1-3 October 1980, pp. 399-410.
- [NIXO83] Brian Nixon, *Translating Taxis Programs*, M.Sc. Thesis, Dept. of Computer Science, University of Toronto, 1983.
- [O'BRI82] Patrick D. O'Brien, *TAXIED: An Integrated Interactive Design Environment for TAXIS*. (M.Sc. Thesis) Tech. note #29, Department of Computer Science, University of Toronto, 1982.
- [PARK84] Sun G. Park, *Implementation of Extended Taxis Environment*, M.Sc. thesis, Department of Computer Science, University of Toronto, 1984.
- [PILO83a] Michel Pilote, *A Framework for the Design of Linguistic User Interfaces*. Ph.D. thesis, Dept. of Computer Science, University of Toronto, 1983.
- [PILO83b] Michel Pilote, "A Programming Language Framework for the Design of User Interfaces," *Proc. of the Conference on Principles of Programming Languages*, June 1983. (also [TAXI84])
- [ROSS77] D. T. Ross, "Structured Analysis(SA): A Language for Communicating Ideas," in *IEEE Transactions on Software Engineering*, Volume SE-3, Number 1, January 1977, pp. 16-34.
- [SCHM80] Joachim W. Schmidt and Manuel Mall, *Pascal/R Report*. Bericht Nr. 66, Fachbereich Informatik, Universitaet Hamburg, Jan. 1980.
- [SMIT77] J. M. Smith and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems*, Volume 2, Number 2, June 1977, pp. 105-133.
- [TAXI84] *Taxis '84: Selected Papers*, Brian Nixon (ed.), Technical Report CSRG-160, June 1984, Department of Computer Science, University of Toronto.
- [WONG83] H. K. T. Wong, *Design and Verification of Interactive Information Systems*, Ph. D. Dissertation, University of Toronto, 1983.
- [ZISM78] Michael D. Zisman, Use of Production Systems for Modeling Concurrent Processes. In D. A. Waterman and Frederick Hayes-Roth (Eds.), *Pattern-Directed Inference Systems* New York: Academic Press, 1978, pp. 53-68.

The Database Design and Evaluation Workbench (DDEW) Project at CCA

David Reiner, Michael Brodie, Gretchen Brown, Mark Friedell*,
David Kramlich, John Lehman, Arnon Rosenthal

Computer Corporation of America
Four Cambridge Center Cambridge, MA 02142 USA
617/492-8860

Abstract

The Database Design and Evaluation Workbench (DDEW) is a graphics workstation for database designers. DDEW provides an interactive support environment for specifying and experimenting with database structures and designs, while automatically maintaining a complete history of the design alternatives that are investigated. It allows easy and uniform access to a highly integrated and extensible suite of evaluation, analysis, and design transformation tools that range over the entire database design life-cycle. The system is object oriented, supports multiple windows, and has powerful diagram representation and editing capabilities.

1. Introduction to the Project

DDEW will reside on a Jupiter 12 workstation with a 68010 microprocessor (running Berkeley 4.2 UNIX**), 2 MB of main memory (with 160 MB more on a hard disk), a bit-sliced graphics processor, and a 1280 x 1024 color frame buffer. This system may be loosely coupled to a DEC VAX with an Ethernet link. The workstation accepts input from both a keyboard and a mouse. The mouse is used to point to objects and to rapidly select commands from fixed and pop-up menus.

On the workstation screen, the designer can build, display, and manipulate objects of two fundamental types: lists and diagrams. Restricting the number of fundamental object types to two makes designer-system interactions quite uniform. Free-form and formatted textual data (such as requirements, attribute definitions, constraints, and design annotations) are represented as list items, which may be added, deleted, and modified with DDEW's list editor. Database schemas and the design history are represented as diagrams, which may be edited with DDEW's diagram editor. Contrasting colors and graphic icons help clarify design structures.

2. The Database Design Process

2.1 Design Methodology

DDEW supports a stepwise methodology for database design that is based on earlier work by Teorey and Fry [TEOR82b]. Its steps and the tools that DDEW will provide for them are shown in Figure 1 (see [REIN84] for more details). Preliminary versions of some of these tools were developed at the University of Michigan [TEOR82a]. Improved versions are being built specifically for the DDEW project. The methodology is iterative; earlier decisions always can be reconsidered, and alternatives to them can be explored in parallel.

DDEW's integrated methodology is a framework for the efficient use of the system by all designers, and an educational aid for novice designers. New and improved tools (e.g., for physical design) can be incorporated as they become available.

This project is supported by the Rome Air Development Center (of the United States Air Force) under contract number F30602-83-C-0073.

* Center for Research in Computing Technology, Harvard University

** UNIX is a trademark of Bell Laboratories

Contributing to this ease of modification is the use of a common storage system for design data, a relatively general system-tool interface, and a clean division of the methodology into levels of abstraction.

2.2 Data Models

An extended version of the Entity-Relationship model [CHEN77] (referred to as ER+) underlies all design phases of DDEW. The principal components of the ER+ model are entities, binary relationships between entities, and attributes (of both entities and relationships). Multivalued attributes (repeating groups) are represented as (weak) entities. DDEW recognizes and exploits ER+ functional dependencies (including keys), inclusion dependencies (subset constraints), and constraints on cardinalities of relationships and on data types of attributes. Transaction specifications take the form of a sequence of operations (query, update, insert, delete) on data objects and on intermediate, set-valued results of other operations.

The user does conceptual design in the ER+ model, and logical design in (generic) relational, network, or hierarchical models. The system represents all DDEW designs internally as ER+ schemas, with additional information and restrictions for the logical (and subsequent) levels of design. Examples of data model restrictions on ER+ are: for the relational model, each entity must have a declared key, and no relationships are allowed; and for the network model, no m:n or cyclic relationships are permitted. The (generic) hierarchical model is a subset of the network model, where there is at most one incoming relationship for entities and no cycles.

Future extensions to the ER+ model may include: aggregation, generalization hierarchies, multivalued dependencies, "null not allowed" declarations for attribute values, derived fields (a feature often implemented in commercial systems), and an ER query language.

3. User Interface and Graphics Support

3.1 Earlier CCA systems

Many characteristics of the DDEW user interface originated in three earlier systems developed at CCA: SDMS (Spatial Data Management System), the View System, and PV (Program Visualization). SDMS [HERO80] is a graphical user interface to conventional databases. Graphical icons that represent entities in the database are arranged in 2-D Information Spaces, over which the user can scroll and zoom to examine different parts of the database at different levels of detail. The View System [FRIE82] is an enhanced SDMS that automatically creates new Information Spaces in response to a user's ad hoc queries. A repertoire of layout heuristics enables View to arrange icons within Information Spaces. PV [KRAM83] allows programmers to examine and manipulate dynamic graphical representations of programs.

3.2 DDEW Screen Layout

Figure 2 shows a typical DDEW screen configuration (mocked up) that a designer of a project control database might have created. The screen is divided into fixed areas. The rectangular workspace takes up roughly two-thirds of the screen. On it appear windows onto design diagrams and the design tree, lists, and pop-up menus. Windows can be scrolled left, right, up, and down, or resized. Below the workspace is a legend, from which the designer selects object icons or list templates to add to the active window. The legend contains only those icons or templates that legally can be added to the diagram or list in the active window.

To the right of the workspace is the fixed menu area containing global commands that are always accessible. These include basic window operations (move, cycle, copy, save, restore) and commands to control tools running in the background (abort, suspend, resume). Across the top of the screen is an area for system messages and prompts, a mailbox for messages from tools running in nonactive windows, and a navaid (navigational aid). The navaid shows a miniaturized view of the entire diagram in the active window, with a dotted rectangle to indicate the portion of the diagram that is visible in the window.

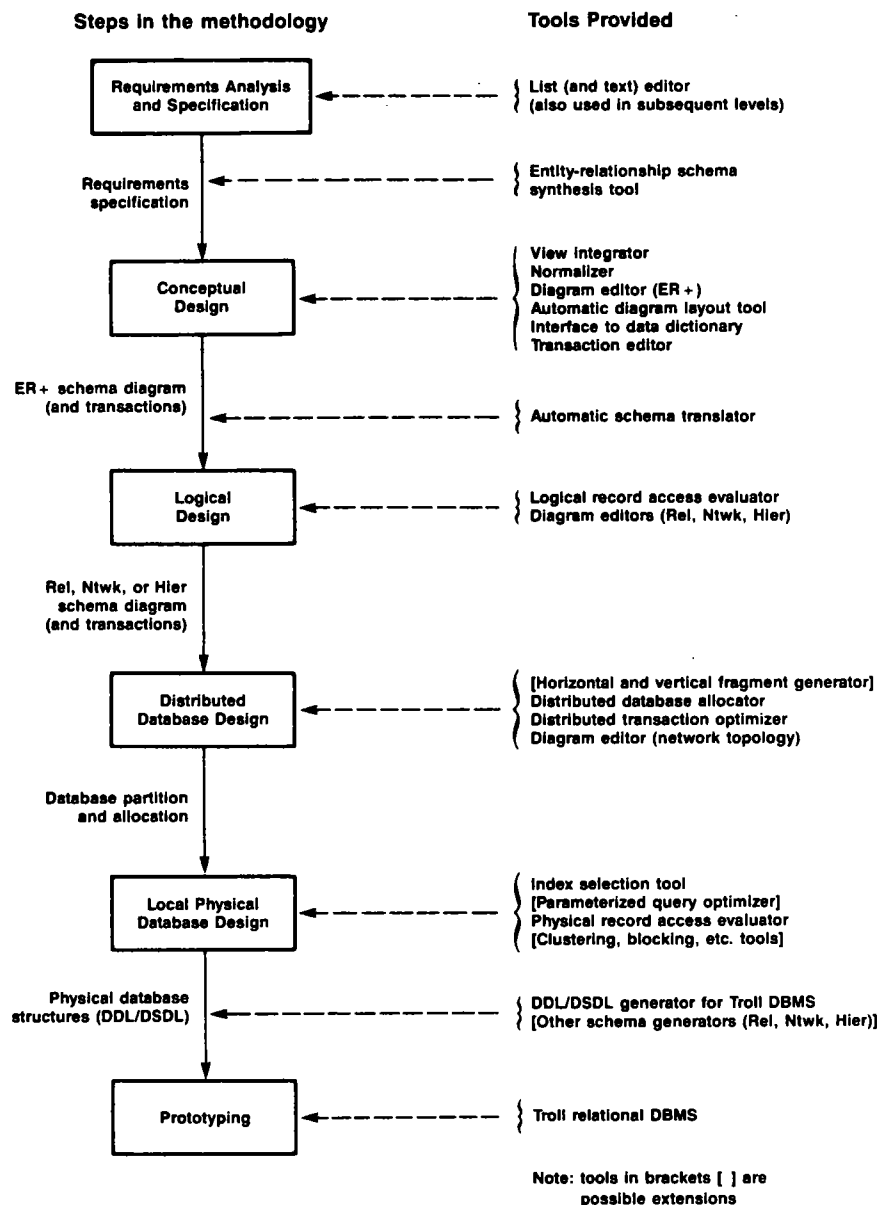


Figure 1. DDEW Methodology and Tools

3.3 Object-Centered Command Interface

To avoid making the designer learn arcane and convoluted syntax, DDEW has an object-oriented command interface and uses pop-up menus. When the designer selects an object from a diagram or list (using the mouse), a pop-up menu appears containing the operations that are applicable to the selected object. This menu disappears when an operation is selected from it (or from the fixed menu, which is always available). The designer may be prompted for additional arguments to the command.

3.4 On-Line Help

In DDEW, help is provided by two mechanisms: referential help and context-sensitive help. Referential help, invoked by the MORE HELP command on the fixed menu, provides descriptions of DDEW commands, graphical notation, and the design methodology. Context-sensitive help, invoked with a dedicated button on the mouse, provides fast, concise help messages that focus on interaction expectations. It gives the designer a view of where he or she is in the system and what the system expects next. Context is based on the designer's position in the design tree, the states of active database design tools, and the state of the user command handler.

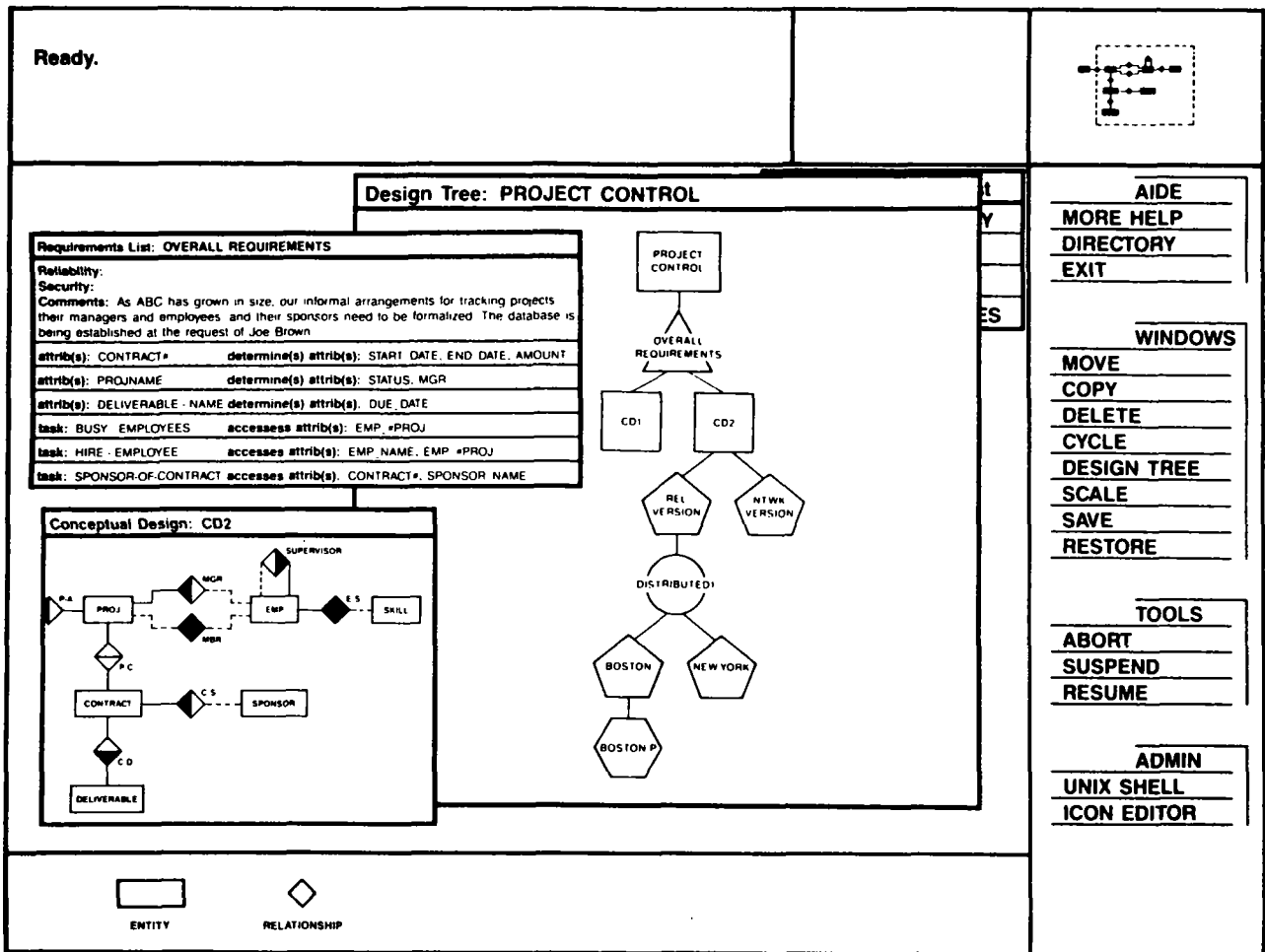


Figure 2. Conceptual Design (workstation screen mockup)

3.5 Windows

DDEW allows the user to create multiple overlapping windows (e.g., to view related lists and diagrams). Multiple windows promise efficient use of that precious resource, screen real-estate. The designer can open several windows onto different diagrams or onto different portions of the same diagram. DDEW maintains consistency among its windows by propagating changes to all relevant windows. Obscured portions of a window are saved and restored by the system to avoid having to recalculate their contents when the obscured portion is brought into view. The scrolling mechanisms and the navaid help the designer browse through and keep from getting lost in diagrams and lists that are too large to fit in a window.

3.6 Design Tree

As the designer moves through the database design life cycle, DDEW keeps a graphic record of the alternatives that are investigated and their interrelationships, in the form of a design tree. This ensures that no design work will be lost, and allows the designer to track the progress of designs. The tree contains the designer's comments about designs, retains the results of design analysis tools, and allows inspection of the ancestors of a design. Figure 2 shows a sample design tree.

3.7 Conceptual and Logical Design Diagrams

ER+, relational, network, and hierarchical schemas are represented graphically by nodes with arcs connecting them. For the ER+ model, we developed new graphic methods of specifying relationship cardinalities (by shading the halves of a diamond), and partial versus total participation in relationships (using dotted and solid lines, respectively). In Figure 2, P-C is 1:1, and both PROJ and CONTRACT par-

ticipate totally in P-C. EMP to PROJ is 1:n through MGR; an employee need not manage any projects, but each project must have a manager. EMP to SKILL is m:n; some skills possessed by no employee may be present, but every employee must have at least one skill.

Network schemas are similar, but half-diamonds (which look like arrowheads) are used to represent set types between record types. The network display includes ER+ information that cannot be captured in a network schema (e.g., whether or not a set owner instance must have at least one member instance).

For relational schemas, inclusion dependencies (represented by set inclusion symbols) are shown as graphic links between relations, similar to with set types in network schemas. Translation to the relational model always creates one or two inclusion dependencies (which ensure referential integrity) when a relationship is given a value-based representation. Representing dependencies graphically allows the designer to perceive the structure of the database much more easily than when a relational schema is represented as a collection of unrelated boxes.

Diagram editing commands allow the designer to manipulate single nodes and arcs in a diagram. Basic editing functions for nodes include: create a new instance, delete, move, and rename. Arcs can be moved and reconnected. The designer also can specify collections of nodes, called affinity groups, as named subsets of the entire diagram, and can move them as a block.

DDEW provides layout assistance ranging from incremental placement and connection of nodes and affinity groups to automatic layout of design diagrams, drawing partly on placement techniques developed for VLSI design. Fast layout heuristics (e.g., placement of nodes only at grid positions) are used to improve response time and diagram uniformity. The goal is to quickly produce a first-cut layout that the designer can modify if desired.

3.8 List Manipulation

Textual data in DDEW generally is displayed in lists. The main list types are: information (status of design nodes), free-form annotations (on objects), attributes, transactions, functional dependencies, inclusion dependencies, system and user-defined affinity groups, requirements, mail, tool input (e.g., candidate attributes for indexing), tool output (e.g., schedules of transactions in a distributed environment), and both generic and DBMS-specific DDL. The field-sensitive DDEW list (and text) editor will allow the designer to perform uniform operations on a broad range of objects.

3.9 Tool Interface

One of DDEW's main strengths is the high degree of integration of its evaluation, analysis, and design transformation tools. The interface to all the tools is uniform, and the designer will not have to memorize invocation conventions, write format translation programs between tools, learn a file system, cope with version control, or know UNIX. The system automatically retrieves and stages tool input data, and the designer need only examine and evaluate the results.

4. Systems Aspects

DDEW has an internal DBMS (Troll) that provides both centralized data storage and a high degree of flexibility and data independence, making it easy to incorporate new tools and data model extensions into the system. The design database will include information about entities, relationships, attributes, relationship cardinalities, functional and inclusion dependencies, transactions, display positions of objects (such as nodes and arcs), and relationships among design tree nodes. Much meta-information also will reside there (e.g., list item templates, legal design tree and design diagram configurations, and menu contents). Other systems aspects of DDEW are described at greater length in [REIN84].

5. Related Work

Giant wall charts maintained by pencil, scissors, and paste are useful for schema visualization but are unwieldy and a poor form of input to computerized analysis tools. Data dictionary systems enforce a certain amount of uniformity across schemas but are not as complete as design tools. Some early automated database design aids have become popular but are either limited in their scope of application to a single system [IBM75] or are overly text-oriented [TEIC77]. Research into similar (and fairly ambitious) database design systems is underway in Sweden [SYSL83] and Italy [BATA84].

6. Future Plans

The DDEW prototype will be completed by July 1985, after a total effort of eight person-years. The Rome Air Development Center, the project sponsor, expects to use DDEW in a variety of upcoming design and redesign efforts and to encourage standardization of design methodologies.

Possible extensions to the prototype DDEW include capabilities for enhanced semantic modeling in ER+, query optimization (parameterized for the target DBMS), evaluation of database security, extensive physical design, generation of other DBMS-specific DDL/DSDL, hard-copy output of design diagrams, off- or on-line database tuning, and semiautomatic schema generation from requirements. An intriguing direction is enhancing the user interface to make DDEW more of an expert system, capable of understanding the designer's context, goals, and available tools, and of giving useful advice about alternative choices and their implications.

7. References

- [BATA84] Batini, C., De Antonellis, V., Di Leva, A., "Database Design Activities within the DATAID Project," Database Engineering, 7, 4, December 1984.
- [CHEN77] Chen, P., "The Entity-Relationship Approach to Logical Data Base Design", Q.E.D. Monograph Series, Wellesley, Massachusetts, 1977.
- [FRIE82] Friedell, M., Barnett, J., and Kramlich, D., "Context-Sensitive, Graphic Presentation of Information," SIGGRAPH '82 Proceedings, July 1982.
- [HERO80] Herot, C., Carling, R., Friedell, M., and Kramlich, D., "A Prototype Spatial Data Management System," SIGGRAPH '80 Proceedings, July 1980.
- [IBM75] "IBM Data Base Design Aid - A Designer's Guide", Program No. 5748-XX4, GH20-1627-0, 1975.
- [KRAM83] Kramlich, D., Brown, G., Carling, R., and Herot, C., "Program Visualization: Graphics Support for Software Development," ACM IEEE 20th Design Automation Conference Proceedings, June 1983.
- [REIN84] Reiner, D., et al., "A Database Design and Evaluation Workbench: Preliminary Report," Proc. SPOT-3 Conf., SYSLAB, Chalmers University of Technology, Goteborg, Sweden, August 1984.
- [SYSL83] "Progress Report, July 1982 - June 1983", The Systems Development Laboratory, University of Stockholm and Chalmers University of Technology, Sweden.
- [TEIC77] Teichroew, D., and Hershey, E.A., "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Trans. on Software Eng., SE-3, 1, 1977.
- [TEOR82a] Teorey, T.J., and Cobb, R., "Functional Specifications for a Database Design and Evaluation Workbench", Working Paper 82 DE 1.15, Information Systems Research Group, Graduate School of Business Administration, University of Michigan.
- [TEOR82b] Teorey, T.J., and Fry, J.P., Design of Database Structures, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

DATABASE DESIGN ACTIVITIES WITHIN THE DATAID PROJECT

C. BATINI (+), U. DE ANTONELLIS (++) , A. DI LEVA (+++)

(+) Dipartimento di Informatica e Sistemistica
Universita' di Roma "La Sapienza"

(++) Istituto di Cibernetica - Universita' di Milano

(+++) Istituto di Scienza dell' Informazione - Universita' di Torino

ABSTRACT

This paper summarizes current research and development activities carried on by the Italian DATAID five-years research project in the area of methodologies and tools for database design. The main contributions of the project were a manual methodology, DATAID-1, that covers all the phases of database design, and a set of tools that support the methodology in most critical activities.

1. INTRODUCTION

In September 1979 the Italian National Research Council (C.N.R.) started, within the PROGETTO FINALIZZATO INFORMATICA research project, a five-year project, called DATAID, to develop a computer aided methodology for data base design. DATAID has considered the development of a computer aided methodology for database design with the following features: it covers all the design phases from requirements collection and analysis to the choice of the internal data structures; it is aided by tools which support the execution of the methodological steps, automatically verify design consistency, and handle design documentation; it is oriented towards both centralized and distributed applications.

A manual methodology, DATAID-1, was first been developed and is now available together with teaching modules. Several tools supporting the methodology have been designed and implemented as prototypes. The research program until the end of the project includes the extension of the methodology to incorporate distribution design and the full development and testing of tools.

A detailed description of the first project results is in [CER183a], while the most recent results are described in the forthcoming book [ALBA85b]. In this paper we provide a brief description of both the methodology and tools developed within the DATAID project.

2. THE DATAID-1 METHODOLOGY

In order to experiment with the desired general features mentioned above, we decided to produce, first of all, a simplified prototype of the design methodology, called DATAID-1, whose major characteristics are:

1. the methodology is manual, i.e. it does not require a computer support;
2. the conceptual model adopted is the Entity Relationship Model, enriched with generalization hierarchies;
3. in the conceptual step the methodology provides several heuristics to guide the designer in choosing conceptual structures and checking the schema quality.
4. in the logical and physical design phases, both relational and CODASYL

classes of DBMS's are considered;

5. the logical and physical design phases are driven by quantitative parameters in order to optimize the execution of the most important operations that will be performed on the database.

The DATAID-1 methodology decomposes the design of a database application into four design phases: user requirements collection and analysis; conceptual design (views design and integration); logical design; physical design. These phases are described below.

2.1 Requirements collection and analysis

The aim of this phase is to collect, for each enterprise environment, user requirements and formalize them into descriptions of data, operations, and events. Inputs to the phase are natural language sentences, collected following a collection plan which identifies, for each environment, users from whom to get the requirements of the database application; users are asked to describe how the functions are executed, specifying the data and the operations of interest for each function. Output of the phase is a collection of glossaries describing data, operations and events. In order to fill in the glossaries, natural language sentences are filtered and rewritten in a restricted language; revised requirements are then classified into different sentence types (data, operations and events sentences).

2.2 Conceptual Design

In the conceptual design phase a conceptual view is built for each organization environment (view) of the enterprise; these views are then integrated to form a global conceptual description. The conceptual schema of an environment is the formalized representation of both the static (data) and the dynamic requirements (operations and events).

Starting from the glossaries built in the previous phase, a set of operation schemas is first modeled. An operation schema is a high level, nonprocedural description of the data involved in the operation, expressed using an Entity Relationship description. The data schema, which is the complete representation of all the data used by the operations of a particular environment, is built incrementally: when an operation schema is completed, it is aggregated with the previous partial data schema. This process is iterated for all the operations and eventually produces the data schema. A similar process produces the events schema, a description of all the functions of the environment as a network of conditions and operations. Data and operations schemas are described with an extended version of the Entity Relationship model; for the events schemas, the Petri net formalism is used.

Data schemas are then analyzed and integrated to build the global data schema. Because of different perspectives that users may have and different equivalent representations that exist in the model, several complex activities are required during views integration: finding the common part of the different schemas, finding the different representations chosen by the designers, and so on. After the global data schema has been obtained, operations schemas are restructured in order to make them consistent with the final representation of data in the global schema, and enriched with procedural and quantitative information. Finally, events schemas are coordinated in order to represent communication among the

environments of the enterprise. This results in the global events schema, the formal specification of the database application behaviour. In the present version of the methodology, the events schema is considered as an input to software design and is not used in further design phases.

Recently the DATAID-1 conceptual step has been improved in several directions:

1. Attention has been given to the influence of linguistic representations used to express requirements on design activities. The idea is that each linguistic representation used for requirements (e.g. natural language, forms, record formats) has its own peculiarities in expressing the semantics of data; such peculiarities should be captured by the designer to simplify the design process (see [BATI84a]).
2. A deeper comprehension has been obtained of conceptual design goals (completeness, minimality, readability, self-documentation) and activities needed to achieve them (see [BATI84b]).

Presently, research activities concern the extension of the methodology to specific classes of applications (e.g. statistical databases) and the introduction of a maintenance phase, whose goal is to reuse as far as possible documentation on previous designs when some restructuring is needed.

2.3 Logical Design

The process of logical design transforms the global conceptual schema into a logical schema, depending on the database management system chosen for the implementation. Two classes of database management systems are considered: relational DBMSs and network (Codasyl-like) DBMSs. The transformation process is based on two fundamental tasks:

1. Simplification of the global conceptual schema: data structures not directly translatable into the logical model (such as generalization hierarchies and multiple relationships) are converted into simpler ones.
2. Refinement of the simplified schema: a set of transformations on the simplified schema is applied (typically, partitioning of entities and replication of attributes); performance measures (based on the number of logical accesses) are used in order to select, among different alternatives, the solution which optimizes the execution of the most important operations.

2.4 Physical Design

Physical design decisions for Codasyl-like databases are subdivided into three broad decision areas:

1. access path support decisions: implementation strategies for entry point records (LOCATION MODE clause options) and sets (SET MODE clause options) are considered;
2. placement strategy decisions: member records of a set are dispersed throughout the database area or clustered so that neighbouring member records tend to be stored in the vicinity;
3. storage allocation decisions: database areas for storage of records, indexes and pointers array tables are selected; each area is subdivided into a number of pages and the page length is fixed.

The methodology is based on the evaluation of all possible record and set implementation strategies from the global processing point of view. This

is accomplished in the following tasks:

1. creation of record usage trees: accesses to a record are globally described as a tree where the leaf nodes are the different types of operations performed on the object record;
2. storage allocation: heuristic rules are used for calculating record length, record allocation in areas, area and page size, and other physical parameters.
3. evaluation of implementation strategies: starting from the relative costs and frequencies of the operations, implementation strategies are evaluated.

The main goal of physical design for relational DBMS's is the selection of secondary indexes of the relations of the schema. Physical design in this case proceeds evaluating first of all costs of operations and generating then an efficient set of indexes.

Logical and physical design activities are now being extended to distributed environments (see [CER183b], [NAVA], [CER184a], [CER184b]). The main contribution is the distinction between two steps, a logical step that concerns design of the structure of horizontal and vertical fragments, and a physical step that concerns the optimal allocation of fragments to nodes.

3. THE COMPUTER AIDED METHODOLOGY

After experimenting with the DATAID-1 manual methodology, several automated tools were developed; the tools were intended to provide support to the data base designer by means of an environment in which a design tool collects the static and dynamic definitions, tests their mutual consistency, and sometimes detects or solves design problems. A detailed description of tools may be found in [CER183a], [ALBA85a]: we focus here on INCOD-DTE, DIALOGO, ISIDE.

INCOD-DTE (Interactive Conceptual Design: Data, Transactions, Events) was developed at the Dipartimento di Informatica e Sistemistica - Università di Roma, and at the Istituto di Cibernetica - Università di Milano, with the collaboration of Database Informatica - Roma. The tool provides an integrated environment for the definition of data, transactions, events. Data are described using the extended Entity-Relationship Model, as in DATAID-1. Transactions are described at different levels of abstraction, as the tool provides transaction definition commands at a conceptual level (describing the data involved), at a navigational level (describing access paths) and at an executable level (testing a prototype implementation of the database). Events are modeled through Petri Nets, and are specified using event specification commands (concerning events, see also [DEAN83a], in which the original formalism is extended to express executable procedures). INCOD-DTE can support the designer in the conceptualization of static and dynamic requirements, automatically checking the consistency of the process and simplifying the management of the corresponding documentation. It handles several types of metadata, interacts with the user in discovering conflicts, prompting possible solutions (scenarios) and guiding him through some steps of the design activities. INCOD-DTE is currently being implemented in a UNIX environment.

INCOD-DTE is now being extended with GINCOD, a graphical interface that can also be seen as a self-contained graphical editor for interactive design of

conceptual schemas expressed in terms of a diagram. Placement of symbols and layout of diagrams can be driven in GINCOD both by the designer (with classical graphical editor commands) and by the system. When layout is performed by the system, the following "aesthetics" are taken into account (see also [BAT184c]):

- minimization of the number of crossings between connections
- minimization of the number of bends in connection lines
- minimization of the global length of connections lines.

A running prototype of GINCOD has been implemented in PASCAL.

DIALOGO was developed at the Dipartimento di Informatica - Universita' di Pisa with the collaboration of Systems and Management - Torino. DIALOGO is an interactive system that allows use of the programming language GALILEO in the design of a conceptual schema. The main characteristics of DIALOGO are that the user interacts with the system using a single language to: edit the schema and operations definitions; ask the system information about definitions; load sample data to test the behaviour of the operations; define new operations to personalize the working environment. DIALOGO is now being extended to manage also the phase of requirements collection and analysis in a unified environment. For further information on DIALOGO, see [ALBA84a] in this issue.

INCOD-DTE and DIALOGO can be seen as two complementary attempts to design effective tools for the designer of a conceptual schema.

In INCOD major emphasis is given to identifying which specific design activities can be given to an automatic system; e.g. in order to increase the readability of the schema, several possible restructurings can be performed. While it is a task of the designer to find the fragments of the schema to be restructured, INCOD can perform the transformation. In DIALOGO the major effort is to provide a highly sophisticated linguistic feature that allows expression at the conceptual level of a rich set of semantic properties of data and executable specifications for processes. Although the systems are based on different data models, that is an extended Entity Relationship data model and a semantic data model, the tools provided by INCOD-DTE can also be embedded in DIALOGO and in fact the conceptual design produced by INCOD-DTE can be considered as a first step toward an executable conceptual design given in GALILEO.

Automated support for the logical and physical design phases is under development as a Joint Project - the ISIDE (Integrated System for Implementation Design) project - by the Dipartimento di Informatica - Universita' di Torino, CIOC - Bologna, CSELT - Torino and CRAI - Cosenza. ISIDE (see [BONA], [DILE], [ORLA]) is a collection of integrated tools that support the logical and physical design phases and give an evaluation of data manipulation operations considered in the conceptual phase. ISIDE is composed of six modules:

1. the Dynamics analyzer generates quantitative parameters for the logical and physical design process.
2. the Logical designer translates the conceptual description into a Codasyl or relational logical schema. The translation process is driven by the quantitative parameters that characterize the workload defined on the database.
3. Relational and Codasyl Physical designers refine the logical schema with the appropriate physical data structures definition statements to become a complete database schema processable by the target DBMS.

4. Relational and Codasyl performance predictors help the designer in testing the adequacy of design decisions. Given a processing scenario (number and type of operations simultaneously active on the database at a given time), the predictors evaluate the expected response time for each operation. This information can be used to check whether the design is likely to satisfy the user efficiency requirements. The ISIDE system is being developed using PASCAL under UNIX. For modules 2,3,4,5 and 6 a working prototype is operative; module 1 is under development.

Concerning extensions to distributed design, various algorithms for vertical and horizontal partitioning have been implemented and are currently integrated.

REFERENCES

- [ALBA84a] A. Albano, R. Orsini - A PROTOTYPING APPROACH TO DATABASE APPLICATIONS DEVELOPMENT - IEEE Database Engineering, this issue.
- [ALBA85a] A. Albano, L. Cardelli, R. Orsini - GALILEO: A STRONGLY TYPED, INTERACTIVE CONCEPTUAL LANGUAGE - To appear in ACM Transactions on Database Systems, 1985.
- [ALBA85b] A. Albano, V. De Antonellis, A. Di Leva (eds.), COMPUTER AIDED DATABASE DESIGN, North Holland 1985.
- [BATI84a] C. Batini, B. Demo, A. Di Leva - A METHODOLOGY FOR CONCEPTUAL DESIGN OF OFFICE DATA BASES - Information Systems, Vol. 9 N. 4 (1984)
- [BATI84b] C. Batini, M. Lenzerini - A METHODOLOGY FOR DATA SCHEMA INTEGRATION IN THE ENTITY RELATIONSHIP MODEL - IEEE Transactions on Software Engineering, 1984.
- [BATI84c] C. Batini, M. Talamo, R. Tamassia - COMPUTER AIDED LAYOUT OF ENTITY RELATIONSHIP DIAGRAMS - Journal of Software and Systems, 1984.
- [BONA85] R. Bonanno, D. Maio, P. Tiberio - AN APPROXIMATION ALGORITHM FOR SECONDARY INDEX SELECTION - to be published in Computer Journal.
- [CERI83a] S. Ceri (ed.) - METHODOLOGY AND TOOLS FOR DATABASE DESIGN, North Holland 1983.
- [CERI83b] S. Ceri, S.B. Navathe, G. Wiederhold - DISTRIBUTION DESIGN OF LOGICAL DATABASE SCHEMAS - IEEE Transactions on Software Engineering, Vol SE-9, N. 3, July 1983.
- [CERI84a] S. Ceri, G. Pelasatti - DISTRIBUTED DATABASE SYSTEMS - McGraw Hill, 1984.
- [CERI84b] S. Ceri - AN OVERVIEW OF RESEARCH IN THE DESIGN OF DISTRIBUTED DATABASES - IEEE Database Engineering, this issue.
- [DEAN83a] V. De Antonellis, R. Bertocchi, B. Zonta - CONCEPTS AND MECHANISMS FOR HANDLING DYNAMICS IN DATA BASE APPLICATIONS, Proc. 7th ICS ACM European Regional Conference, Nurnberg 1983.
- [DEAN85] V. De Antonellis, A. Di Leva - DATAID-1: A DATABASE DESIGN METHODOLOGY, Information Systems 1985.
- [DILE] A. Di Leva, P. Giolito - AUTOMATIC LOGICAL DATA BASE DESIGN IN A CODASYL ENVIRONMENT - Proc. IEEE 4th Jerusalem Conference on Information Technology, Jerusalem 1984.
- [NAVA] S. B. Navathe, S. Ceri, G. Wiederhold, J. Dou - VERTICAL PARTITIONING ALGORITHMS FOR DATABASE DESIGN - ACM Transactions on Database Systems, Vol. 9, N. 3, September 1984.
- [ORLA] S. Orlando, P. Rullo, W. Staninzky - TRANSACTION WORKLOAD EVALUATION IN THE CODASYL DATABASE ENVIRONMENT - Proc. IEEE Computer Data Base Engineering Conference - Los Angeles, 1984.

A REALISTIC LOOK AT DATA

W. Kent

IBM General Products Division
Santa Teresa Laboratory
San Jose, California, 95150

Re-examining the basic properties of records leads to a simpler, fact-based approach to data analysis, design, and documentation.

It's useful to keep in mind that the subject matter of data analysis, design, and documentation is data records. How we perform those functions is guided by our view of the nature of records themselves. As with constellations and elephants, we can choose to see the "reality" of these things in different ways.

Our goal initially is to determine what concepts are really essential for the tasks of logical design and documentation of normalized relational databases. The concepts will apply quite readily to record design in other kinds of data bases, and provide a sound basis for considering non-normalized designs as well [KENT83a,b,c].

The central questions in logical data design are: (1) what records should be specified? and (2) what fields should those records have?

Current methodologies define records on the basis of an entity concept, perhaps motivated by the perception of records as entities in themselves. A data record has the characteristics of an entity. It is inserted, deleted, and retrieved as a single unit. Record occurrences are uniquely identified (by their keys), and they are aggregated into record types, corresponding to the notion of "entity type". Hence records are entities.

There is an understandable appeal in the apparent analogy between records and real-world entities. The fundamental modelling assumption underlying most current methodologies is that some things in the real world are entities and others are not, and records are chosen to correspond to the entities. That is, data records constitute a "model" of real entities. In simple situations we have, e.g., one record to represent an employee, another to represent a department, and so on.

In this approach, which we can call "entity-based", two kinds of information are distinguished: attributes of entities, and relationships among entities. Records are populated with fields in two steps:

1. Provide fields for single-valued attributes of the entity.
2. Provide fields for many-to-one relationships with other entities.

Afterwards, additional records must be defined to accommodate many-to-many relationships, n-ary relationships, and multi-valued attributes.

This entity-based approach provides a reasonable first approximation, but leaves considerable room for refinement. In the first place, there really are no effective criteria for deciding which things in the real world are entities and which are not. Ultimately we are left only with a circular definition: we ought to define as entities those things for which we want corresponding records. The distinction between attributes and relationships is similarly blurred. At best, if we have figured out which things are entities, then we might say that relationships are facts which connect entities, while attributes connect entities with non-entities.

Furthermore, the correspondence between records and real-world entities is quite imperfect. In most real databases, one can find:

1. Information about a given entity distributed over several records.
2. Information about several entities contained in the same record.
3. Records that don't represent any one entity, or records that "represent" several entities.
4. Things which we might intuitively consider to be entities but having no corresponding records to represent them.

As much as we might wish it, we cannot find a general one-to-one correspondence between records and real-world entities -- unless we rely on the circular argument which defines an entity to be something represented by a record.

At best, we will have a one-to-one correspondence between some records and some entities. In such cases, a record type is serving as a "master list" for the population of the corresponding entity type. That is, entities of this particular type are deemed to exist if and only if they have a corresponding occurrence of the record type. This underlies the concept of "referential integrity" [DATE81], wherein entities of certain types may be referred to only if they are identified by a "primary key", i.e., in a master-list record.

We could arbitrarily define "entity" to mean those things for which we do provide such master-list records, but that dis-

inction is not particularly useful, and should not be exaggerated.

Now suppose we discard those modelling assumptions, and take a fresh look at the realities of data records.

If we examine any field in a record, we find that it contains a character string. These character strings generally represent something in the real world, e.g., a person, a department, a color, a length, a money amount, and so on. Furthermore, any of these things in the real world can often be represented by different strings in different fields, i.e., according to different representation conventions. A department may have a name and a number; a color may have different names in different languages; lengths can be expressed in different units, and money in different currencies. Some representation schemes provide unique identification: "D99" represents one department, and "6 feet" represents one distance.

How is information conveyed by fields? Any field considered in isolation contains very little useful information. A field by itself can only tell you that certain things exist, e.g., employees, dates, money amounts, etc. Most of the real information contained in databases consists of the interconnections among such things. A money field is only meaningful because it is connected in a record with the employee who earns that money as a salary. A date field is only meaningful when connected to a field representing something related to that date, e.g., the employee who was born, or hired, or married, on that date.

Information is provided by the aggregation of fields into records, i.e., by the connections among fields. This is the essential information-bearing property of records: they tie fields together in order to capture the connections among them. That's why we have records: to connect related fields.

There are thus three main components in the descriptions of fields and records:

1. What each field mentions, i.e., the entity type.
2. How each field refers to that entity, i.e., the form of representation used.
3. Why each field is present in the record, i.e., what information is being conveyed by the fields. This corresponds to a connection among the things occurring in several fields, i.e., a relationship, or fact.

Everything we have said about fields applies uniformly to all fields. We don't have any basis here for saying that entities occur in fields in one way and non-entities occur in a different way, or that fields for attributes are somehow different from fields for relationships. Facts are "about" any of their participants. The assignment of employees to departments is as

much a fact about departments as about employees. Birthdates are facts about people, and also about dates (who was born on that date).

Records are, we observe, aggregations of facts. But, given a collection of facts to be maintained in a database, how should they be clustered into records? That question essentially restates the fundamental task of logical database design.

Here the fact-based approach departs significantly from the entity-based approach. Rather than saying that we choose entities as clustering points, and aggregate facts around entities, we observe that clustering is actually dictated by the mechanism for maintaining single-valued facts (many-to-one relationships). This mechanism is provided by single fields in keyed records.

Key fields, containing unique and singular identifiers, insure that there will not be more than one record corresponding to the entity identified in the key. This allows a record type to serve as a "master list" for the population of an entity type.

Single-valued facts are then maintained by representing them in (non-repeating) fields in keyed records. For example, if there is only one record per employee, and each such record contains only one field for a department, then we have no way to associate an employee with more than one department. A single field in a keyed record enforces the many-to-one relationship, i.e., the single-valuedness of the fact.

Many-to-many relationships, such as the list of departments in which an employee has worked in the past, or the list of his skills, must go in a separate record, sometimes referred to as an "intersection record".

The structure of records in a logical database design is dictated by the following principles:

1. The single-valuedness of a fact about something is enforced by maintaining that fact in a record having that thing's identifier as its key.
2. Several single-valued facts about the same thing can be maintained in the same record.
3. For all other kinds of facts (e.g., multi-valued, n-ary), a separate record is required for each fact.

The results of the entity-based design methodologies eventually conform to this principle. It is simpler to use this record pattern directly as the basis for a design methodology.

We note that this record pattern is the same for "relationships" and "attributes". If an employee is assigned to a single department and earns a single salary, then each

gets a single field in the employee record. If an employee has been assigned to several departments and has earned several salaries, then each of these must be maintained in a distinct intersection record. We only need to distinguish which facts are single-valued and which are multi-valued. There is no motivation to distinguish between "relationships" and "attributes" based on record structure.

The record pattern principles subsume the master list concept, in practice. In all real applications, it seems that there are always some single-valued facts to be maintained about any entity type that may need a master list. Thus the fact-based approach will define the necessary records, without having to rely on a master-list principle for defining records.

The "first principles" developed in this re-examination of record semantics lead naturally to certain conclusions about entity-relationship models, data analysis and design, and data documentation.

A realistic look at the semantics of records and at the task of logical design reveal no basis for distinguishing between things which are entities and things which are not, or between facts which are relationships and facts which are attributes. This justifies the simplified entity-relationship model based on just the two fundamental constructs: entities and relationships [KENT78, NIJS83].

This does not imply a totally homogeneous view of all entities; some significant distinctions remain. Some entity types may be considered more important than others, and will be dealt with at earlier stages of data analysis and design than others. Some will be considered important enough to be included in conceptual schema diagrams, and others will not. Some entity types have corresponding record types to serve as master-lists, others don't.

The significance of such distinctions should not be exaggerated. They do not imply essential distinctions in the semantics of data structure, or in the methodologies of data analysis and design.

The fact-based approach leads to a simple synthetic methodology for data analysis and logical design [KENT83c]. Data analysis consists essentially of identifying the facts to be maintained, without trying to distinguish between entities and non-entities, or between relationships and attributes. It is only necessary to distinguish between single-valued and multi-valued facts.

Logical data design then consists simply of aggregating together single-valued facts about the same things. Multi-valued facts and n-ary relationships will, in most cases, remain unaggregated, i.e., they will be left in separate

records. The end results are the same as for the entity-based methodologies, but achieved more simply and directly.

The fact-based approach also leads to a simple structure for documenting the meanings of data elements in a dictionary [KENT85]. The first requirement is to have a distinct dictionary entry for each field. Factoring several fields into the same "data element", as is common practice, precludes the possibility of fully documenting the unique meaning of each field. The documentation for each field should capture:

1. The entity type represented in the field.
2. The form of representation used in the field.
3. The relationship(s) in which the field is involved, including the role it plays.

This documentation can be developed during the design process.

References

[DATE81] C.J. Date, An Introduction to Database Systems (third edition), Addison-Wesley, 1981.

[KENT78] W. Kent, Data and Reality, North Holland, 1978.

[KENT83a] W. Kent, "A Simple Guide to Five Normal Forms in Relational Database Theory", *Communications of the ACM* 26(2), Feb. 1983, 120-125.

[KENT83b] W. Kent, "A Catalog of Logical Data Design Options", IBM Technical Report TR03.224, March 1983.

[KENT83c] W. Kent, "Fact-Based Data Analysis and Design", in Entity-Relationship Approach to Software Engineering, North Holland, 1983 (Davis, Jajodia, Ng, Yeh, eds.). Also in *Proc. Symposium on Data Base Management Systems*, Nov. 15-17, 1983, Sydney, Australia. Also in *Journal of Systems and Software* 4(2,3), July 1984.

[KENT85] W. Kent, "The Semantics of Records", (in preparation).

[NIJS83] G.M. Nijssen, "From Data Bases Towards Knowledge Bases", in DBMS: A Technical Comparison, (Infotech State of the Art), P.G.H. King, editor, Pergamon Press, U.K., 1983.

TOOLS FOR VIEW INTEGRATION

Ramez Elmasri (1) James A. Larson (2), Sham Navathe (3)
and T. Sashidar (3)

(1) University of Houston
(2) Honeywell Computer Sciences Center
(3) University of Florida

ABSTRACT

Tools to aid the DBA in view integration are needed in two different contexts: a) merging views to form a logical schema of a single database, and b) merging views of separate databases to form a global schema representing the content of all the databases. This paper describes tools that are being designed for translating views to a canonical format, collecting correspondences among schema objects, merging views, and generating mappings between views and the merged schema.

I. INTRODUCTION

With the diversity of data models and database management systems, the development of view integration tools is becoming increasingly important. View integration arises in two different contexts:

- a) Logical database design. Several views are merged to form a logical schema describing the entire database. User queries and transactions specified against each view are mapped to the logical schema.
- b) Global schema design. Several databases already exist and are in use. The objective is to design a single global schema which represents the contents of all these databases. This global schema can then be used as an interface to the diverse databases. User queries and transactions specified against the global schema are mapped to the views (schemas) supported by the relevant databases.

Figure 1 illustrates four major tasks to be performed during integration:

1. Convert views to canonical data model. Intra-schema transformations are applied to each view to be merged so that it is converted into an easy-to-integrate form.
2. Specify correspondences among objects in two or more schemas. Correspondences are specified among the attributes, object classes, and relationships of the views to be merged.
3. Merge view. Objects in two or more views are merged, based on the specified assertions.

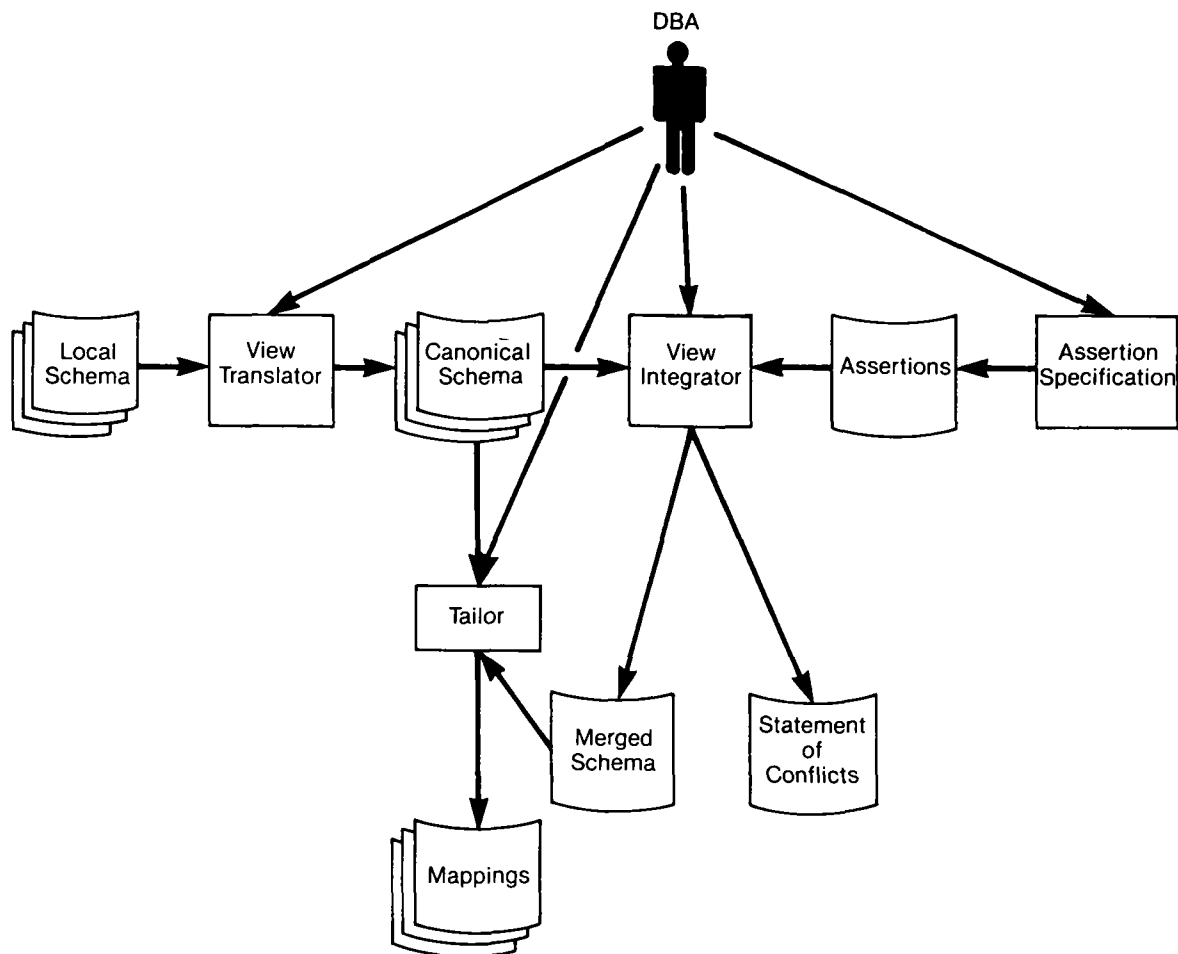


Figure 1

4. Generate mappings. Mappings between schemas and views are generated for use in translating database requests within either of two contexts:

- a) from views to the logical schema
- b) from the global schema to the underlying databases.

These four tasks are based on an extension of the methodology for view integration in [NAVA82]. Because these four tasks are closely related, software tools which aid the DBA in performing these tasks must be likewise interrelated. The remainder of this paper describes proposed software tools for performing these tasks.

II. CONVERT VIEWS TO CANONICAL DATA MODEL

View translators convert data descriptions expressed using different data model representations into a canonical format. For the canonical format, we have chosen an extension to the Entity-Relationship (E-R) data model [CHEN76], called the Entity-Category-Relationship (E-C-R) data model [WEEL80, ELMA]. The E-C-R data model extends the E-R model in two main areas:

1. The category concept is used to represent sub-classes [HAMM81]. Categories can also be used to group entities playing the same role in a relationship.
2. Cardinality and dependency constraints on relationships are specified precisely.

The E-C-R model uses the following constructs: entity sets, relationship sets, categories, and attributes. The term object set refers to entity sets or categories. Entity sets represent sets of entities that have the same attributes. Categories represent additional groupings of entities from one or more entity sets. Similarly, a category can be used to model a subset of one entity set. An example of the E-C-R model is shown in Figure 2; the E-C-R diagram is an extension of the E-R diagram [CHEN76]. Rectangular boxes represent entity sets, hexagonal boxes represent categories and diamond shaped boxes represent relationship sets.

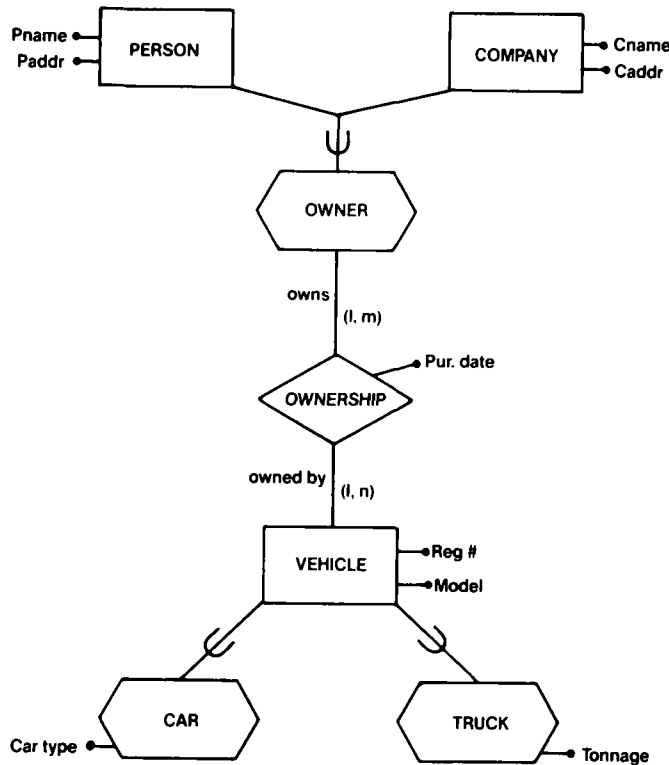


Figure 2.

Two types of categories exist in the E-C-R model. A sub-class category is a grouping of entities from one entity set or category. Members of any entity set may belong to any number of sub-class categories. The category concept allows one to model generalization. In the example shown in Figure 2, CAR and TRUCK are defined to be sub-classes of the VEHICLE set.

The second type of category is used to group entities playing the same role in a relationship. The category OWNER includes the entities COMPANIES and PERSONS playing the owner role in the OWNER relationship. Thus OWNERS are a subset of the union of COMPANIES and PERSONS.

Constraints on the number of instances of an entity or category that may participate in an instance of a relationship are represented by a pair of integers. In Figure 2, each vehicle instance must participate in at least one and at most n instances of the OWNERSHIP relationship.

III. SPECIFY CORRESPONDENCES AMONG OBJECTS

The following correspondences among the individual views are established by the DBA:

1. Object set name correspondences.
2. Attribute name correspondences.
3. Candidate keys for each entity set.
4. Correspondences between object sets in several schemas.
5. Correspondences among role names and relationship sets.
6. Correspondences among relationships sets relating the same object sets.

The correspondence of two object sets from different schemas depends on the possible extensions of these object sets when the database is populated. Two similar object sets A and B from different views are related in one of the following ways:

- (1) For each point in time, the extension of A are the same as the extension of B ($DOM(A)=DOM(B)$)
- (2) For each point in time, the extension of A includes the extension of B ($DOM(A)\supset DOM(B)$)
- (3) For each point in time, the intersection of the extension of A and the extension of B is empty ($DOM(A)\cap DOM(B)=\emptyset$)
- (4) None of the above.

The DBA systematically specifies which of these correspondences apply for pairs of object sets. Checking routines automatically notify the DBA whenever contradictory correspondence relationships have been specified. For example, if $DOM(A) = DOM(B)$, $DOM(D)\cap DOM(A) = \emptyset$, then $DOM(B) = DOM(D)$ would result in a contradiction.

Correspondences are automatically generated for some pairs of objects. For example if $DOM(A) = DOM(B)$ and $DOM(B) = DOM(D)$ have been specified then $DOM(B) = DOM(D)$ is automatically generated.

IV. MERGE VIEWS

We are designing an interactive view integration algorithm that will merge several views given a collection of specified relationships among object sets within the views. Object set integration rules are given in [ELMA84] and relationship set integration rules are presented in [NAVA84].

V. GENERATE MAPPINGS

The TAILOR algorithm, which is invoked by the view integration algorithms, generates mappings between the merged schema and the views to be merged. These mappings are used to translate commands expressed against the merged schema into commands expressed against views, or to translate commands expressed against one of the views into commands expressed against the merged schema, depending upon the context of the view integration process. The TAILOR algorithm will use query modification [STON75] to translate retrieval operations.

The design of a translation mechanism for update operations (sometimes called the view update problem) is difficult primarily because there may be many plausible translations for an update operation, each having a different effect on the database contents. The DBA uses TAILOR to develop several translation procedures for each object set and relationship set of the global schema or view seen by the user. The DBA may (1) restrict the set to one element, leaving the user no choice for choosing a translation procedure at run time; (2) make no restrictions; or (3) restrict the set somewhat but leave some choice for choosing a translation procedure for run time selection by the database user. Figure 3 illustrates the approach.

We have designed translation procedures to modify, delete from, and insert into an entity set derived from two entity sets participating in a relationship in the underlying schemas [LARS83]. However, additional general translation procedures need to be designed for entity sets and relationships sets derived in other ways. The update translation mechanism suggested by Masunaga [MASU83] appears promising for this purpose.

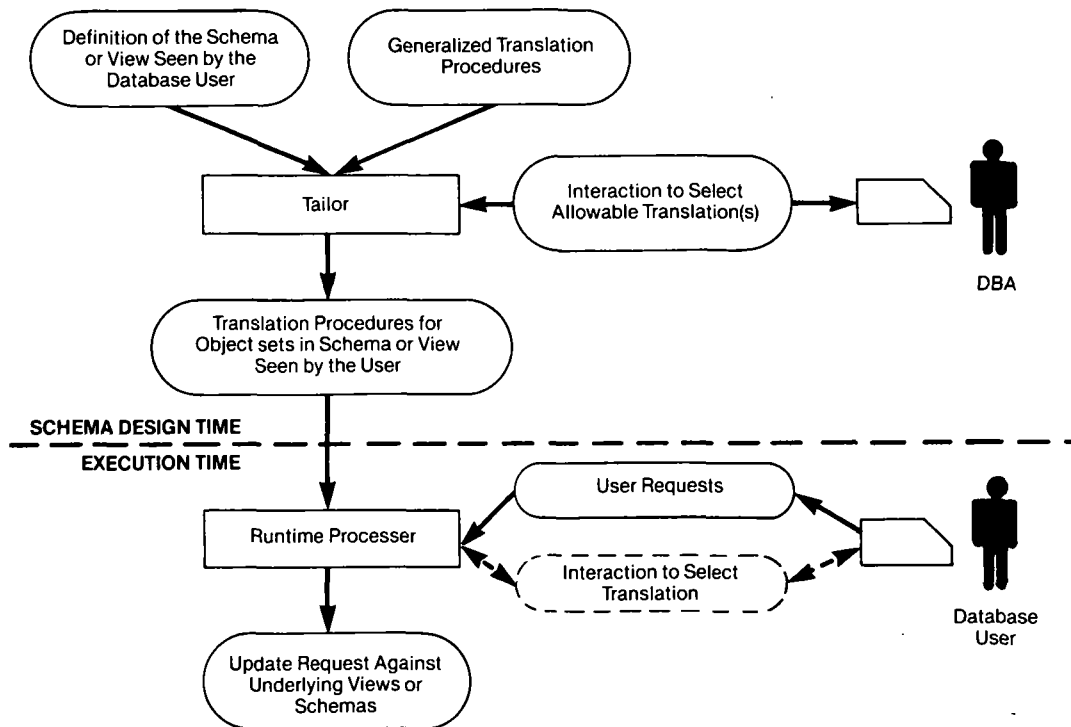


Figure 3

REFERENCES

- [CHEN76] Chen, P.P.S., "The Entity Relationship Model- Towards a Unified View Data," ACM Transaction on Database Systems, Vol. 1, No. 1, 1976.
- [ELMA84] Elmasri, R., and Navathe, S. B., "Object Integration in Database Design", Proceedings of IEEE Conference on Data Engineering, Los Angeles, CA, April 24-27, 1984, pp. 426-433.
- [ELMA] Elmasri, R., Hevner, A., and Weldreyer, J., "The Category Concept: An Extension to the Entity-Relationship Model", submitted for publication.
- [HAMM81] Hammer, M., and McLeod, D., "Database Description with SDM: A Semantic Database Model," ACM Transactions on Database Systems, Volume 6, Number 3, September 1981.
- [LARS83] Larson, J. A., and Dwyer, P., "Defining External Schemas for an Entity-Relationship Database", in Entity-Relationship Approach to Software Engineering (C. G. Davis, S. Jajodia, P. A. Ng, and R. T. Teh, eds.) Elsevier Science Publishers B. V., 1983, pp. 347-364.
- [MASU83] Masunaga, Yoshifumi "A Relational View Update Translation Mechanism", IBM Research Report RJ3742(43118), 1/12/83.
- [NAVA82] Navathe, S.B., and Gadgil, S. G., "A Methodology for View Integration in Logical Database Design; Proceedings of the Eighth International Conference on Very Large Databases, Mexico City, September 1982.
- [NAVA84] Navathe, S.B., Sashidhar, T., Elmasri, R., "Relationship Integration in Logical Database Design", Proceedings of the Tenth International Conference on Very Large Databases, Singapore, 1984.
- [STON75] Stonebraker, M.R., "Implementation of Integrity Constraints and views by Query Modification," Proceedings of the ACM SIGMOD Conference, San Jose, California, May 1975.
- [WEEL80] Weeldreyer, J.A., "Structural Aspects of the Entity-Category-Relationship Model of Data," Report Hr-80-251, Honeywell CCSC, Bloomington, Minnesota, March 1980.

RED1: A Database Design Tool
for the Relational Model of Data*

Anders Bjornerstedt
Christer Hulten

SYSLAB
Department of Information Processing & Computer Science
University of Stockholm
SWEDEN

ABSTRACT

A design tool, RED1 (RELational Database design aid version 1), is presented. The objective of RED1 is to:

- a) give the designer a means of prototyping a small, but representative data base and to actually execute queries and transactions. The designer will then be able to test whether a design (consisting of relation schemes and constraints) performs as expected under transactions and queries. Such behavior is difficult or impossible to determine analytically. As the designer changes the design, RED1 will monitor the consistency of schema and transactions.
- b) give the designer a set of analytical support functions on the meta data-base (schema) level.

The programming language Prolog [CLOC81] is used for the implementation of the tool. This simplifies the programming of the tool, makes modifications of it easy, and gives a natural connection to predicate logic.

1. Introduction

Systems for managing large scale databases under the relational model have become commercially available and therefore the value of design tools for the relational model is obvious. RED1 [BJOR83] is a tool which supports the user in schema and transaction design. It contains a number of small procedures or operations which are used in a free and interactive manner. The operations can be divided into two categories, depending on their objective:

- a) Operations which support a prototyping approach to database design. They collectively provide the functions of a small scale DBMS. That is, the DBMS is functionally fully relational [CODD82, DATE83], even if it is not a

*This work is supported by the National Swedish Board for Technical Development (STU).

'true' DBMS in terms of scale. By using these operations the designer can define a database (in terms of domains, relations, constraints, transactions and views), and experiment with a small extension. The database definition (schema) is itself represented relationally. RED1 is thus in a sense self-descriptive.

- b) Operations which are used to analyze properties of the schema and to suggest designs. For example there are operations for testing whether a relation scheme is in a certain normal form [SAGI80], and for synthesizing relation schemes in third normal form [BERN76]. These operations are presently based purely on the analysis of database dependencies (functional and multivalued).

The user defines relation schemes, transactions, data dependencies and integrity constraints. He may also build the database extension. The same query and data manipulation operations (relational operators) are used for accessing schema definition, transaction definitions, currently defined dependencies, as well as the basic extension of the database being defined.

The programming language Prolog [CLOC81] is used for the implementation of the tool. Prolog is a small, interpretive, high level language, with a very simple syntax. The interpreter is basically a mechanical theorem prover for a restricted first order predicate calculus (Horn Clauses). This simplifies the programming of the tool, makes modifications and extensions to it easy, and gives a natural connection to predicate logic. The last aspect is most apparent in connection with the expression of constraints.

The database which the designer builds in RED1 is in general defined with more precision than that which can be handled by today's large scale relational DBMS. Explicit constraints of many kinds (including dynamic constraints) are allowed. Constraints are expressed in a restricted first order language, which is similar to but not as terse as pure Prolog. The facilities of RED1 should encourage the designer to experiment with different solutions, to "cut and paste" relations. The analytic operations presuppose some elementary knowledge of relational theory [ULLM82, MAIE83], on the part of the designer.

2. Prototyping with RED1

The operations of RED1 operate on the system relations and user defined relations which all are maintained in primary storage. The relations are loaded/saved at the beginning/end of a session with RED1. A schema is created by:

- o Defining data types.
- o Defining domains on data types.
- o Defining attributes on domains.
- o Defining relations on attributes.
- o Defining constraints, transactions and views on relations.

Actually a strict bottom up approach is not required or recommended. The system constraints and system transactions(1) of RED1 will ensure that the database definition will not be left in an incomplete state.

RED1 has an integrity monitor which evaluates relevant constraints after a transaction is completed. If an integrity violation occurs, the designer is notified and either the database state is automatically reset to the state that existed before the transaction, or another (presumably correcting) transaction is automatically invoked.(2)

A data type defines how elements of a domain, defined on that data type, are represented in the database. A domain, on the other hand, is an abstraction on the level of the semantics the designer is trying to express. By making the distinction between domain and data type we avoid confusing syntactic similarity with semantic similarity, e.g., adding a person's salary with his shoe size should not be possible just because they are represented identically. It is our experience that the number of data types needed is usually few compared with the number of domains. RED1 has a few predefined data types which we think suffice for the designer who is concerned with logical design. The distinction between domain and data type also encourages the designer to create as many domains as needed. Creating a new domain simply consists of deciding on a suitable name and stating which data type it is to be defined on. The handling of data type and domain definition in RED1 is based on McLeod's approach [MCLE76, HAMM75].

Attribute names are global in the database. An attribute can appear in several relation schemes, but can only be defined on one domain. An attribute represents one 'role' for a domain in the database.(3)

Relations are defined by a relation name, the set of attributes included in the relation scheme, and the subset of the attributes which is to be the primary key.

Views, constraints and transactions are defined in terms of relations and operations on these. Views and transactions can also be defined in terms of previously defined views and transactions (operations) respectively. The primitive (predefined) operations of RED1 can best be described as a combination of the relational algebra and the relational calculus. Primitive operations can

(1) System-relations, constraints and transactions, could be called meta-relations, constraints and transactions, i.e., they hold and operate on the schema. An example of a system transaction would be "create relation", which would operate on system relations "relation," "attribute," etc. and involve system constraints expressed over those relations.

(2) The risk of infinite recursion is avoided by letting the designer intervene at each violation before a correcting transaction is executed. The designer can then choose to back up instead of allowing the correcting transaction to proceed.

(3) For an explanation of 'role' see [DATE81] p. 86.

be added or modified by the designer, although this does require knowledge of Prolog programming. All the primitive operations of RED1 are regarded as transactions by RED1 if they are used in isolation. More complex operations can be built by composition over primitive or complex operations, and named and stored in the database. The composed operation when executed in isolation will then be regarded by RED1 as a transaction, i.e., the unit of consistency.

3. Explicit Constraints

A constraint is defined by:

- o An integrity assertion.
- o One or more enforcement specifications.
- o For each enforcement specification, a violation action.

The integrity assertion is a labeled statement which must hold in every consistent database state. In RED1 assertions are closed formulas of first order logic. However they are expressed in a restricted way. To achieve goal directness, that is a procedural interpretation, and a simpler language, no explicit quantifiers are used. Instead, quantification of variables is determined by the context in which they appear. Variables always range over the domain of attributes in relations. Relation schemes correspond to predicates in the formula. A simple example would be:

```
shipment(partnum:X, suppnum:Y)
=>
part(partnum:X) & supplier(suppnum:Y)
```

which roughly states that "for every shipment, the part and the supplier must exist". The variables X and Y would in this case be universally quantified. Dynamic constraints, which are restrictions on allowable transitions between database states, are expressed using dynamic relations. With every defined relation, RED1 associates two dynamic relations. Every tuple inserted into a relation by a transaction is also inserted into one of the associated dynamic relations. Every tuple deleted from a relation is inserted into the other associated dynamic relation. When the transaction is completed and integrity is to be determined, the total change to the database is reflected in the dynamic relations. Dynamic constraints are then expressed as static constraints on dynamic relations. If no constraints are violated, the transaction can be accepted and RED1 deletes the extensions of the dynamic relations. This idea is obtained from Nicolas and Yazdanian [NICO78]. An example of a dynamic constraint would be:

```
del_employee(empno:E, salary:S) &
ins_employee(empno:E, salary:Snew)
=> Snew > S.
```

which states that "the salary of an employee cannot decrease".

An enforcement specification is a statement about when a constraint should be evaluated. It would be very inefficient (even in a prototyping situation) to evaluate all constraints after every transaction. A constraint should be tested if and only if the transaction included operations which could have violated the constraint. Enforcements are specified by relating an operation (name), a relation (name) and a constraint (label) to each other. Whenever an operation (primitive or composed) has been performed on one or more relations, all constraints related to this operation and relation are added to a list of constraints to be enforced after the transaction is finished.

To each enforcement specification the designer can associate a violation action stating what RED1 should do in case of violation. The default action is to back up and generate an error message consisting of the label of the constraint which has been violated. The designer can optionally specify his own error message, which may be more explanatory, or indicate that a certain operation (correcting transaction) is to be performed.

4. Analyzing and Synthesizing Schemes

Independently of prototyping the database and transactions the designer can analyze alternative relation schemes on a purely intentional level. RED1 has operations for defining functional and multivalued dependencies by relating sets of attributes with each other. A relation scheme can be tested to see if it conforms to a certain normal form (2nd, 3rd, bcnf & 4th). Furthermore, it can be determined if a particular data dependency logically follows from previously defined dependencies. Given a set of functional dependencies it is possible to mechanically generate a set of relation schemes in third normal form [BERN76]. The designer can easily focus on a subset of the attributes and dependencies defined so far that is considered important for the moment.

5. Concluding Remarks

RED1 is a tool for logical design of relational databases, which tries to integrate analytical and constructive functions with a prototyping approach. It is based on a small relational DBMS with an integrity monitor that aids the designer both in creating a consistent design and testing proposed transactions. However, RED1 has not yet been tested on any larger practical design problem. Such a test may influence its future development. One such development would be to provide automatic schema and application program generation for large scale production DBMS's, with the specified schema and transactions of RED1 as input.

References

BERN76. P.A. Bernstein, "Synthesizing third normal form relations from functional dependencies," ACM TODS Vol. 1(4) pp. 277-298 (Dec. 1976).

- BJOR83.** A. Bjornerstedt and C. Hulten, "RED1 A database design tool for the relational model," Report No. 18, SYSLAB, Dept. of Information Processing & Computer Science, Stockholm, Sweden (March 1983).
- CLOC81.** W.F. Clockskin and C.S. Mellish, Programming in Prolog, Springer-Verlag, Berlin Heidelberg (1981).
- CODD82.** E.F. Codd, "Relational Database: A Practical Foundation for Productivity," Communications of the ACM Vol. 25(2) (February 1982). The 1981 ACM Turing Award Lecture.
- DATE81.** C.J. Date, An Introduction to Database Systems, third edition, Addison-Wesley (1981).
- DATE83.** C.J. Date, An Introduction to Database Systems, Addison-Wesley Systems Programming Series (1983). Volume II
- HAMM75.** M.M. Hammer and D.J. McLeod, "Semantic integrity in a relational database system," Proceedings VLDB, pp. 25-47 (Sept 22-24, 1975).
- MAIE83.** D. Maier, The Theory of Relational Databases, Computer Science Press, Rockville, MD. (1983).
- MCLE76.** D.J. McLeod, "High level expression of semantic integrity specifications in a relational database system," MIT/LCS/TR-165, MIT, Cambridge Massachusetts (1976).
- NICO78.** J.M. Nicolas and K. Yazdanian, "Integrity Checking in Deductive Databases," pp. 325-344 in Logic and Databases, ed. H. Gallaire & J. Minker, Plenum Press, New York and London (1978).
- SAGI80.** Y. Sagiv, "An algorithm for inferring multivalued dependencies with an application to propositional logic," Journal of the ACM Vol. 27(2) pp. 250-262 (April 1980).
- ULLM82.** J.D. Ullman, Principles of Database Systems, Second Edition, Computer Science Press, Rockville, Maryland 20850 (1982).



An Automated Logical Data Base Design and Structured Analysis Tool

Robert M. Curtice
Arthur D. Little, Inc.
Acorn Park
Cambridge, Massachusetts
(617) 864-5770

Abstract

A brief description of the main features and examples of the outputs from an automated tool are presented. The tool is an aid in logical data base design and in structured systems analysis.

Introduction


ADL IRMA (Information Resource Management Aid) is a system designed to aid in logical data base design, structured system analysis, and strategic systems planning. It has been under development at Arthur D. Little, Inc. for more than four years and has been successfully applied to more than a dozen industrial and government situations. ADL IRMA operates on the IBM Personal Computer (or compatible) and utilizes 128K bytes of memory and the graphics option.

Logical Data Base Design Methodology

ADL IRMA is designed to support a particular methodology for logical data base design, and to appreciate its features, a brief introduction to this methodology is required. A more substantial and rigorous description of the method can be found in [CURT82].

By logical data base design (sometimes called conceptual design) we mean a description of data which is free from any physical implementation considerations or specifications. The intent is to define what the data is, what it means, and how it is interrelated. The resulting logical data base design does not imply implementation using Data Base Management Software. The level of definition dealt with by the methodology is appropriate for data modeling (in which case the design does not imply implementation at all), or for the logical level design of an application's data base or files.

As with most logical data design approaches, this method begins with the identification of types of objects, called entities, about which data needs to be recorded in a data base or file (or modeled in a data model). For example: Employee, Vendor, Department, Part, Customer, Account, and Policy

 is a ServiceMark of Arthur D. Little, Inc. Hereinafter referred to as ADL IRMA.

are typical entity types. In our terminology, each of these entity types becomes a "key" in a data structure chart, and is depicted graphically at the top of such a chart in a box. Subordinate to each key, we will record the elements of data which are about that entity.

Four types of data elements are permissible:

- (1) The regular data element, which characterizes or describes the key. It is depicted in an oval shape on the data structure chart.
- (2) The multiply-occurring regular data element, for cases in which more than one value of the characteristic may apply to one instance of the key. This is depicted as a three dimensional oval.
- (3) The associator data element, which serves to relate the key of this chart to the key of another chart, or perhaps to itself. It is shown in a box on the data structure chart.
- (4) The multiply-occurring associator, used when the key may be related to more than one instance of the related key, in the same relationship. It is depicted in a three dimensional box.

There are three remaining fundamental aspects to the methodology. If the key element may validly exist with no value of the subordinate element (regular or associator) then we insert a small "o" in front of the element's shape on the chart to signify it as optional. Also, the elements subordinate to the key may be arranged in a tree structure with the following interpretation: Each node (data element) in the tree becomes a created entity type, with subordinate data elements being "about" that newly created entity.

Finally, we always observe the reversal rule: If Key A relates to Key B through an associator in A's structure chart, then we must supply another associator in B's chart to relate B to A in the inverse relationship. This not only provides a cross-reference of all associators, but forces the designer to specify the one/many and optional/mandatory choices for the inverse relationship.

All of these components of the methodology are exhibited in Figure 1 which shows three keys in an oversimplified "toy" data base. The interpretation of the first of these three data structure charts is as follows:

For each Part there must exist one and only one Name of Part. Each Part may be related to none, one, or more than one Approved Vendor. For each Approved Vendor which a Part does have, there must be one and only one Vendor's Part Number. Finally, each Part may be associated with one or more Purchase Orders Issued for that Part, but some parts may exist with no Purchase Orders Issued.

The methodology makes use of the concept of domain. We distinguish a data element (each shape on a data structure chart) from a domain which is the set of values or identifiers which a data element can assume. Thus Part (as a key), Part Vendor Supplies, and Part Ordered are three distinct data elements all defined over the same domain of Part Number.

A 'TOY' DATA BASE

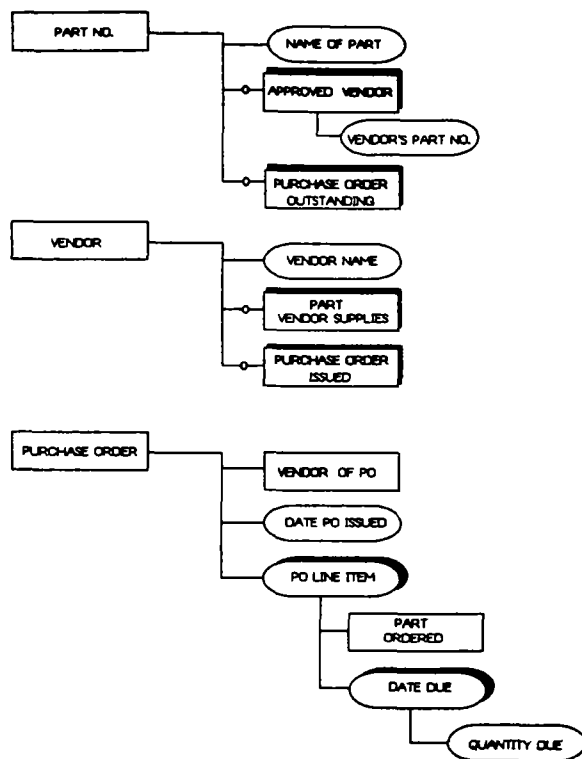


Figure 1
Example of Data Structure Charts

The reader may want to satisfy himself that an unambiguous interpretation can be made for the remaining two keys in Figure 1. Also notice that each associator (box subordinate to a key) does indeed reference a legitimate key, and that all such associations are reversed (e.g. Part Vendor Supplies reverses Approved Vendor). The element Vendor's Part No. is subordinate to both Part and Approved Vendor; as such it is a characteristic of a created entity which is the idea of a Part as it is supplied by an Approved Vendor.

Finally, students of relational data base theory will observe that each branch of the tree under a key forms a third normal form (but not first normal form) relation, since the concept of functional dependency is imbedded into the logic of data structure chart formation.

ADL IRMA as a Logical Data Design Aid

The heavy involvement of user personnel in the design process is central to the way in which the methodology is applied. Thus the approach must be explainable to non-data processing personnel in 15 minutes or less. The actual design process is conducted in intensive working sessions during which the data structure charts are initially prepared, and reviewed and modified in iterative sessions. A major benefit of the ADL IRMA system is that it permits entry and maintenance of all the data associated with a logical data design, and can rapidly produce up-to-date data structure charts for use by all design participants. A sample of the computer generated data structure chart is shown in Figure 2.* In this output, note that the domain of each element is explicitly listed and that a text definition of each data element is also printed.

* This chart is drawn from the sample problem set forth in [VANG82].

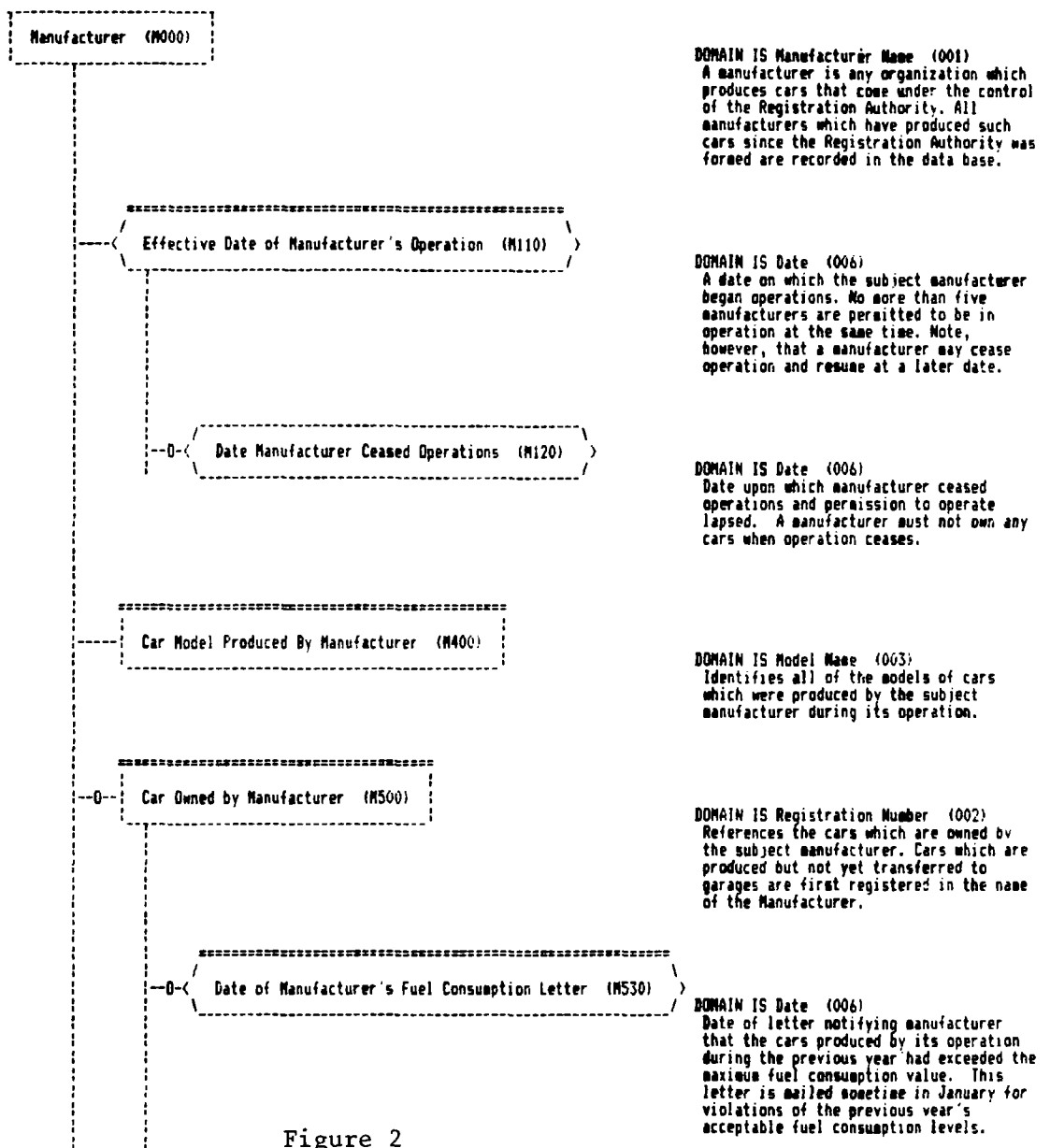


Figure 2
Data Structure Chart Output from ADL IRMA

Data structure charts can be immediately displayed on the monitor using ADL IRMA. Modification to the specifications of a data element and the structure is made in an interactive mode. Moving an element in the structure automatically moves the entire sub-tree subordinate to that element. In addition to the ability to easily modify the data structure charts and print them, ADL IRMA produces a variety of other reports along with appropriate error messages indicating a violation of the methodology. Among these are:

- A domain report which cross references all data elements defined over each domain.
- An element report which flags errors such as an invalid domain, or an associator which has no valid reference.
- A report and associated file which can be used to load one of a number of DBMS data dictionaries on the mainframe.

ADL IRMA as a Structured Analysis Aid

The PC user may create a data flow diagram on the monitor and to print it using a standard dot matrix printer. In order to provide sufficient room for a complex diagram, four "windows" each slightly smaller than the monitor screen may be utilized. Moving the cursor across the window boundary automatically displays that window.

The basic operation of the system in this mode is to place the cursor at the desired position on the diagram and press a single key to generate the appropriate symbol for a process, datastore, external entity or data flow; these are the basic primitives of structured analysis [GANE79]. Then the identifier of the process, data store, external entity or data flow is entered; the system looks it up in its files, and if valid, places the full name on the diagram. An example of the resulting diagram is presented in figure 3.

Processes, external entities and data flows are described to the ADL IRMA system prior to creating the diagram. Each datastore is equivalent to a "key" as previously described. This approach integrates the structured analysis support with the logical data design. Additionally, the data flow descriptions can reference the data elements which appear on them; again data elements are precisely those described during logical data design.

In addition to the data flow diagram itself, ADL IRMA produces a series of reports about processes and data flows. If a key data element is referenced as the source or destination of any data flows, then these are cross referenced in the corresponding data element report.

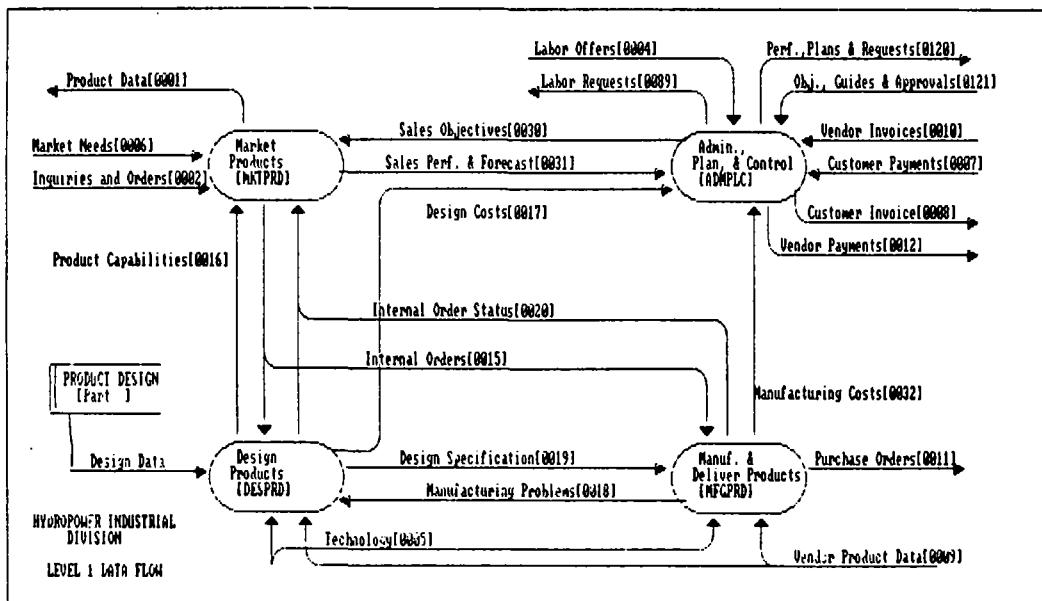


Figure 3

Sample Data Flow Diagram Output

Experience with ADL IRMA

As previously mentioned, ADL IRMA has been utilized in more than a dozen actual design and planning situations. The major benefit of the system is the ability to maintain consistent documentation about a rapidly evolving design, and the facility to easily modify specifications and obtain new documentation (including graphics) with short turnaround. For example, if a data flow appears on several data flow diagrams when its name is changed once, then subsequent printings of each diagram will reflect the current name. Typically, both data base and data flow designs are initially prepared on flipcharts and entered into the ADL IRMA system by a para-professional. Viewgraphs prepared from ADL IRMA output are then marked up and used to modify the specifications in the system in an iterative manner.

A secondary benefit of the system is the checking and cross referencing that it performs. These features help insure that the final design is consistent and within the rules of both the logical data design and structured analysis methodologies.

Finally, one of the interesting side benefits of the system is that it has allowed us to build up a number of data designs in machine readable form which cover a cross section of industries and data problems. From this "data base" of data base designs, we are able to collect and analyze properties across designs, which few people are likely to be in a position to think about. Empirical data about the properties and statistical attributes of data base designs are not well developed. We plan to report on the results of these investigations in the near future.

References

- [GANE79] Gane, C. and T. Sarson, Structured Systems Analysis; Tools and Techniques, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.
- [CURT82] Curtice, R., and P. Jones, Logical Data Base Design, Van Nostrand Reinhold, Inc., New York, NY, 1982.
- [VANG82] Van Griethuysen, J. (ed) Concepts and Terminology for the Conceptual Schema and the Information Base, International Standards Organization, ISO/TC97/SC5-N 695, March 15, 1982.

An Overview of Research in the Design of Distributed Databases

Stefano Ceri, Barbara Pernici, and Gio Wiederhold*

Dip. di Elettronica, Politecnico di Milano, P.za L.da Vinci 32, Milano Italy; (2)236-7241

*Computer Science Dep., Stanford University, Stanford CA 94305; (415)497-0785

Abstract

Distributed database design is viewed as consisting of three phases, partitioned to minimize their interaction. The initial and final phases relate strongly to traditional logical and physical database design. The second phase is itself subdivided into fragmentation and allocation tasks. We report on methods appropriate to those tasks and their interaction, based on research work done at Stanford, under the KBMS project[†], and in Italy, under the DATAID project[‡].

1. Introduction

Several prototypes and commercial systems for accessing distributed databases have been developed. Rapid growth in their application is predicted. Most operational applications have been handcrafted, and appear difficult to optimize initially and maintain in the long term. The problem of designing effective applications using these systems is thus becoming more and more important.

2. Framework

The design of a database application in a nondistributed environment is typically subdivided into two distinct phases. The logical design aims at producing a representation (schema) of all the data and their relationships required by the application. The physical design aims at selecting the physical storage structures which allow efficient access to the data. In this paper we do not address the techniques which are required for these two phases, which are described, for instance, in [Ceri 83] and [Wied 83].

The design of a distributed database application inserts a phase, called distribution design, between the logical and physical phases. Distribution design aims at determining the distribution of the data to the sites of the distributed database. Two possible approaches exist:

a: Top-Down Approach.

This approach is typical of a distributed database system which is developed from scratch. An integrated database schema which models all the data of the distributed application is initially designed; the schema is then partitioned into several subschemas, one at each database site, and data are distributed accordingly. In particular, global data objects (relations) belonging to the integrated schema are decomposed into fragments, and fragments are then assigned to the subschemas at the different sites. Fragmentation is an important aspect of distributed databases, because it allows the user to write programs which use global data objects and are transparent to data distribution and fragmentation, while the system controls the placement of fragments in the most efficient way. Top-down distribution design is therefore decomposed into two phases:

a1: The design of fragmentation. The goal of this subphase is to determine fragments as "units of data distribution", i.e., portions of global data objects which can conveniently be distributed. The design of fragmentation requires an understanding of the properties of global

[†] sponsored by DARPA contracts N39-82-C-25 and N39-84-C-211

[‡] sponsored by the National Research Council

data objects and of the applications which use them; therefore, the design of fragmentation can be regarded to be the logical phase of the distribution design.

a2: The allocation of fragments. The goal of this subphase is to determine the sites where fragments should be allocated, possibly by replicating fragments over multiple sites. Allocation of fragments aims at the optimization of the distributed database performance, and can therefore be considered a “physical” decision.

With this approach, distribution design follows the logical design of the global schema, and the physical design at each site follows, in turn, the design of data distribution., as shown in Fig. 1.

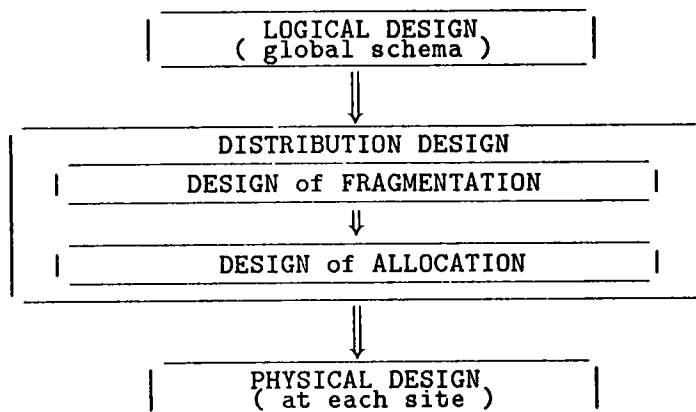


Fig. 1: Phases in the Design of a Distributed Database

b: Bottom-Up Approach.

This approach is typical of distributed databases which are developed as an aggregation of pre-existing databases. With this approach, schemas representing the portion of data stored at each individual site constitute the starting point, and distribution design consists in identifying the data which are common to the distinct schemas, and giving them a common representation. The integration process should solve “conflicts”, i.e., different representations of the same data in different schemas. In this paper, we concentrate on the top-down approach; the bottom-up approach was studied in [Daya 82], and the integration of different schemas is described in [Bati 83], [Nava 82], and [Nava 84b].

3. Design Problems

In this section, we review the design problems which arise in the top-down approach; in the next two sections, we will indicate the approaches developed to the solution of each individual problem, and to the integration of all the solution methods within a single framework.

Let us assume the relational model for the global schema. With the relational model, two types of fragmentations are possible:

a: Horizontal. Fragments are subsets of the tuples of a global relation. Each horizontal fragment of a same global relation has the same relation schema. Each fragment is associated with a predicate (called “qualification”) which indicates the distinguishing property possessed by the tuples of that fragment. In this paper we assume disjoint fragments, and therefore also the predicates should be disjoint. Horizontal fragments can be defined from global relations using selections and semi-joins (see [Ceri 84]).

b: Vertical. Fragments are defined by projecting global relations over subsets of their attributes. In order to make the projection lossless the key attribute or a tuple identifier is included

within each fragment. A vertical fragmentation can be nonoverlapping, when each nonkey attribute belongs to one and only one vertical fragment, or overlapping, when some nonkey attributes belong to two or more vertical fragments.

Mixed fragmentations are built by combining horizontal and vertical fragmentation. The number of fragments becomes rapidly greater than the number of relations in the database. Correspondingly, the problem of optimal design becomes even less tractable on a quantitative basis. Overlapping or replicated fragments increase complexity further.

In the allocation of fragments, two alternatives can be considered:

a: Nonreplicated allocation

Here each fragment is allocated to one site. A nonreplicated allocation is required if the system does not support data replication. The optimization criteria which is primarily used for solving this problem is the maximization of processing locality of applications.

Capacity constraints within the sites are commonly ignored. This assumption is certainly valid whenever distributed transactions are a small fraction of the load submitted to a site.

b: Replicated allocation

Here a fragment is allocated to one or more sites. In this case, the degree of replication of each individual fragment is a variable of the problem. Update of replicated data has stringent requirements if global consistency is to be maintained. Typically, the benefit of replication is higher if the ratio between retrieval and update accesses to the fragment is high. If update is infrequent capacity constraints may become limiting.

Replication may have the auxiliary benefit of redundancy, which can allow recovery after loss of data within one site [Apcers 84], [Mino 82]; this additional benefit is hard to assess. If replication is to be used to this end, then a minimum replication factor may be initially established.

Given the above classification of problems, the top-down approach to distribution design consists of solving for each relation the following individual design problems: horizontal partitioning (H), vertical partitioning (V), nonredundant allocation (A), redundant allocation (R). In current practice R is often a post-process on the allocation produced by A. This leads to sufficient, but does not guarantee optimal results.

4. Solutions to Individual Design Problems

In the past two years, we have concentrated on the solution of some combinations of the above design problems. The extreme complexity of the mathematical models required for solving the problems has led to the development of heuristic approaches.

In [Ceri 83] an analytical model was presented for selecting among alternative horizontal fragmentations of global objects. Users specify alternative fragmentation criteria, and also indicate for each fragment a single preferred allocation site in case the fragmentation criterion is selected by the solution method. The solution is produced by a linear integer program; in the solution, each relation is either fragmented with the best possible fragmentation or stored whole at one site. Thus, the model solves the H + A problems. Heuristics are added in order to generate replicated allocations starting from the nonreplicated solution, thus solving the H + A + R problem.

In [Nava 84a] several algorithms are shown for the vertical partitioning of relations in different design contexts. The basic method used by all algorithms is based on the notion of affinity among attributes, which measures the common usage of any two attributes. Using affinities, attributes are clustered together, and algorithms then determine the best vertical partitioning by considering all clusters. One heuristic algorithm is developed for the vertical partitioning of relations with nonredundant allocation, solving the V + A problem. Another algorithm is used for generating replicated allocations starting from the nonreplicated solution, thus solving the V + R problem.

In [Sacc 83] the problem of distributing the database on a cluster of processors was considered. The motivation of such an architecture is increasing performance, availability, and reliability. The problem differs from the standard distribution design problem, because the applications need also to be allocated to processors; thus, an object to be allocated can be either an application or a fragment. The model must now consider capacity constraints for the CPUs, processors' I/O, and network communication. The definition of fragments is an input for the solution method, which produces a nonreplicated allocation of objects - fragments and applications - thus solving the A problem. The solution uses two well-known heuristics in a novel combination: a greedy algorithm combines pairs of objects based on maximal benefit, and the combination is then allocated to sites using a first-fit bin-packing heuristic.

5. Integration of Solution Methods

Our current research interest is in the integration of several solution methods within a single framework. However, the solution of design subproblems is hampered by their extreme computational complexity (N-P completeness), as shown in [Aper 84] and in [Sacc 83]. Thus, simple merging of our algorithms and solution spaces would be computationally unfeasible for problems of realistic sizes.

The complexity of models is not fictitious; it is due to the intrinsic complexity of the overall design problem and the quantity of objects. The numbers of sites, relations, applications, horizontal fragments, vertical fragments, and copies of fragments considered in a particular design problem can be rather large, and each of these numbers has a potential for generating a combinatorial explosion of the solution space.

We envision the development of an environment of tools which are able to solve partial problems, under the control of an expert database designer. The experience of the designer will be primarily useful in selecting the order of evaluation and iterations of individual problems (e.g., H + V + A + R is a likely combination), and in interpreting the results returned by each tool. In general, the following precedence relations hold between the evaluation of problems:

$$A < R \qquad H, V < A, R$$

The designer can decide to repeat design steps based on the feedback of previous computations.

Moreover, the designer will be able to give the tool indications which might reduce the dimensions of the design space (for instance, by indicating some promising initial fragmentations and allocations). Solutions which violate capacity or common-sense constraints can also be avoided.

The major goals of such an environment would be:

- a. To include tools for the solution of each of the design problems we defined (H, V, A, R).
- b. To allow for exchangeability of design methods and algorithms.
- c. To have a common basis of evaluation of execution strategies over distributed databases, used by all the tools. This evaluation of execution strategies might be substituted by the use of the optimizer of one particular distributed database system.

We have started working towards such an environment by outlining the architecture of the design system, and designing and implementing some portions of it. A similar approach is also assumed in a project conducted at the Computer Corporation of America, described in [Rein 84].

In order to accomplish goal **b** above and to accommodate the use of our previous solution algorithms, the architecture of the system must be very modular. The design system should provide functions for designing the fragmentation, for selecting and allocating fragments, for performing query optimization, and for evaluating an execution strategy. These functions are shown in Fig. 2.

Tools and solution models should not necessarily be in one-to-one correspondence with these functions. Thus, for instance, the algorithms described in [Ceri 83] and in [Nava 84a] cover both

fragmentation and allocation design; we do not plan to decompose those algorithms in order to obtain a more layered software architecture.

As shown by Fig. 2, the tool environment should also be capable of performing a query optimization function. This is necessary: in order to evaluate a specific fragmentation and allocation, it is required to know the query execution strategy that will be developed by the optimizer of the actual distributed database system. Unfortunately, this requirement compromises the generality of the tool environment, because very many different approaches exist to query optimization. Moreover, the optimization should be relatively efficient, even if suboptimal, because it has to be repeated for each candidate solution produced by the fragmentation and allocation design and for each considered query and update transaction.

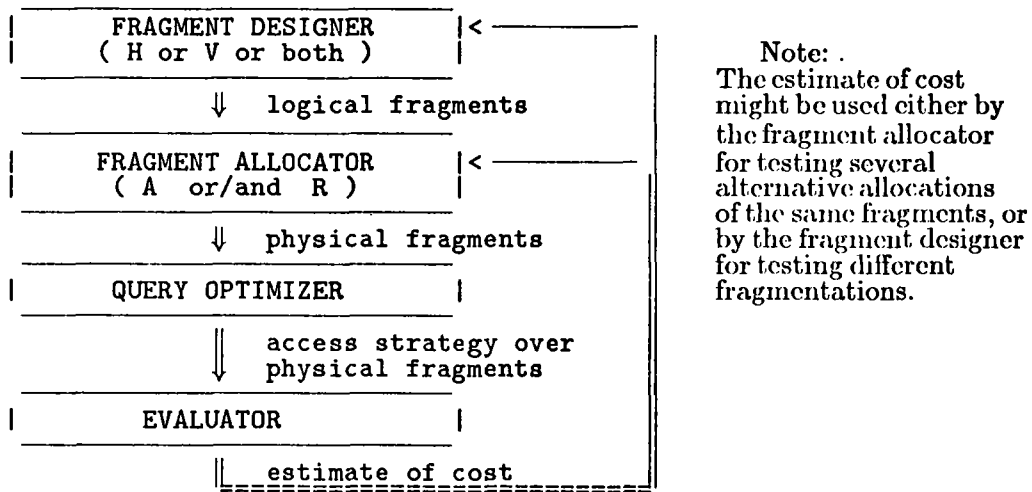


Fig. 2. Functionalities to be Provided by a Design Tool for Distributed Databases

In [Sacc 83] the query optimizer was outside of the design system, since the actual R* optimizer was to be used. In such an architecture the cost of repeated evaluation is high so that four tactics were considered. Greedy allocation was driven by selecting the pair of objects which benefitted most from placing them at the same node. Minimal query evaluation cost is obtained by computing the object² benefits for all pairs only once. The benefits for the selected pair versus all other objects are added to estimate benefit values for the next object allocation. This technique leads to imprecision as the allocation progresses. The best approach, full benefit recomputation throughout the iterations, requires object⁵ evaluations.

The lower-level function in Fig. 2 consists in providing an evaluator for a specific query execution strategy on a specific data distribution and allocation. The evaluator should be able to indicate specifically the costs which are due to the execution of the query, in terms of CPU, I/O, and messages required; it should also be possible to indicate the distribution of CPU and I/O costs to the various sites, and of transmission costs along the various links.

We have recently designed and implemented an evaluator for distributed database applications. The evaluator uses an algebraic representation of queries, which can include selection, projection, join, and union operations. It also considers updates, with update specifications which apply directly to tuples of fragments. The evaluator receives as input the description of the query in terms of an operator graph, similar to an operator tree but with the additional capability of using

the result of an expression more than once. It produces as output a detailed description of CPU, I/O and transmission costs. Each fragment is described by giving its profile; profiles include the cardinality (number of tuples) and, for each attribute, the size (number of bytes), the number of distinct values, and the minimum and maximum value. In order to evaluate the profiles of relations produced at each operation, we used the formulas from [Ceri 84].

After completing the evaluator, we started integrating the vertical fragmentation algorithms described in [Nava 84a] within the new environment. The analytical formulas used by the vertical partitioning algorithm are replaced by calls to the new evaluator. The vertical fragmentation algorithm produces at each iteration a possible vertical fragmentation and allocation; in order to provide an interface between the vertical fragmentation algorithm and the evaluator, which evaluates specific execution strategies over a distributed database, we have designed a small query optimizer, which transforms the queries over global relations into queries over fragments, and then allocates each operation of the queries over fragments to the most convenient site; we are currently implementing the query optimizer. This experiment will allow us to test the vertical fragmentation algorithms with more accurate cost evaluations, and at the same time will give us insight on the practicability of our approach towards the integration of distribution design tools.

Acknowledgments

Sham Navathe, from the University of Florida, Giuseppe Pelagatti, from the Politecnico di Milano, and Domenico Saccá, from C.R.A.I. (Cosenza), have contributed to our past work.

6. References

- [Aper 84] P. Apers and G. Wiederhold: "Transaction Classification to Survive a Network Partition"; Stanford University, submitted for publication, 1984.
- [Bati 83] C. Batini, M. Lenzerini, and M. Moscarini: "Views Integration"; in [Ceri 83a].
- [Ceri 82] S. Ceri, M. Negri, and G. Pelagatti: "Horizontal Partitioning in Database Design"; *Proc. SIGMOD Conference*, June 1982.
- [Ceri 83a] S. Ceri (ed.): *Methodology and Tools for Database Design*; North Holland, 1983.
- [Ceri 83b] S. Ceri, S. Navathe, and G. Wiederhold "Distribution Design of Logical Database Schemas"; *IEEE Transactions on Software Engineering*, Vol. SE-9 no. 4, 1983.
- [Ceri 84] S. Ceri and G. Pelagatti: *Distributed Databases: Principles and Systems*; McGraw-Hill, 1984.
- [Daya 82] U. Dayal and H. Y. Ilwang: "View Definition and Generalization for Database Integration in Multibase: A System for Heterogeneous Databases"; *Proc. Sixth Berkeley Conf. on Distributed Data Management and Computer Networks*, 1982.
- [Mino 82] T. Minoura and G. Wiederhold: "Resilient Extended True-Copy Scheme for a Distributed Database System"; *IEEE Trans. on Software Engineering*, Vol. SE-8 No. 3, May 1982.
- [Nava 82] S. B. Navathe and S. Gadgil: "A Methodology for View Integration in Logical Database Design"; *Proc. 8th VLDB Conference*, Mexico City, Sept. 1982, pp. 142-164.
- [Nava 84a] S. B. Navathe, S. Ceri, G. Wiederhold, and J. Dou: "Vertical Partitioning Algorithms for Database Design"; to appear in *ACM-TODS*, vol.9 no.4, Dec. 1984.
- [Nava 84b] S. B. Navathe, T. Sashidhar, and R. ElMasri: "Relationship Merging in Schema Integration"; *Proc. 10th VLDB Conference*, Singapore, Aug. 1984.
- [Rein 84] D. Reiner, M. Brodie, G. Brown, M. Chilenskas, M. Friedell, D. Kranlich, J. Lehman, and A. Rosenthal "A Database Design and Evaluation Workbench" Preliminary report, SPOT conference, Göteborg, Sweden, Aug. 1984.
- [Sacc 83] D. Saccá and G. Wiederhold: "Partitioning in a Cluster of Processors"; IBM Report RJ4076, ext. abstract in *Proc. 9th VLDB Conference*, Florence Italy, 1983, pp. 242-247.
- [Wied 83] G. Wiederhold: *Database Design* 2nd ed.; McGraw-Hill, 1983.

Current Research in Database Design at the University of Minnesota

Salvatore T. March
Sunder Mendu
Prashant Palvia *
Michael Prietula **
Dzenan Ridjanovic
Management Sciences
(612) 373-4363
School of Management

John V. Carlis
Diane Beyer
Karen L. Ryan
Computer Science
(612) 376-4592
Institute of Technology
University of Minnesota
Minneapolis, MN 55455

0. ABSTRACT

Research in database design at the University of Minnesota is organized into three major areas: abstraction, optimization, and implementation. Efforts in each area are briefly described and directions for future research are discussed.

1. INTRODUCTION

Databases are formally defined and logically controlled collections of information that are of interest to people within an organization. The effective development of organizational databases requires a three phase process:

1. Abstraction: the determination and documentation of the information, processing and performance requirements of the database,
2. Optimization: the design of the logical and physical database structures that best accomplish these requirements, and
3. Implementation: the creation of computer code to accomplish the design and meet the requirements on a target computer system.

In the following sections we describe research efforts at the University of Minnesota aimed at supporting each of these processes. Since the Optimization process has historically received the most attention in the literature, we begin with research aimed at supporting this process in Section 2. In Section 3 we treat the Abstraction process and in Section 4, the Implementation process. Finally, in Section 5, we present a summary and directions for future research.

2. SUPPORT FOR DATABASE DESIGN OPTIMIZATION

Given a formal description of the informational, functional, and performance requirements for a database (presumably obtained during the Abstraction Process), the Optimization Process seeks to determine a physical database design that most effectively meets those requirements. The term "optimization" is not strictly used in the mathematical sense of maximizing or minimizing a given and fixed objective (e.g., minimize computer resource usage) since there are a number of complex and conflicting design objectives and many nonquantifiable factors. Rather, the process selects "the best" design as a compromise among such factors as operating efficiency (which can be mathematically optimized), design flexibility, simplicity, and development effort.

* Present Address: Temple University, Philadelphia, PA
** Present Address: Dartmouth College, Hanover, NH 03755

In (MARC82, CARL83a) we describe A Database Design Methodology (ADDM) that focuses on supporting the Optimization Process. The methodology is based on a multileveled descriptive model (CARL84) that defines database design problem parameters (e.g., logical data content and user activities) in its logical level and database design solution parameters (e.g., record and data structures) in its physical level. The logical level uses an Entity-Attribute-Relationship data model to describe the logical data structures and a FORAL like language (SENK75) to describe user activities on the logical data. The physical level uses a generic file organization model to parametrically define supported database record structures as well as inter and intra file access paths. The model supports record segmentation and aggregation (MARC83a) and a wide range of data access paths (MARC83b).

Database designs are produced interactively in ADDM. The human designer inputs the design problem (expressed using the logical level of the descriptive model) and interacts with the software to establish constraints on the solution space (e.g., to confine consideration only to those physical design alternatives permitted in a specific DBMS). The software utilizes both heuristic and analytic procedures to generate and search the solution space and produce an efficient database design (CARL83b). The software can also be used to perform sensitivity analysis on various problem and solution parameters.

Current research efforts are aimed at:

1. expanding the scope of problems to which this approach can be applied,
2. improving the human interface to the software, and
3. analyzing the sensitivity of database design parameters to variations in the problem parameters.

As initially developed, ADDM was limited to the design of a centrally stored and maintained database. It did not directly support data redundancy, data distribution or the analysis of backup and recovery policies. These are the major area of functional capability currently under investigation.

March and Scudder (MARC84) analyzed the interactive effects of record segmentation and backup and recovery policies. Segregating data items that have high update frequencies (e.g., current balance, project assignment) from those that have low update frequencies (e.g., name, sex) can significantly reduce the backup and recovery costs. Such cost savings must, however, be weighed against possible retrieval cost increases due to such an arrangement of data items. March and Scudder developed and analyzed heuristic procedures to select an efficient combination of record segmentation and backup and recovery policies. Potential cost savings of up to 40 percent were demonstrated over earlier heuristics that did not consider the interactive effects of record segmentation and backup and recovery. Cost savings for a particular database depend on data item lengths, update and retrieval patterns and frequencies and hardware characteristics.

Mendu (MEND85) is addressing the problem of data allocation in a distributed database environment. Distributed databases offer increased reliability (e.g., "failsoft" capability), faster access to data, reduced communications costs and better overall responsiveness to user needs as compared to centralized systems. The provision of these benefits depends on the effective design of the database, of which data allocation is a critical component. Past research on data allocation has focused, for the most part, on building mathematical models to

allocate entire files to the nodes in a network. Users, however, typically require subsets of files rather than entire files. These file subsets utilize both vertical (record instance) and horizontal (data items) selection criteria. Frequently the location of a user request correlates strongly with the file subset required (e.g., information about customer account balances for customers in the northwest sales region may be nearly exclusively requested by people located in the northwest sales region). Hence, the allocation of (horizontal and vertical) file subsets rather than whole files can lead to significant savings in system operating costs. The change in the unit of allocation from files to file subsets has impacts on other design areas such as the management of the network directory and the implementation of security mechanisms. Both analytic and heuristic algorithms will be developed and tested to efficiently allocate file subsets based on a characterization of user access patterns and network characteristics.

While data allocation is a significant problem in distributed database design, the efficiency of any data allocation is dependent on the way in which queries are processed. Similarly, the solution of this "Query Optimization" Problem is dependent on the data allocation. Approaches to distributed database design have, traditionally, solved one or the other of these problems assuming that the other was fixed. Beyer is investigating the relationships between these problems with the goal of developing heuristic procedures to be imbedded within the ADDM framework to aid a designer in solving the combined problem.

Palvia (PALV84) developed and tested very fast heuristic procedures that produce efficient physical record structures given a problem as defined in the logical level model. He also analyzed the sensitivity of physical record structures to variations in the problem parameters. His analysis includes such factors as: the number of entities in the logical data structure; the number, frequency, retrieval proportions and degree of conflict for retrieval activities; access and storage costs; pagesize; data access methods (dependent on the availability and use of buffers to hold intermediate results) and pointer types. While the complete results are contained in (PALV84), Palvia generally concludes that the structure of database records are most sensitive to the method of data access used and to the activities on the data. In addition, Palvia formulated a simple subcase of the general physical database design problem as a non-linear, zero-one integer program. Efforts are underway to optimally solve this formulation and to apply this approach to the more general problem.

Prietula (PRIE84) analyzed the reasoning processes employed by people as they go about designing physical database structures. He studied both experts and novices and developed a model of human reasoning for database design. He concludes that people use a number of different types of strategies and heuristic procedures to configure database designs. In addition, he suggests that humans rely more on qualitative reasoning rather than quantitative reasoning when designing databases. The implications for database design are that tools aimed at supporting database design must be flexible enough to accommodate different design strategies and heuristics and that quantitative output is insufficient to insure adequate support for database design.

3. SUPPORT FOR DATABASE ABSTRACTION

Before a database can be built to effectively meet the needs of its community of users, these needs must be determined and documented. The database literature has focused primarily on developing formalisms in

which such requirements could be specified rather than on the process of specifying them. Research efforts at Minnesota are addressing both issues.

Databases, no matter how efficiently designed, are of little value unless they contain the correct information and are appropriately organized. Flory, March and Ridjanovic (FLOR84a) have developed a framework for developing the information resource within an organization. They proposed a two dimensional grid which was used to characterize existing database design methodologies. The first dimension delimits four development levels: Corporate Requirements, Application Requirements, Logical Database, and Physical Implementation. The second dimension differentiates the means of development for each level: Models, Methods and Tools. They established goals and objectives for each level and argued that existing methodologies tend to focus on one level only. As a result the process of mapping from one level to another is not well facilitated by any methodology. This is particularly evident at the Corporate Requirements level where planning models are developed but often do not impact the databases or systems that are implemented.

Current research aimed at resolving this problem includes using MIS Analysis methods (such as Business Systems Planning (BSP), Critical Success Factors (CSF) and Means/Ends Analysis as described in (WETH83)) to establish an overall "Information Architecture" and to establish the global database requirements. Semantic data modelling principles (BROD84) can then be used to develop an "abstract" conceptual data model. Refinement of the conceptual model is accomplished using more rigorous and formal data models (FLOR84b) that can then serve as the input to the Optimization Process as described above.

Support tools for building semantic data models are nearly nonexistent. Data dictionaries and tools like PSL/PSA (TEIC77) can serve as containers for the model, but they provide no support for building the model. Similarly, tools to support the building of the formal data models needed for the Optimization Process are also lacking. While microcomputer graphics packages can support the drawing of such data models, they lack the intelligence to support the process of creating the well formed structures. The Functional Dependency Model (FDM) developed by Flory and March (FLOR84b) addresses this issue by providing a mathematical definition of "well formed" data. Algorithms can then be developed to support building data models, not merely drawing them.

A global characterization of user activities is often used to build a data model for a database application (i.e., data dynamics); however, it is typically not until after the data model has been built that a detailed specification of the retrieval and update patterns is developed. This detailed specification of activities is often more difficult to develop than the data model. Ryan and Carlis (RYAN84) present a procedure that analyzes a data model and automatically suggests representative query sets for the database. Rather than having the users specify their queries, these representative sets of queries can be validated or perhaps modified by the users. Having concrete sets of queries for an application reduces the burden for the user and increases the probability that a good characterization of the retrieval patterns will be obtained.

4. IMPLEMENTATION

Implementation is the production of computer code to accomplish the

database design and meet the user requirements on some target computer system (including DBMS software). Productivity in this area has been less than spectacular. Fourth generation languages based on relational database management systems promise order of magnitude increases in productivity (CODD82) for simple queries, but they offer only slight improvement over third generation languages for transaction processing, a critical area for database systems.

An approach to increased productivity suggested in the Software Engineering literature is the utilization of reusable software components -- generic procedures that define a collection of high level building blocks from which a particular system can be configured. In fact, database management systems may be considered to be a collection of such components at a primitive level.

Ridjanovic is developing a taxonomy of generic business database functions, and a corresponding library of generic procedures. This effort will focus on maintaining database integrity while supporting transaction processing activities (including input validation, exception handling, and update propagation). Three applications from different business areas will be developed to test and improve the library of generic procedures. Results from this effort will be analyzed with respect to reusability among different business areas. Finally, a controlled experiment will evaluate the impact of these generic procedures on the productivity of system developers.

5. SUMMARY AND FUTURE DIRECTIONS

Research efforts are underway to develop and evaluate tools to support three database design processes: Abstraction - the determination and documentation of database requirements, Optimization - the design of effective and efficient physical database structures, and Implementation - the production of code to implement the designs.

Future research efforts will continue in these directions with increased emphasis on the human engineering of the resultant tools. Prietula (PRIE84) has laid the groundwork for new research in this area by analyzing and developing a model of human reasoning processes in physical database design. There are aspects of database design which, at this time, are best left to a human designer. Thus, the tools must be able to both support and adapt to the ways "people think" about those aspects of database design.

6. REFERENCES

- BROD84 Brodie, M. L. and Ridjanovic, D., "On the Design and Specification of Database Transactions," in On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages, Springer-Verlag, 1984.
- CARL83a Carlis, J. V. and March, S. T., "A Computer Aided Database Design Methodology," Computer Performance, December 1983.
- CARL83b Carlis, J. V., Dickson, G. W., and March, S. T., "Physical Database Design: A DSS Approach," Information and Management, Vol. 6, No. 4, August 1983.
- CARL84 Carlis, J. V. and March, S. T., "A Descriptive Model of Physical Database Design Problems and Solutions," Proc. Data Engineering Conference, IEEE, Los Angeles, April 24-27, 1984.

- CODD82 Codd, E. F., "Relational Database: The Key to Productivity," CACM, Vol. 25, No. 2, February 1982.
- FLOR84a Flory, A., March, S. T., and Ridjanovic, D., "A Framework for the Development of the Information Resource," University of Minnesota Working Paper 85-01, August 1984.
- FLOR84b Flory, A. and March, S. T., "The Functional Dependency Model: A Unified Approach to Information System Development," Working Manuscript, University of Minnesota, May 1984.
- MARC82 March, S. T. and Carlis, J. V., "An Overview of Physical Database Design Research at the University of Minnesota," IEEE Database Engineering Newsletter, IEEE Computer Society, Vol. 5, No. 1, March 1982.
- MARC83a March, S. T., "Techniques for Structuring Database Records," Computing Surveys, Vol. 15, No. 1, March 1983.
- MARC83b March, S. T., "A Mathematical Programming Approach to the Selection of Access Paths for Large Multiuser Databases," Decision Science, Vol. 14, No. 4, Fall 1983.
- MARC84 March, S. T. and Scudder, G. D., "On the Selection of Efficient Record Segmentations and Backup Strategies for Large Shared Databases," ACM Transactions on Database Systems, Vol. 9, No. 3, September 1984.
- MEND85 Mendu, S., An Investigation into the Unit of Data Allocation for the Design of Distributed Databases, Ph.D. Dissertation, University of Minnesota, expected June 1985.
- PALV84 Palvia, P., An Analytical Investigation into Record Structuring and Physical Database Design of Generalized Logical Data Structures, unpublished Ph.D. Dissertation, University of Minnesota, April 1984.
- PRIE84 Prietula, M. J., An Investigation of Reasoning Methods Used in Physical Database Design Problem Solving, unpublished Ph.D. Dissertation, University of Minnesota, expected December 1984.
- RYAN84 Ryan, K. L. and Carlis, J. V., "Automatic Generation of Representative Query Sets," Proceedings 1984 Trends and Applications Conference, Making Database Work, National Bureau of Standards, Gaithersburg, MD, May 23-24, 1984.
- SENK75 Senko, M. E., "Specification of Stored Data Structures and Desired Output results in DIAM II With FORAL," Proceedings Very Large Data Base Conference, Framingham, MA, 1975, pp. 557-587.
- TEIC77 Teichroew, D. and Hershey, E. A., "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems," IEEE Transactions on Software Engineering, Vol. 3, No. 1, January 1977.
- WETH83 Wetherbe, J. C. and Davis, G. B., "Developing a Long Range Information Architecture," Proceedings National Computer Conference, Anaheim, 1983.

RESEARCH ON FORM DRIVEN DATABASE DESIGN AND GLOBAL VIEW DESIGN

Michael V. Mannino † and Joobin Choobineh ‡

† Database Systems Research and Development Center
Computer & Information Sciences Department
University of Florida
Gainesville, FL 32611
CSnet: mannino@ufl

‡ Department of Management Information Systems
University of Arizona
Tucson, AZ 85721

ABSTRACT

We are currently investigating two non-traditional approaches to database design. Form-driven database design is the derivation of a database schema from a collection of documents or forms. Global view design is the problem of defining the objects and mappings in a global view which is a view of more than one database. In this short paper, we discuss our results and on-going work on both of these topics.

1. FORM DRIVEN DATABASE DESIGN

We define a form as any standardized set of data variables. A form schema is the definition of the form fields and their constraints. A form template is a particular representation of a form type on a given medium. A form schema is medium independent whereas a form template is medium dependent. The same form schema may have form templates on a video screen, paper, and even voice. This definition of a form is quite broad and fits well into the current research in the area of office automation [TSIC82].

Forms are attractive as a primary input to the database design process for several reasons. First, forms are formal models and do not have the ambiguities of natural language requirements. Second, a form model is a data model. By studying and analyzing a form and its relationships to other forms, many data dependencies and mappings can be discovered. Third, many forms contain instructions about filling out the form which provide additional information about an organization's data. Fourth, forms are easy to understand and are widely used in most organizations. Users should be able to participate in the database design process through the definition of their forms.

Our research approach involves the definition of a form model and the investigation of intra- and inter-form analysis [CHOO84]. Our form model permits a hierarchical grouping of form fields. The grouping is indicated by form field embedding and cardinalities which indicate the number of occurrences of form

fields. The cardinality of a form field can be 1 (1 value per form), set (collection of unique values), or bag (collection of values with duplicates). We also specify the domain and origin (input, display, computed) of each form field. In Figure 1, all form fields have a cardinality of one except for PRODUCT-NO which is set-valued, LOCATION which is set-valued within PRODUCT-NO, BIN-NO which is set-valued within LOCATION, QUANTITY which is bag-valued within LOCATION, and WEIGHT which is bag-valued within LOCATION. Alternately, WEIGHT and QUANTITY could be considered as single-valued within BIN-NO.

Intra-form analysis converts a form schema into an extended Entity-Relationship diagram. We first partition form fields according to their cardinality and embedding level. All top level single-valued fields are placed in one group which is named with the form name. Each top-level, set-valued form field is placed in a separate group along with any embedded form fields. We then use a rule-driven dialogue to establish entity types and relationships, cardinalities, and some functional dependencies. In the Shipment form, we have the following form field groups:

SHIPMENT group: DATE OF AUTHORIZATION, INVOICE NUMBER,
NAME, ADDRESS, SALES ORDER NO., TOTAL, AUTHORIZED BY,
CARRIER

SHIPMENT-LINE group: PRODUCT-NO, LOCATION, BIN-NO, QUANTITY,
WEIGHT

Figure 2 shows a possible Entity-Relationship diagram that represents the form. The top-level, single-valued fields are split into four entity types: CUSTOMER, SHIPMENT, CLERK, and CARRIER. This splitting can be accomplished by collecting functional and multi-valued dependencies or by asking for the the entity types. The rest of the Entity-Relationship diagram is more difficult to derive. It is possible to consider the SHIPMENT-LINE group as a single entity type, but the designer would likely indicate that it should be decomposed into PRODUCT, WAREHOUSE, BIN, and SHIPMENT-LINE. This is because the designer can recognize that PRODUCT-NO, LOCATION, and BIN-NO represent entity types.

The use of pure intra-form analysis usually requires redundant user interaction. To reduce the user interaction, knowledge gained from analyzing previous forms is utilized in analyzing the current form. The user interaction may be significantly reduced if previously analyzed forms contain similar information. For example, if NAME and ADDRESS are grouped together as the result of the analysis of another form, they can be immediately split into a separate group without user interaction.

Inter-form analysis involves form flow and incremental design. The form flow shows the relationship between form fields across all forms. The form flow analysis can resolve many synonym and homonym differences and ensure that each form field has an origin (i.e., is an input field on some form) and a destination (i.e., is a display field on some form).

The conceptual schema can be incrementally designed by integrating form schemas. Forms can be analyzed in groups where each form in a group is closely-related to the others. Two forms are closely related if they have many form fields in common. The incremental analysis of forms allows the identification of conflicts in requirements, attributes of a form field group on one form which are entity types on other forms, additional attributes for entity types and

SHIPMENT AUTHORIZATION

DATE OF AUTHORIZATION	12-12-83	INVOICE NO.	580
SHIP TO:	NAME ABC Stores ADDRESS Any Street New York, N.Y. 20000	SALES ORDER NO.	153

WAREHOUSE					
PRODUCT-NO.	LOCATION	BIN-NO.	QUANTITY	WEIGHT	
P10	TUC	2	18	180	
		3	8	80	
P20	TUC	8	10	50	
		PHE	4	10	50
			8	5	25
TOTAL -----				385	

AUTHORIZED BY	anyclerk	CARRIER	ACME
---------------	----------	---------	------

Figure 1: Shipment Authorization Form

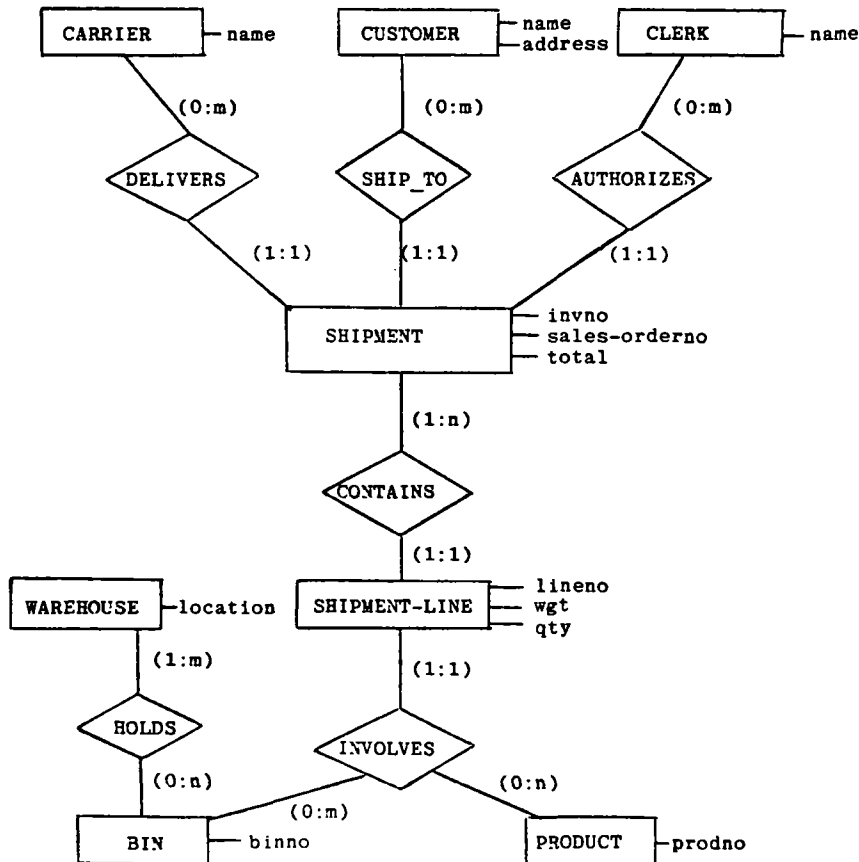


Figure 2: One Possible ER Diagram for the Shipment Authorization Form

relationships, and entity generalizations.

After the conceptual schema design is complete, the mappings to materialize the forms are defined. If a form schema includes updatable fields, the form's underlying mapping must either produce an updatable view or the designer resolves the ambiguities. This is the general problem of view updates where the database system restricts the type of view mappings and possibly permits the designer to resolve ambiguities where the mapping is outside of this range.

Most of this work is currently in progress. Our form model is complete and we are now investigating intra-form analysis. A detailed investigation of inter-form analysis will follow.

2. GLOBAL VIEW DESIGN

In recent years, there has been an increasing interest in uniform access to heterogeneous, distributed databases, while preserving investment in existing software and hardware [GLIG84, LAND82]. An important element of heterogeneous database technology is the global view which is an integrated view of multiple databases. A global view consists of a set of global schema objects and a mapping from the global objects to the underlying local objects. The design of large-scale global views can be difficult because of the number of global objects, the number of global to local mappings, and the complexity of the mappings.

To address the difficulties of global view design, we have developed a methodology [MANN84a,b] which consists of four steps. In the first step, the local schemas are translated into equivalent schemas in a common or unifying data model. This resolves differences merely due to the different local data models. We use the Generalized Entity Manipulator (GEM) [ZANI83] as a common or unifying data model. GEM supports generalization, entity-valued attributes, set valued attributes, and surrogate primary keys.

In the second step, assertions are defined and analyzed about entity types and attributes in the local unifying schemas. The entity types are partitioned into identification-independent and identification-dependent groups. Within each group, the designer identifies the entity types that can be generalized. Then, for each generalization group, the designer makes assertions about the semantic equivalence and range of meaning of attributes. Assertions can be made about individual attributes, groups of attributes, and functions of an attribute. We have devised an algorithm to check the consistency and completeness of the entity and attribute assertions. The major part of the algorithm is to transitively derive omitted attribute assertions including the range of meaning.

In the third step, entity types in each generalization group are merged. We have devised an interactive merging algorithm to assist the designer with the merging process. The merging algorithm connects two entity families at a time where an entity family is a set of entity types related by generalization. The designer indicates where to connect the two entity families and the merging algorithm removes unneeded parts of the local entity families, connects the local entity families, and assigns attributes to the global entity family as indicated by attribute assertions.

In the fourth step, the format and mappings of the global attributes are defined by the designer using a collection of conversion operators. Our conversion operators include data type converters, arithmetic formulas, concatenation and substring, table lookups (actually GEM queries), and database procedures.

These operators can handle differences in names, data types, lengths, scales, groupings, and codes. Combinations of differences can be handled by nesting conversion operators. We are currently studying rules to aid a designer in defining global attribute formats and mappings. These rules may possibly be incorporated into an expert system.

The major thrust of our on-going work is to implement the methodology. In one project, we are developing a graphical schema merging editor. The editor will graphically display the entity families of interest and then connect them as indicated by the designer. The designer can see alternative merging possibilities by placing the result of a merge operation in a window. Our initial implementation will be to merge a pair of entity families. Later, we plan to extend this to N schemas at a time and to incrementally merge two schemas at a time.

In a second project, we are developing an expert system for converting from an INGRES schema to a Generalized Entity Manipulator (GEM) [ZANI83] schema. The conversion can be trivial because GEM is compatible with the data definition language of INGRES. However, our effort focuses on the hidden semantics in the INGRES schema. For example, a group of three relations may represent a set of entity types related by generalization. We are investigating the conversion of the following hidden semantics: 1) generalization, 2) physical database design options that affect the conceptual schema such as vertical partitions and denormalization, and 3) referential integrity. We hope to extend this work to different relational database systems and to different classes of database systems (e.g., CODASYL, IMS).

In a third project, we have developed a global view definition language [KARL84]. The global view definition language can be used instead of the design methodology or in conjunction with the methodology. For example, the designer could initially use the methodology to design a global view and then make changes to the generated view definition statements instead of using the methodology again. Implementation of the language is planned.

Future work is planned in the areas of updatable global views and complex objects. We need to extend the previous work on updatable views to global views. We foresee changes to steps three and four of our methodology to support the design of updatable global views. Complex objects are an important data modelling feature in design data management such as CAD/CAM. We foresee extending GEM and our methodology to support complex objects. We would like to support the design of global views with complex objects materialized from more than one database.

ACKNOWLEDGEMENTS

We thank Dr. Stanley Su and Dr. David Reiner for useful suggestions on earlier versions of this paper.

REFERENCES

- CHOO84 Choobineh, J. *Form Driven Database Design*, Ph.D. Dissertation, Dept. of Management Information Systems, University of Arizona, 1984. (in preparation).
- GLIG84 Gligor, V. and Luckenbaugh, G. "Interconnecting Heterogeneous Database Management Systems." *IEEE Computer* 17, 1 (Jan. 1984).

33-43.

- KARL84** Karle, C. *A Global View Definition Language*, Master's Thesis, Computer and Information Sciences Dept., University of Florida, August 1984.
- LAND82** Landers, T. and Rosenthal, R. "An Overview of Multibase," in *Proc. 2nd Intl. Symposium on Distributed Databases*, H.J. Schneider (ed.), Sept. 1982, Berlin, F.R.G., pp. 153-184.
- MANN84a** Mannino, M. and Effelsberg, W. "Matching Techniques in Global Schema Design," in *Proc. Intl. Conf. Data Engineering (COMPDEC)*, IEEE, Los Angeles, CA, April 1984, pp. 418-425.
- MANN84b** Mannino, M. and Effelsberg, W. "A Methodology for Global Schema Design," UF-CIS Tech. Report TR-84-1, Computer and Information Sciences Dept., University of Florida, Sept. 1984, submitted for publication.
- TSIC82** Tsichritzis, D. "Form Management," *Communications of the ACM* 25, 7 (July 1982), 453-478.
- ZANI83** Zaniolo, C. "The Database Language GEM," in *Proc. ACM SIGMOD Conf.*, San Jose, CA, pp. 207-218.

A Prototyping Approach to Database Applications Development

Antonio Albano and Renzo Orsini

Dipartimento di Informatica, Università di Pisa
Corso Italia, 40 - 56100 Pisa, Italy

Abstract

Several research projects are presently in progress to implement an integrated environment of tools to support the database design process. The conventional approach is based on the principle that the design process produces a non executable specification of the application. An alternative approach, called the operational approach, is presented, based on recent trends in software engineering: it assumes that the conceptual design should be made with an high level, executable language to implement rapidly a prototype of an application. The approach will be briefly motivated and a description will be given of the database designer's workbench Dialogo that is currently being implemented.

1. INTRODUCTION

Several research projects are presently in progress to implement an integrated environment of tools to support the database design process [ALBA85a], [ATZE82], [CERI83], [REIN84], [TEOR82]. There is a general consensus on the proper phases of the process and on the decisions to be made during those phases. This conventional approach to database design is pictured in Figure 1. The main assumptions upon which almost all the proposals are based are:

- The goal of the process is to design a reasonable structure of the database that will allow an efficient implementation of the applications during the implementation stage, which however is not considered as part of the design process, but it is seen as a separate stage. In other words, the proposed tools support the design process, but not the database life cycle.
- The attention of the database workbench designers is mainly focused on tools for data analysis, view integration, schema mappings and physical design. Other important issues that should be considered in the design stage are often underestimated: constraint specification and verification, transaction specification and dialog specification. These issues have been considered by many authors, but for the time being the tool-builders are not offering specific solutions for them.
- Conceptual design is considered a fundamental step of the design process, and it must be independent of the DBMS used in the implementation stage. A great deal of attention is given to a graphical notation to visualize the information content of the database, but the proposed tools usually are not based on an executable conceptual language.

This paradigm has a strong analogy with the conventional "software life cycle", which has been followed in the field of software engineering, and it presents the same drawbacks that have been pointed out by software designers [BALZ83]:

- The tools supporting the design process are of limited interest once the applications have been implemented. In particular they can not be used to maintain the released information system, which has been implemented in a target language not contemplated during the design process.
- Users have no means of ensuring that the specified system matches their intent before a first release of the implementation is operational. This compounds the maintenance problem.

This work was supported in part by the Consiglio Nazionale delle Ricerche, Progetto Finalizzato Informatica, Obiettivo DATAID, and in part by the Ministero della Pubblica Istruzione.

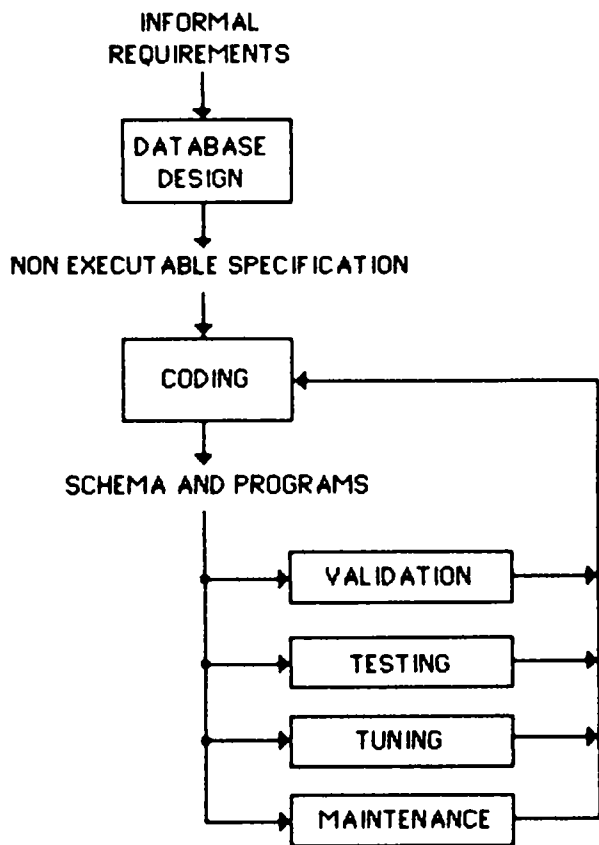


Fig.1 Conventional Paradigm

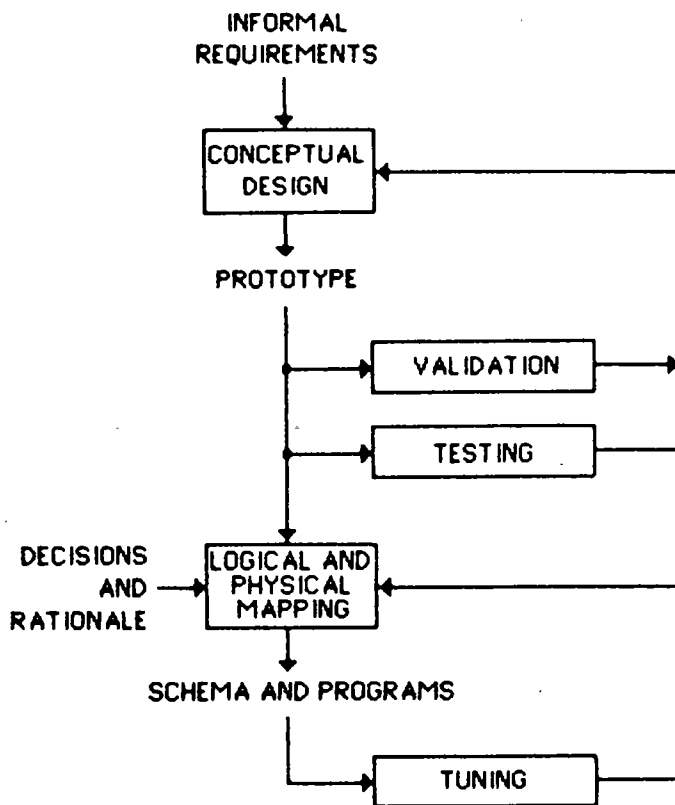


Fig. 2 Operational Paradigm

To overcome these difficulties a new approach to software development is gaining popularity among practitioners and academicians in the software engineering and information systems fields: an operational, or prototyping, approach to software development (Fig. 2) [ALAY84], [BUDD84], [ZAVE84].

The basic assumption of this approach is that, during the design phase, computer specialists give an implementation independent, executable (operational) specification of the system to be implemented. The operational specification is a prototype, an early version of the system that exhibits the essential features of the future operational system, except for the resource requirements and performances.

The first advantage of this approach is that users can experiment with the prototype to determine whether or not it matches their requirements. Such prototypes are expected to improve the users' perception of their needs, and so the resulting systems are more responsive to those needs. A second advantage is that computerized tools can be designed to support the transformation of the prototype into an efficient, system dependent implementation. A third advantage is that the main result of the design phase, the prototype, is not discarded once the system is operational, but will be used to perform maintenance of the applications.

In our opinion, the operational approach is also relevant to the database field. The natural stage to introduce prototyping capabilities is at the conceptual level, by providing an executable, high level language to specify data, constraints, operations, and dialogs [ALBA83a], [OBR183]. Investigations are required to define which features a language should have for these purposes and how to embed this tool in a system to support the database life cycle. We are presently engaged in a project, called the Galileo Project, to design and implement a database designer's workbench based on this approach, where the language Galileo is used for prototyping [ALBA85b]. We do not claim that Galileo is the best language for this scope, because it is still a research topic to isolate the right language features. However, the language is sufficiently expressive and DBMS independent to be considered as a candidate

to experiment. Another reason why we are using Galileo is that the language is almost operational and it can be used on real applications to test its features and to investigate the architecture of the database designer's workbench, called Dialogo.

The next section presents an overview of the Galileo Project and the language Galileo. Section 3 briefly describes the architecture of Dialogo. Section 4 contains the summary and comments on the current status of the project. A more detailed description of the implemented tools appears in [ALBA85c].

2. THE Galileo Project

The goal of the project is to design and implement an interactive integrated system to design and prototype database applications. The language supported by the system is Galileo, specifically designed to deal with the different aspects of complex database applications. The system will support also a non-executable version of the language, Galileo/R, to document user requirements. Both the languages are intended to be used by experts, when detailed descriptions must be given. However, for high level descriptions of the database structure, graphical notations and interactive interfaces are provided to communicate with casual users. The underlying assumption is that the system should allow users to give specifications at different levels of detail, using a single set of abstraction mechanisms; the more the user becomes familiar with the capabilities of the tools and with the problem to solve, the more he should be able to give a detailed description of the specifications. In other words, requirement specification and conceptual design should not appear as two isolated worlds, but only two different levels of description that can be transformed into each other once the user becomes more expert. Of course, the less detailed the specification, the less possibilities there are to verify it.

2.1 The Galileo Language

We assume that the goal of the conceptual design is the implementation of a working prototype that exhibits the essential features of the final product. Therefore a great deal of attention has been given to the design of a high level programming language that supports adequate abstraction mechanisms for database applications. The language should also allow the definition and test of the prototype in a small fraction of the time required to make the same prototype with languages available in commercial DBMSs. Here is an overview of the language.

Galileo is an expression language: each construct is applied to values to return a value.

Galileo is an interactive language: the system repeatedly prompts for inputs and reports the results of computations; this interaction is said to happen at the top level of evaluation. At the top level one can evaluate expressions or perform declarations. This feature, which is not present in other conceptual languages, allows the interactive use of Galileo without a separate query language.

Galileo is higher order, in that functions are denotable and expressible values of the language. Therefore, a function can be embedded in data structures, for instance to model derived properties of entities, passed as parameter and returned as value.

Galileo is a safe language. Every denotable value of the language possesses a type. Besides predefined types, type constructors exist to define new types, from predefined or previously defined types. They are: tuple, sequence, discriminated union, array, function and abstract types. In defining abstract types, it is possible to restrict the set of possible values with assertions and to inherit the primitive operations of the representation type. In general, any expression has a type that can be statically determined, so that every type violation can be detected by textual inspection (static type checking). Although any statically detectable error could also be detected at run-time, the language has been designed to be statically type checkable for the following basic reasons: firstly, programs can be safely executed disregarding any information about types; secondly, the language offers considerable benefits in testing and debugging applications, since the type-checker detects a large class of common programming errors without the need of executing programs, while errors at run-time could be detected only by providing test data that cause the error to be raised. In fact, static type checking is considered an example of consistency checking extremely useful to detect frequent semantic errors. For database applications the above benefits are certainly valuable, but static type checking does not prevent dynamic testing for assertion enforcement. However, the type-checker is still useful to provide information to the translator to produce specialized testing code.

Galileo has type inheritance. If a type T is a subtype of a type T' , then a value of T can be used as argument of any operation defined for values of T' , but not vice versa because the subtype relation is a

partial order. Type hierarchy is important to incorporate the generalization abstraction mechanism of semantic data models into a strongly typed programming language [ALBA83b].

Galileo has control mechanism for failures and their handling.

Galileo supports the abstraction mechanisms of semantic data models: classification, aggregation and generalization. Classes are the mechanism to represent a database by means of sequences of modifiable, interrelated objects, which are the computer representation of certain facts of entities of the world that is being modeled. Class elements possess an abstract type and are the only values which can be destroyed. Predefined assertions on classes are provided and, if not otherwise specified, the operators for including or eliminating elements of a class are automatically defined.

Galileo supports modularization as another abstraction mechanism to partition data and operations into interrelated modules. Therefore, a complex schema can be structured into smaller units. For instance, a unit may model a user view or a description of the schema produced by a stepwise refinement methodology [ALBA83c].

Galileo has a process mechanism to model user activities and dialogs with the system as long term transactions which can proceed in parallel and interact by message passing.

Galileo has a form oriented interface to input or to display database objects.

3. THE Dialogo SYSTEM.

Dialogo is an environment for the development of complex, interactive information systems. While its current scope is towards the production of prototypes, in the future it will be extended to deal efficiently with large quantities of data in secondary memory, and so to develop applications programmed in Galileo on graphics workstations. Besides tools designed to support prototyping, other tools will exist to collect Galileo/R definitions, to transform them in Galileo, and to translate the Galileo design in a DBMS language.

To control the designer's activities, the system keeps track of the alternatives investigated and of the "derived" relationships between the products of the various stages of the development, such as designer's produced schemas or results of design analysis tools. A "project tree" is used to give an overall view of the project, to control different versions, to inspect the ancestors of a design, and so on. This idea is borrowed from the DDEW system, being implemented at CCA [REIN84].

Dialogo is intended to be used mainly by expert designers, although a simple graphical interface will be provided for end users, to let them interact with the system both to verify the designer's work, and to check the prototype's behaviour. However, graphics is also relevant for designers, since there is an increasingly and widespread conviction about the importance of user-friendly interfaces in all the software systems used by humans. For these reasons, the system is designed to offer a uniform graphical interface to all its users, and it is implemented in a graphics workstation with a high-resolution screen and a "mouse" as a pointing device. This approach is not new, and a notable example in the database design area is the DDEW system [REIN84]. A preliminary version of Dialogo was described in [ALBA83a].

3.1 System architecture.

The system's kernel is the "project database", shared by all the system's components. This database is in effect a collection of databases. Its overall structure is a tree, representing the project tree. The nodes of such a tree are the meta databases of the various schemas produced automatically or manually during the design phasis, in all their different versions, and the databases created during the execution of the prototypes.

Another fundamental component of the system is the graphical editor. In fact, it is the only component seen by the user, in the sense that it provides a double function: an homogeneous way to interact with the project database, and mechanism to activate the other tools of the system. From these points of view, some of its functions are similar to those performed by the Macintosh™ operating system.

Through a graphical representation of the project tree and of the databases, the user can inspect and modify the project database. The basic operation is the "opening" of an icon representing some kind of data, which pops-up a new window on the screen, showing a portion of the interested data, which can be of graphical or textual nature or a mix of these. Due to the highly structured kind of data in the database, this selection will continue typically for several levels. When the user wants to change something in the actual window, he must not change the environment: simply he will use the

appropriate (graphical or textual) modifying functions of the editor. Windows can be scrolled in any directions, with scroll bars, and resized and deleted with corresponding tabs at the corners of the window. The command language is provided by menu mechanisms. A pull-down menu in a fixed area of the screen will contain global commands always accessible. Pop-up menus come out in response to particular user's actions, typically to the push of a mouse's button, and are appropriate to the context in which the user is currently working. This kind of command language is used both to invoke and to dialogate with the tools of the system.

The system provides also tools to input schemas, to analyse and execute prototypes, and to map from conceptual to logical designs. All these tools, invoked by the user via the graphical editor, access and manipulate the project database through a set of Galileo interfaces (abstract types with appropriate operators). They will be listed in the next section. Finally, to develop Galileo programs before the graphical interface is operational, there are tools to input a schema with traditional textual editors.

3.2 Functional components.

A first set of tools is standard for any software development system based on a programming language: a) semantic analysers and type-checkers for the control of data and programs definitions; b) interpreters and compilers for the execution of Galileo prototypes; c) interactive monitors and debuggers, to trace a program execution and to inspect graphically its data, with the possibility of intervening during the execution.

Another set of tools is addressed more directly to the database design process: a) report generators, both batch-like, to produce summary and cross-reference tables, and interactive, to show graphically data relationships, connections between data and programs, etc; b) tools for the specification and verification of user requirements in Galileo/R, with type checking facilities, although limited by the possibility of inserting natural language specifications intermixed with formal specifications; c) report generators for user requirements and their relationships with the prototype; d) tools for the translation of Galileo definitions into the language of a specific DBMS, for the logical and physical design phase.

4. CONCLUSIONS

An overview of the Galileo Project at University of Pisa has been presented. The project is finalized to the implementation of an experimental database designer's workbench for graphics workstations, based upon an operational approach. The implementation of the workbench is in progress on a Sun Workstation and on a Vax 11/780, both running UNIX™ Berkely 4.2. The implementation is carried out in Pascal in cooperation with Systems & Management S.p.A., and our efforts are presently on the implementation of Galileo. The following tools are in the testing stage: a) an interactive compiler for Galileo. In the present implementation, the management of persistent data has not yet been included, but two functions, save and restore, are provided to save and restore the current state of a working session on a specified file; b) a syntax driven editor of Galileo programs. The editor and the compiler are presently two independent tools; c) an integrated syntax driven editor and interpreter, implemented at Systems & Management S.p.A.. The implementation reflects the kind of architecture we have in mind for the final system, but it does not make use of graphics [CAPA85].

The following tools are not yet operational: a) a graphical editor for requirement specification and analysis in Galileo/R; b) the metadatabase to collect information on projects in progress; c) a form oriented interface to input and display objects of a database described in Galileo.

Since the graphics workstation has been acquired only recently, the graphical interface has not yet been designed, but this aspect will be one of our main concerns for the near future, together with the integration of editor, interpreter, compiler, and project database. Next, the inclusion of mapping tools, from the prototype in Galileo to specific DBMS languages, will be considered.

SELECTED BIBLIOGRAPHY

- [ALAV84] Alavi, M., "An Assesment of the Prototyping Approach to Information Systems Development", *Communications of the ACM* 27, 6, 556-563, 1981.
- [ALBA83a] Albano, A., and R. Orsini, "Dialogo: An Interactive Environment for Conceptual Design in Galileo", in *Methodology and Tools for Database Design*, S. Ceri (ed.), North Holland, Amsterdam, 229-253, 1983.
- [ALBA83b] Albano, A., "Type Hierarchies and Semantic Data Models", *ACM Sigplan '83: Symposium on Programming Language Issues in Software Systems*, San Francisco, 178-186, 1983.
- [ALBA83c] Albano, A., M. Capaccioli, M.E. Occhiuto, and R. Orsini, "A Modularization Mechanism for Conceptual Modeling", *Proc. 9th Intl. Conf. on VLDB*, Florence, Italy, 232-240, 1983.
- [ALBA85] Albano A., V. De Antonellis, and A. Di Leva (eds.), *Computer Aided Database Design*, North-Holland, Amsterdam, 1985 (to appear).
- [ALBA85b] Albano, A., L. Cardelli, and R. Orsini, "Galileo: A Strongly Typed, Interactive Conceptual Language", *ACM TODS*, 1985 (to appear).
- [ALBA85c] Albano, A., and R. Orsini, "A Software Engineering Approach to Database Design: The Galileo Project", in *Computer Aided Database Design*, Albano A., V. De Antonellis, and A. Di Leva (eds.), North-Holland, Amsterdam, 1985 (to appear).
- [ATZE82] Atzeni, P., C. Batini, V. De Antonellis, M. Lenzerini, F. Villanelli, and B. Zonta, "A Computer Aided Tool for Conceptual Database Design", in *Automated Tools for Information System Design*, H.J. Schneider and A. Wasserman (eds.), North Holland, 85-106, 1982.
- [BALZ83] Balzer, R., T.E. Cheatham, and C. Green, "Software Technology in the 1990's: Using a New Paradigm", *Computer*, 39-45, 1983
- [BUDD84] Budde, R., K. Kuhlenskamp, L. Mathiassen, and H. Zullighoven (eds), *Approaches to Prototyping*, Springer-Verlag, Berlin, 1984.
- [CAPA85] Capaccioli, M., and M.E. Occhiuto, "A Workbench for Conceptual Design in Galileo", in *Computer Aided Database Design*, Albano A., V. De Antonellis, and A. Di Leva (eds.), North-Holland, Amsterdam, 1985 (to appear).
- [CER183] Ceri, S. (ed.), *Methodology and Tools for Database Design*, North-Holland, Amsterdam, 1983.
- [REIN84] Reiner, D., M. Brodie, G. Brown, M. Chilenskas, M. Friedell, D. Kramlich, J. Lehman, and R. Rosenthal, "A Database Design and Evaluation Workbench: Preliminary Report", *Intl. Conf. on Systems Development and Requirement Specification*, Gothenburg, Sweden, Aug. 28-30, 1984.
- [TEOR82] Teorey, T.J., and R. Cobb, "Functional Specification for a Database Design and Evaluation Workbench", Working Paper 82 DE 1.15, Information Systems Research Group, Graduate School of Business Administration, University of Michigan, 1982.
- [ZAVE84] Zave, P., "The Operational Versus the Conventional Approach to Software Development", *Communications of the ACM* 27, 2, 104-118, 1984.

A CAUSAL APPROACH TO DYNAMICS MODELING

V. De Antonellis(*), B. Zonta(**)

(*) Istituto di Cibernetica, Università degli Studi di Milano
Via Viotti 5 - 20133 Milano (Italy)

(**) Consiglio Nazionale delle Ricerche - Milano (Italy)

Abstract

This paper summarizes the results of a research devoted to the specification of dynamic properties of database applications. The concepts of event and procedure have been introduced. Petri Nets have been adopted to represent formally them, and methodological steps for their modeling in the design process of database applications have been defined.

1. Introduction

In the design of a database application, four phases /CERI83,LUMV78, NAVA80/ are usually distinguished: requirements collection and analysis, in which users information needs are identified; conceptual design (view design and integration), in which information needs are expressed in a formal language; logical design, in which the conceptual schema is translated into a DBMS-processable schema; and physical design, in which physical optimizations are performed.

In particular, the aim of conceptual design is to obtain a formal, integrated, and DBMS independent description of the relevant concepts of the object system to be automated. Such a description has to involve both static and dynamic aspects. For the description of static aspects, several data models with related languages have been proposed in literature /ALBA85,CHEN76,ELMA79,HAMM81,.../. For the dynamic aspects, there are various proposals concerning modeling single operations (transactions) on data classes. Less attention has been given to the equally important problem of modeling application dynamics: sets of operations, and their precedence relationships (i.e., the organization activities).

The ISO report /ISOT81/ identifies three approaches to modeling dynamic aspects: state-oriented, command-oriented, and interaction-oriented approach. In a state-oriented approach, the after-states which may be reached from a given before-state are described by means of rules. The rules can be expressed by sentences from the predicate logic. There is no interest in the commands performed to transform one state into the next one. In a command-oriented approach, on the contrary, the main interest is in describing allowed command sequences, which cause permissible actions which change a permissible state into another permissible state. Rules are expressed in conditional sequences of commands. Both state-oriented and command-oriented approaches formulate which changes are permissible and concentrate on the description of rules and constraints. Some sources refer to them using the terms 'definitional' and 'procedural' approaches respectively. Finally, an interaction-oriented approach describes the interaction between the application and the environment, and the causality of the changes in the application. The concepts used are: operations, events, and synchronization.

Our research /BERT83,DEAN80,DEAN81/ addresses the specification of dynamic properties of database applications using an interaction-oriented approach. We analyze the behavior of the object system (i.e., the activities to be automated) and formalize it into procedures against the database using the Petri Net formalism /PETR80/. Other similar approaches /BARR80,ROLL83,TARD80/, differ from ours in various aspects. In /BARR80/, nets are used to express only the control flow of procedures, with knowledge about them expressed in production rules. In our approach, nets express also knowledge, as a consequence of the adopted methodology, in which the description of the knowledge acquired on events is constitutive of the net building itself, from the first step to the last step. Unlike /ROLL83,TARD80/, in which Petri Nets constitute only a starting point for peculiar modeling formalisms, our approach exploits the original Condition-Event interpretation of Petri Nets with its syntactic and semantic capabilities, and the related simulation rules.

2. Relevance of Dynamics Modeling

Application dynamics modeling results in a clear and unambiguous representation of the way in which the activities to be automated must be executed. Specifically, operations and causal dependencies/independencies between them are identified, and the related conditions are described. This representation, if expressed in a formal language, highlights the design choices, allows the discovery of inadequacies and inconsistencies, and suggests alternatives. In the conceptual design, it allows to check the completeness and consistency of the data and operations schemata (i.e. to test whether all the operations are defined in the operations schema, and all the data that are needed to execute the operations are adequately represented in the data schema). In the logical/physical design, it provides guidelines in choosing optimization parameters. Furthermore, such a formal representation, approved by experts and designers, can be made automatically executable. In this way, operations are executed according to their causality relationships, keeping track of the execution 'iter'.

In our approach, the activities are formalized into procedures by means of Petri Nets which have a well-defined syntax and a sound basic interpretation. Petri Nets have been proved easy to comprehend and use in the phase of conceptual design. Furthermore, their capability to represent causal dependencies/independencies by means of structures of 'sequence', 'conflict' (mutual exclusion) and 'concurrency' (parallelism), can be exploited in the phases of logical and physical design (i.e., when optimization choices on data structures and operations are made /BALB84/).

3. Causal Model of Procedures

A procedure consists of a set of operations and precedence relationships between them. Precedence relationships are expressed by means of conditions which hold before (or after) the execution of an operation. The causal model is based on the notion of event. An event is a change of conditions from those which hold before (pre-conditions) the execution of an operation to those which hold after (post-conditions) such an execution. An event is described by means of its pre-conditions and the operation which makes the change (post-conditions are left implicit in the semantics of the respective operation). A procedure is a 'texture' of events which expresses their mutual dependencies/independencies.

To illustrate our procedure modeling method, let us consider as an example the dispatch of an order by a customer and the order handling by the involved enterprise. We can distinguish two activities. That of the customer who sends the order, and waits for the answer from the enterprise; then, in the positive case (the required items are available) he receives the items and the bill, and pays the bill. That of the enterprise, which receives the order and checks item availability. If the items are available, the enterprise prepares the shipment together with the bill, and sends everything to the customer, waiting for payment; otherwise, the enterprise informs the customer, provides for replenishment, and put the order into the backorders.

For these activities, we can define two procedures, "customer" and "enterprise", each of them composed by a number of events. An example of event, for the enterprise, is: when an order arrives (preconditions), check-availability (operation which makes the change), availability has been checked (postconditions).

Features of procedures are the following. The execution of procedures can be carried on concurrently. See for example the procedures "customer" and "enterprise". The customer can prepare to receive the arriving items, and start an advertising campaign in order to promote sales, concurrently with the enterprise, which has to prepare the bill and the items. Note that also operations within the same procedure may be concurrent, as for example item preparation and bill preparation. Procedures can interact according to a synchronous or asynchronous communication protocol. In fact, it is possible that a procedure needs information from another procedure, or that a procedure starts another one. Communication is synchronous if a procedure sends a message to another and cannot proceed until it answers. For example, the procedure "customer" cannot proceed, after having sent the order, until the procedure "enterprise" answers. Asynchronous communication is also possible, that is, if a procedure sends a message to another one and proceeds without waiting for an answer. In our example, this happens when the enterprise sends the positive message to the customer. In fact, after sending the message, the enterprise begins preparing the items and the bill without waiting for a confirmation by the customer. Since not all the aspects of the activities can be automated, procedures communicate with users, who provide information which cannot be obtained automatically. Each procedure can have multiple instances, some or all of which may be in execution at any instant. The behavior of an instance depends on events, that is, on change of conditions as a consequence of the execution of operations.

4. Methodological Steps in Modeling Procedures by Means of Petri Nets

Petri Nets are the language adopted to represent procedures. The net interpretation adopted here is essentially the original C(ondition)-E(vent) interpretation, both in the structural and the dynamic aspects. Specifically, transitions (graphically, bars) represent operations; places (graphically, circles) represent conditions; an arc from an (input) place to a transition defines a pre-condition of an operation, while an arc from a transition to an (output) place defines a post-condition. A marker in a place (graphically, a dot in a circle) indicates the holding of the corresponding condition. If all the input places of a transition are marked, and if no output place of its is marked, that transition is enabled (i.e., it can fire); after firing, the markers disappear from the input places and a marker appears in each of the output places of the transition.

In our interpretation, the firing of a transition corresponds to an event occurrence (i.e., to a change of conditions because of the execution of an operation). Labels naming conditions and operations are related to the corresponding elements of each net. Figure 1 shows the net of the procedure "enterprise" (for simplicity, one assumes that the arrived orders refer to items present in the catalogue of the enterprise).

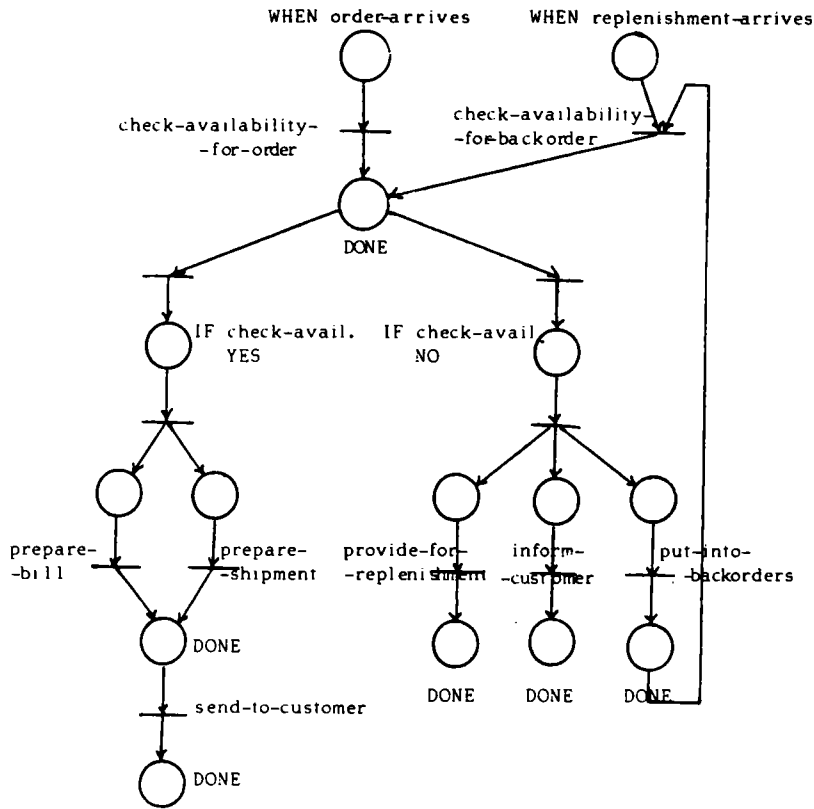


Fig. 1 Net of the procedure "enterprise"
 (unlabelled elements represent conflict or concurrency structures)

The methodological steps which allow the formal specification of procedures starting from the user requirements have been defined within the DATAID-1 methodology /CERI83,DEAN81/ and are here summarized.

Requirements collection and analysis:

User requirements regarding the object system to be automated are translated into a restricted natural language (synonyms are eliminated, pronouns are replaced by the corresponding nouns, restrictions are imposed on the use of articles, quantifiers, plurals, and so on). In particular, requirements descriptions for events are extracted. Since, at this level, it is already possible to recognize events which share the same preconditions either in conflict or in concurrency, such information is preserved into major units, called event-blocks. For each event or event-block, details and classifications of its conditions and operations are described in a form. Specifically, two types of conditions are distinguished according to whether they hold as a consequence of operations performed within or outside the same procedure. The

first are expressed by IF plus the name of the corresponding operation, the latter by WHEN plus a message or time reference. Operations are expressed by imperative verbs followed by complements. Multiple conditions/operations for one event are coordinated according to their mutual relationships by AND, OREX, OR, and by means of parentheses (forming hierarchies). The events are then collected and coded in a glossary, together with: references to activities to which they belong; involved data and operations; and, possibly, information about precedence relationships between events of the same activity.

Conceptual design:

By means of the association grammar /DEAN81/, each event-block is related to the corresponding Petri Net graph, whose elements are labelled with condition/operation expressions. Activity by activity, these graphs are composed in structures of sequence, conflict and concurrency according to rules which exploit the identity or the equivalence of labels. The composition results in a texture of events which shows the reciprocal causal dependencies/independencies. The final graph represents the way in which the activity to be automated must be executed, that is, the corresponding procedure. Finally, the defined procedures are coordinated through communication links which represent message exchanges between them.

Logical and physical design:

Formal procedure specifications are used in the logical and physical design phases. Causal dependencies between operations provide guidelines in choosing logical and physical parameters. In particular, a flow analysis of the Petri Nets representing procedures is performed in order to derive the operation activation frequencies /BALB84/.

5. Conclusions

The aim of our research is to model database procedures in terms of events and to execute the modeled procedures against a database /BERT83/. For modeling procedures at the conceptual level, we have developed a methodology and a computer-aided system, INCOD-E /ATZE82/. For automated operational support, we have designed an interactive system for the execution of procedures, IEHS /BERT84/. Our goal is to obtain an integrated work environment for procedure design and execution. We are currently investigating problems related to the interface between the two systems. Such an interface should allow the designer to generate a net description using the incremental facilities of INCOD-E, to make such a description executable, and to execute it using IEHS. The systems are currently being implemented in a UNIX environment with the INGRES relational database management system.

Bibliographical References

- /ALBA85/ Albano,A., Cardelli,L., Orsini,R., "Galileo: A strongly typed, interactive, conceptual language", to appear in ACM TODS, 1985.
- /ATZE82/ Atzeni,P., Batini,C., De Antonellis,V., Lenzerini,M., Villanelli, F., Zonta,B., "A computer-aided tool for conceptual database design", Proc. of the IFIP WG 8.1 Working Conf. on Automated Tools for Information Systems Design and Development, New Orleans, 1982.

- /BALB84/ Balbo,G., Demo,G.B., Di Leva,A., Giolito,P., "Dynamics analysis in database design", International Conf. on Data Engineering, Los Angeles, 1984.
- /BARR80/ Barron,J.L., "Dialogue organization and structure for interactive information systems", TR-CSRG-108, CSRG, University of Toronto, 1980.
- /BERT83/ Bertocchi,R., De Antonellis,V., Zonta,B., "Concepts and mechanisms for handling dynamics in database applications", 7th ICS, ACM European Regional Conf., Nurnberg, 1983.
- /BERT84/ Bertocchi,R., De Antonellis,V., Zhang,X.W., "An interactive events handling system, 1st Intern. Conf. on Computers and Applications, Peking, 1984.
- /BROD83/ Brodie,M., Silva,E., "Active and passive component modeling: ACM/PCM", in Information System Design Methodologies: a Comparative Review, North Holland, 1983.
- /CERI83/ Ceri,S.(ed), Methodology and tools for database design, North Holland, 1983.
- /CHEN76/ Chen,P.P., "The entity-relationship model: toward a unified view of data", ACM TODS, Vol. 1.1, 1976.
- /DEAN80/ De Antonellis,V., Degli Antoni,G., Mauri,G., Zonta,B., "Extending the entity relationship approach to take into account historical aspects of systems", in E-R Approach to Systems Analysis and Design, P. Chen ed., North Holland, 1980.
- /DEAN81/ De Antonellis,V., Zonta,B., "Modeling events in database applications design", Proc. Int. Conf. on Very Large Data Bases, Cannes, 1981.
- /ELMA79/ El-Masri,R., Wiederhold,G., "Data model integration using the structural model", Proc. ACM SIGMOD, 1979.
- /HAMM81/ Hammer,M., McLeod,D., "Database description with SDM: a semantic database model", ACM TODS, Vol. 6, 1981.
- /ISOT81/ ISO TC97/SC5/WG3, "Concepts and terminology for the conceptual schema", 1981.
- /LUMV78/ Lum,V.Y.,et al., 1978 New Orleans Database Design Workshop, IBM Report RJ2554 (33154).
- /NAVA80/ Navathe,S.B., et al., "Information modeling tools for database design", Data Base Directions, Fort Lauderdale, Florida, 1980.
- /PETR80/ Petri,C.A., "Introduction to general net theory", in Lecture Notes in Computer Sciences, n.84, Springer-Verlag, 1980.
- /ROLL83/ Rolland,C., "Database dynamics", DATA BASE, Vol. 14, n.3, 1983.
- /TARD80/ Tardieu,H., Nanci,D., Pascot,D., "Conception d'un système d'information", Les Editions de l'Organisation, Paris, 1980.

DESIGNING DATABASE UPDATES*

*Sharon Salveter
Douglas E. Stumberger*

Boston University Computer Science Department
111 Cummington Street Boston, MA 02215
(617) 353-8927

ABSTRACT

Natural language database access requires support of both query and update capabilities. Although a great deal of research effort has been expended to support natural language database *query*, little effort has gone to support *update*. We describe a model of action that supports natural language database update, and the implementation of a system that supports the model. A major goal of this research is to design a system that is easily transportable to both different databases and different DBMSs.

1. Introduction

Database access includes both query and update. In order to access a database, an end-user has traditionally had two options. He could learn the database structure and the DML required by a particular database, and formulate the access request himself. Alternatively, he could explain his request to a programmer who then writes the DML. Both options have serious drawbacks. In the first, it is not always reasonable for a possibly naive user to learn a formal DML and database navigation strategies. In addition, because of database integrity constraints and view update problems, users are often prohibited from writing transactions that update the database. The second option places a level of administration between the user and the database that is generally cumbersome, and which is inappropriate for a user sitting at a terminal in his office. Such an approach is particularly inappropriate for personal databases.

In order to avoid these impediments to database access, it is desirable to support natural language database access. Although a great deal of research effort has been expended in support of natural language database query [HARR77, DAME78, WOOD76, KAPL79, WALK78, WALT78], and at least one commercial system is available [HARR79], little effort has been expended in support of natural language database update, as noted by Wiederhold [WIED81]. In this paper, we describe a system that supports natural language database updates that are admissible for a given database. An additional goal is to design a system that is easily transportable both to different databases and DBMSs.

2. The Update Problem

Previous research [SALV82] has shown that it is not possible, in a natural manner, to extend natural language query systems to support update. In order to support natural language database query, a computer system must be able to represent a *stative correspondence* between database states and real world states, as shown in Figure 1. This stative correspondence logically connects database objects with real world entities and relationships. The stative correspondence is not adequate for supporting natural language database update. In Figure 2, we see that when some action in the real world causes a state change from RWS1 to RWS2, we must execute a DML sequence to change the database state from DBS1 to DBS2. Given an update command that describes a real world action we need to find a DML sequence that will accomplish the corresponding change in the database. We need to specify *what* is to be modified

* This research is partially supported by NSF grant IST-8214622.

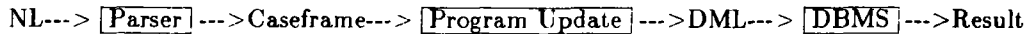


Figure 3
Domain-Independent Natural Language Database Access Architecture

caseframe representation as input, links linguistic elements to database objects and updates, and produces a DML sequence that represents what the natural language command means *with respect to* this database. We are using the INGRES DBMS [HELD75]. Actually, we are not tied to a particular DBMS, as Figure 3 implies. Our system produces a representation in a formal intermediate language (IL) that can be translated into the DML of a given DBMS. Thus, the overall architecture looks like Figure 4.

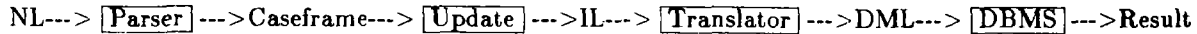


Figure 4
Transportable Natural Language Database Access Architecture

An IL-to-QUEL translator has already been written [ASSI84]. Thus our attention is limited to the nature of the caseframe-to-IL conversion. The architecture of this component is shown in Figure 5.

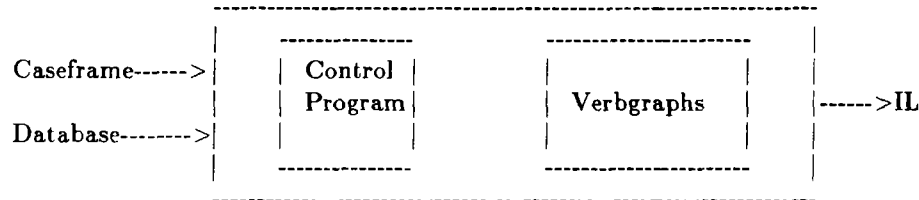


Figure 5
Caseframe-to-Intermediate Language Translator Architecture

Because the IL-to-DML translation is a well-understood problem, and not of further interest here, we sometimes speak of the IL as the language that actually updates the database. The verbgraphs are a set of structures that represents what various natural language update commands mean with respect to this database; they must be specified for each database. The control program directs execution of the verbgraphs; it need never be rewritten. This design is analogous to expert systems where domain-dependent information is represented in production rules. There, the control program is domain-independent; it controls selection and execution of productions.

4. The Verbgraph

A natural language verb may have a number of different senses. A *sense* of a verb roughly corresponds to the different definitions that might be given in a dictionary. For example "run" has at least the three senses: a person moving quickly, a machine operating, and a person campaigning for public office. In our scheme, a verbgraph represents a single verb sense. Thus one verbgraph might represent the "hiring" action, regardless of how the update is specified in natural language. A given verb sense may have a number of *variants*. For example, hiring faculty may require different database actions than hiring secretaries. A verbgraph represents all legal variants of an action, and is also used to determine which variant is specified in the natural language command. The set of verbgraphs for a database is the repository for several kinds of information (constraints and default values may eventually be supported by some ideal DBMS):

1. Linkage of linguistic constructs to database objects and updates. (Use the AGENT case as the value of attribute NAME in EMP, "hiring" results in insertion into EMP.)
2. Constraints on the database. (Maximum of 40 students in a course.)

3. Default values. (All courses are 4 credits, unless otherwise specified.)
4. Parameterized IL update commands that will ultimately comprise the update transaction.
5. Database retrievals that may have to be performed in order to process the update request.
6. Questions to ask the user if insufficient information is specified in the natural language command.
7. Templates for tuples to be inserted into, deleted from, or modified in the database.

A verbgraph is composed of a tree and a blackboard, as shown in Figure 6a.

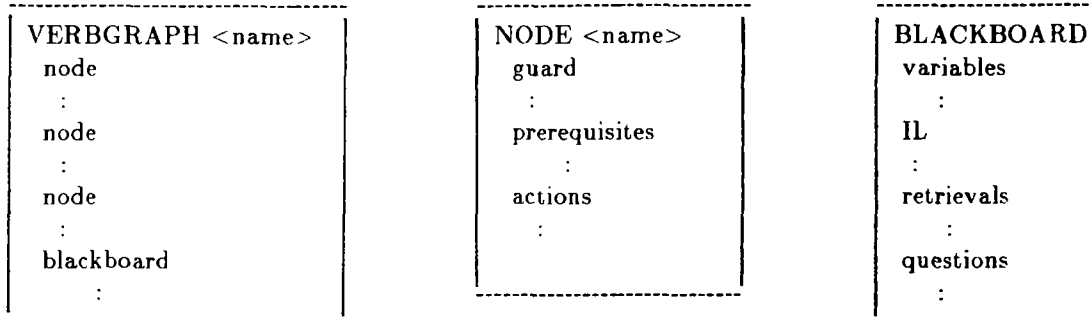


Figure 6a
Verbgraph

Figure 6b
Node

Figure 6c
Blackboard

Figure 6
The Verbgraph

The tree controls instantiation of the blackboard, which contains objects that will ultimately determine the IL sequence. The tree has at most two levels*. There is a distinguished root node and an arbitrary number of leaf nodes. Each node contains a *guard*, a boolean expression. The root node has guard true. A variant is defined by the accumulation of all the nodes in the tree whose guards evaluate to true for a given input. The order of execution of the leaf nodes is unimportant. When the control program executes a verbgraph, it evaluates the guards of all nodes, selects all nodes whose guards evaluate to true, and executes each node in turn. Node execution may cause access to the caseframe, the database, the user, or the blackboard. When all the chosen nodes have been executed, the blackboard will contain a correct IL sequence that will update the database. It is important to note that that *no* update action is taken on the database until all selected nodes have been executed, at which time the IL is translated to DML, and executed by the DBMS.

The Blackboard

The blackboard, shown in Figure 6c, is a common repository of information that any node can access. It is composed of four types of information: variables, parameterized IL update commands, IL database retrieval commands, and questions that may be asked of the user. Variables are of two kinds, tuple and local, which are instantiated during node execution. A *tuple variable* is bound to a relation and is composed of fields that correspond to the attributes of the named relation. The other kind of variable is a *local variable*. The blackboard IL component is a set of labeled IL sequences. Each sequence consists of parameterized IL update operations, which were specified during verbgraph design. An IL update command may cause insertion, deletion, or modification of zero or more tuples in the database. During node execution, the label of an IL sequence may be "checked off." The checked off sequences will comprise the transaction to be translated into DML. The three types of IL updates are insert, delete, and modify. The forms are:

* A two-level tree simplifies our control strategy. We could allow N-level trees, as discussed in [SALV84].

<u>insert</u> <tvar> <u>into</u> <relname> <u>take</u> <attrset> <u>from</u>	<retrieve-label> <set> <interval>
<u>delete</u> <tvar> <u>from</u> <relname> <u>where</u> <boolexp>	
<u>modify</u> <tvar> <u>from</u> <relname> <u>where</u> <boolexp> <u>assign</u> <attrspecs>	

Here we are not concerned with the particular semantics of these statements. It is only important to note their flavor: a pseudo-DML with a high-level insert commands that can cause the insertion of many tuples.

The retrieval component of the blackboard is a set of labeled IL retrieval commands. They are retrievals that may have to be performed against the database to determine the correct variant of a verb sense, instantiate blackboard variables, or process the insert command.

It may be the case that that the natural language update command did not contain sufficient information to complete processing: it may be impossible to determine the variant, or the variant may be determined but the required data incomplete. The questions component of the blackboard is a place for specifying canned natural language questions to be asked of the user.

The Tree

The blackboard stores information needed by the tree. However, the control program does not directly access the blackboard: it selects a set of nodes and executes them. The nodes are responsible for ensuring that the correct IL transaction is constructed. It is in the tree, then, that linguistic objects are linked to database objects, real world actions are linked to database updates, database integrity constraints are specified, and the blackboard objects are manipulated. Conceptually, the root of the tree represents what is true for all variants of this verb sense, and each leaf represents that which is the case for some aspect of a variant. (Recall that a variant is defined by the root and all leaves, or aspects, whose guards evaluate to true.) The tree is specified by a series of node definitions, as shown in Figure 6a. One node is the distinguished root, the remaining nodes are the leaves. The format of a node is shown in Figure 6b. A node consists of three parts: guard, prerequisite, and action.

The *guard* determines whether a node is selected for execution. Guards test caseframe values. Because a variant is determined by information in the natural language sentence (and therefore the caseframe), the guard can only compare caseframe values against constants, caseframe values, and results of retrievals on the database. The *prerequisite* component indicates those caseframe slots that must have values before node execution is to proceed. If any specified caseframe slot does not have a value, a blackboard question is asked of the user, and processing is suspended until the caseframe slot is instantiated.

Actions are the crux of the node. They perform a wide variety of functions: instantiate blackboard variables, ask the user for more information, "check off" blackboard IL, retrieve database values, specify default values, and specify database integrity constraints. An action is either a checkoff or an assignment. The various forms are best illustrated by example. A checkoff is the simplest action. The statement check (A3) causes the blackboard IL sequence labeled A3 to be checked off on the blackboard; it will be part of the final IL transaction.

The basic function of an assignment is to instantiate a blackboard variable. The value may be:

- a constant: t.X := 5
- a system variable: t.X := clocktime
- a blackboard variable: t1.X := t2.Y
- a caseframe value: t.X := cf//location
- a default value: t.X := cf//duration | default(1hour)

If the caseframe slot named duration has no value, use 1 hour.

- the result of a database retrieval: t.X := R1

R1 is the label of a blackboard retrieval that returns a single value.

- required to be in a range of values: t.X := cf//location in R1 # Q8

R1 is the label of a blackboard retrieval, Q8 of a question. If the cf//location value is in the result of R1,

then use it. Otherwise, ask question Q8 and wait for a response.

- required to be in a range of values if it exists, otherwise a default is used:

$t.X := cf//location \text{ in } R1 \# Q8 \mid \text{default } (R2)$

If $cf//location$ has a value and it is in $R1$, then use it. If it has a value but it is not in $R1$, ask $Q8$ and wait for a response.

If $cf//location$ does not have a value, use the default, which is the result of executing blackboard retrieval labeled $R2$.

A formal description of the verbgraph language, a more complete description of the system, and illustrative examples can be found in [SALV84].

5. Concluding Remarks

A prototype of this system is implemented in C on a VAX 11/780 running Berkeley UNIX. We expect the full system to be completed in 1985.

We plan to implement a toolbox for verbgraph specification, which will include a structured editor. We will also provide an interactive verbgraph debugger, which will allow the verbgraph designer to test the correctness of the verbgraphs. He will be able to test which verbgraph nodes evaluate to true for a given input or class of inputs, step through instantiation of blackboard variables and IL checkoff, and query the consistency of verbgraph components.

We also need to address the complex problem of run-time interaction with the user. For example, the user may supply additional information that changes which node's guards evaluate to true. We propose to design our end-user interaction package so that minimum reprocessing is necessary, while allowing maximum flexibility. An important design criterion is to avoid forcing the user to restate information that has been given previously.

Acknowledgments

Michael Siegal participated in the verbgraph design. Thomas Schutz assisted in the implementation.

References

- [ASSI84] Assiff, S. Intermediate Language to QUEL Translation. BU CS Dept Masters Thesis, 1984.
- [BOBR78] Bobrow, R. The RUS System. BBN Report #3878, 1978.
- [CHAR76] Charniak, E. and Y. Wilks, *Computational Semantics*, North Holland, 1976.
- [DAME78] Damereau, F., The Derivation of Answers from Logical Forms in a Question Answering System. *American Journal of Computational Linguistics*, microfiche 75, 1978, pp. 3-42.
- [HARR77] Harris, L., Using the Database itself as a Semantic Component to Aid the Parsing of Natural Language Database Queries. Dartmouth College Mathematics Department TR 77-2, 1977.
- [HARR79] Harris, L., Experience with ROBOT in 12 Commercial Natural Language Database Query Applications. *Proceedings of the International Joint Conference on Artificial Intelligence*, Tokyo, 1979, pp.365-368.
- [HELD75] Held, G., M. Stonebraker and E. Wong, INGRES - A Relational Database Management System. *Proceedings of the 1975 National Computer Conference*, 1975.
- [KAPL79] Kaplan, S.J., Cooperative Responses from a Natural Language Database Query System. Stanford University TR HPP-79-19, 1979.
- [SALV82] Salveter, S. and D. Maier, Natural Language Database Updates. *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*, Toronto, 1982, pp.67-73.
- [SALV84] Salveter, S., Natural Language Database Update. BU CS Dept TR# 84/001.
- [WALK78] Walker, D., *Understanding Spoken Language*. American Elsevier, 1978.
- [WALT75] Waltz, D., Natural Language Access to a Large Database: An Engineering Approach. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1975.
- [WIED81] Wiederhold, G., S.J. Kaplan and D. Sagalowicz, Research in Knowledge Base Management Systems. *SIGMOD Record*, 7, 3, April 1981, pp.26-54.
- [WOOD76] Woods, W. et al, Speech Understanding Systems: Final Technical Report. BBN Report #348, 1976.

CALL FOR PAPERS

ER APPROACH

The 4th International Conference on ENTITY-RELATIONSHIP APPROACH

October 28-30, 1985 Chicago, Illinois

Major Theme: The Use of ER Concept in Knowledge Representation

Sponsored by:



IEEE Computer Society
TC on Database Engineering
TC on Machine Intelligence
TC on Office Automation

in
cooperation
with

University of Illinois
Louisiana State University
Purdue University

Conference Chairman

K.S. Fu
Purdue University

Program Committee Chairman

Jane W.S. Liu
University of Illinois

Tutorial Chairman

Robert Carlson
Illinois Institute of Technology

Local Arrangement Chairman

Adarsh K. Arora
Gould Inc.

Conference Treasurer

Gerald F. Dejong
University of Illinois

Publicity Chairman

Kathy Davis
Northern Illinois University

Steering Committee Chairman

Peter P. Chen
Louisiana State University

Program Committee Members

Adarsh K. Arora	USA
Carlo Batini	Italy
Don Batory	USA
Bruce P. Berra	USA
Yuri Breitbart	USA
David Cohen	USA
Gerald F. Dejong	USA
Elizabeth N. Fong	USA
Robert Fraley	USA
A.L. Furtado	Brazil
A. Jay Goldstein	USA
Udai Gupta	USA
Leslie Hazelton	USA
Yahiko Kambayashi	Japan
Won Kim	USA
Wang Tok Ling	Singapore
Vicent Lum	USA
Hector Garcia-Molina	USA
Martin Modell	USA
J. Mylopoulos	Canada
Peter A. Ng	USA
Ross A. Overbeck	USA
D.S. Parker	USA
Neil Rowe	USA
Hirotsaka Sakai	Japan
Peter Scheucman	USA
Gerhard Schiffner	Germany
Stefano Spaccapietra	France
John F. Sowa	USA
T.C. Ting	USA
Julius T. Tou	USA
Ben Wah	USA
L.X. Zhang	China
Rodney P. Zimmerman	USA

This conference will bring together researchers and practitioners to exchange ideas on the concept of entity relationship and its applications on knowledge representation. Papers on both the principles and the pragmatics of ER approach in knowledge representation are solicited.

Major Topics of Interest Include, But Are Not Limited To:

- Expert and Knowledge Based Systems
- Natural Language Processing
- Learning and Knowledge Acquisition
- Memory and Database Models
- Extension of Entity-Relationship Models
- Management Science Models
- Semantics of Entity-Relationship
- Database Design Tools
- Data Dictionary and Directory
- Database Dynamics

Special Invitation:

Several sessions on business and industrial practices will be organized. Practitioners are invited to submit papers that describe the use of ER approach in system maintenance and enhancement projects, projects using 4th generation languages and project leadership, and experiences in training nonexperts to understand the ER approach.

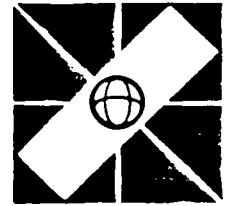
To Submit Your Papers:

- Five copies of double-spaced manuscript should be submitted by March 15, 1985 to the address listed below:
Dr. Jane W.S. Liu
1304 W. Springfield Avenue
Department of Computer Science
University of Illinois
Urbana, Illinois 61801
- Notification of acceptance or rejection will be sent to authors by June 14, 1985.
- For inclusion in the Conference Proceedings, the final, camera-ready manuscript of each accepted paper must be received by the Program Committee Chairman by August 9, 1985.

Important Dates:

Submission Deadline: March 15, 1985
Final Version Due: August 9, 1985

Acceptance Notification: June 14, 1985
Conference Date: October 28-30, 1985



**IFIP Working Group 8.1 Working Conference on
ENVIRONMENTS TO SUPPORT INFORMATION SYSTEM
DESIGN METHODOLOGIES**

Bretton Woods, New Hampshire USA; 4-6 September 1985

Call for Papers

The environment in which information system design methodologies are applied has a strong effect on the design process. Environment, in this sense, refers to a variety of different topics, including:

- the hardware and software facilities available to the information system development organization, including workstations, software tools, and networks;
- the management and structure of the information system development organization;
- the staffing and skills of the information system development organization;
- the ancillary support services, e.g., telecommunications, available to the information system development organization

There is an *ecology* of environments, since changes in any of these areas may affect the other areas. Improvements in the development environment can have a beneficial effect on the development process and/or on the developed product.

We seek papers that address one or more of these areas, with emphasis on the *observed* impact of environmental changes on the use of a methodology or on the results of its use. We are especially interested in papers that discuss the following topics:

- integrated software development environments containing tools explicitly oriented to a methodology;
- the hardware environment, such as a design workstation, to support the development process;
- the communication environment, such as the use of electronic mail and computer networks;
- the quality assurance environment, involving software or organizational procedures to evaluate the development process or its products;
- the physical work environment for information system developers;
- evaluative data on the impact of environmental changes.

Submitted papers should not exceed 6000 words in length and should not have been published or submitted for publication elsewhere. Four copies of the paper should be submitted to the Program Chairman to arrive by 15 February 1985. Authors will be notified of acceptance or rejection by 1 May 1985. Proceedings will be published by North Holland and authors will be expected to sign a copyright transfer agreement. Camera ready copies of the final papers will be due by 15 June 1985.

General Chairman

Prof. Anthony I. Wasserman
Medical Info Science
Room A-16
U. of Calif., San Francisco
San Francisco, CA 94143 USA

1 + (415) + 666-2951

Program Chairman

Prof. Peter C. Lockemann
Institut für Informatik
Universität Karlsruhe
Zirkel 2
7500 Karlsruhe 1
Federal Republic of Germany

49 + (0721) 608-3968

Program Committee

S. Berild (Sweden)
H.M. Blanken (Netherlands)
R. Brooks (USA)
M. Mantei (USA)
M.H. Penedo (USA)
C. Rolland (France)

