

a quarterly bulletin
of the IEEE computer society
technical committee
on

Database Engineering

Contents

Changes to the Editorial Staff	1	A Database System for Engineering Design	48
Letter from the Associate Editor	2	W. Plouffe, W. Kim, R. Lorie, and D. McNabb	
A Selected Bibliography with Keywords on Engineering Databases	3	Using a Relational Database Management System for Computer Aided Design Data—An Update	56
F. Vernadat		M. Stonebraker and A. Guttman	
CAD/CAM Database Management	12	Relational and Entity-Relationship Model Databases and VLSI Design	61
M.L. Brodie, B. Blaustein, U. Dayal, F. Manola, and A. Rosenthal		M.W. Wilkins and G. Wiederhold	
Database Concepts in the Vdd System	21	An Extended Relational Database System for Engineering Data Management	67
K.-C. Chu and Y.E. Lien		Y. Udagawa and T. Mizoguchi	
Revision Relations—Maintaining Revision History Information	26	Integration of Word Processing and Database Management in Engineering Environments	76
M. Haynie and K. Gohl		F. Nakamura, A. Kimura, S. Kanai, and K. Ohmachi	
Database Management and Computer- Assisted VLSI Fabrication	35		
R.H. Katz			
Engineering Data Management Activities Within the IPAD Project	39		
H.R. Johnson			

**Chairperson, Technical Committee
on Database Engineering**

Professor P. Bruce Berra
Dept. of Electrical and
Computer Engineering
111 Link Hall
Syracuse University
Syracuse, New York 13210
(315) 423-2655

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

Dr. Won Kim
IBM Research
K54-282
5600 Cottle Road
San Jose, Calif. 95193
(408) 256-1507

**Associate Editors,
Database Engineering**

Prof. Don Batory
T.S. Painter Hall 3.28
University of Texas
Austin, Texas
(512) 471-1593

Prof. Fred Lochovsky
K52-282
IBM Research
5600 Cottle Road
San Jose, California 95193

Dr. David Reiner
Computer Corporation of America
4 Cambridge Center
Cambridge, Massachusetts 02142
(617) 492-8860

Prof. Randy Katz
Dept. of Electrical Engineering and
Computer Science
University of California
Berkeley, California 94720
(415) 642-8778

Dr. Dan Ries
Computer Corporation of America
4 Cambridge Center
Cambridge, Massachusetts 02142
(617) 492-8860

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 a full member and a participating member of the TC is entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Changes to the Editorial Staff

With this issue, Randy Katz and Don Batory resign as Associate Editors of Database Engineering. Also, I resign as Editor-in-Chief. I have asked Dave Reiner to replace me and guide the publication of the bulletin with a new editorial board.

I shall cherish the pleasure of having worked with my Co-Editors, Alan Hevner, Don Batory, Randy Katz, Dan Ries, Fred Lochovsky and Dave Reiner. The success Database Engineering has enjoyed during the past three years is due largely to their initiatives, cooperation and hard work. The professional and personal friendship that has developed between myself and these outstanding colleagues during the course of shaping and sustaining the direction of the publication more than compensates for all the time and energy I have had to invest since May 1981.

I owe special thanks to Chip Stockton, Director of Publications for the Computer Society, for the extra special way in which he has handled the printing and distribution of Database Engineering. I would like to also thank Jane Liu, past chairperson of the TC, and Bruce Berra, the current chairman of the TC, for their enthusiastic support of the Database Engineering bulletin and attention to all my enquiries and suggestions concerning the TC activities.

This issue contains two papers from Japan, one by Udagawa and Mizoguchi, and another by Nakamura, et al, which, due to late arrival, were included after Randy Katz finalized the manuscript. The September issue will be on multi-media database user interface and is being put together by Dan Ries. Dave Reiner will finish off 1984 with an issue on database design.

Won Kim

Won Kim
San Jose, Calif.
June 1984

Letter from the Associate Editor:

This issue of *Database Engineering Newsletter* focuses on Engineering Data Management. The area has blossomed into a major new thrust of database system research activity since our first issue on the subject two years ago. A number of groups are actively working on the problem in universities and industry, and actual systems are now becoming operational.

Vernadat sets the stage by providing a convenient bibliography of the CAD database literature. Many of these references are from outside the traditional literature read by the database research community, and should prove useful to researchers. Some of the papers are quite old, testifying that design databases are not a new problem!

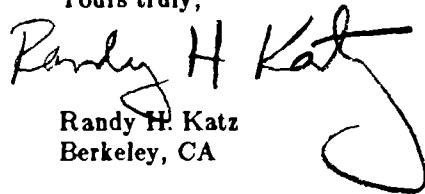
Chu & Lien, Johnson, and Plouffe, et. al., describe the status of several large system building efforts. Brodie, et. al., describes an implementation effort just getting underway at Computer Corporation of America, Inc. Chu & Lien and Plouffe, et. al. deal with electrical CAD, while the other two papers are concerned with the engineering data needs of the aerospace industry.

While much interest has been generated for design databases, this is only half of the engineering task. The papers by Johnson, Brodie, et. al., and Katz place some emphasis on the flip side of design, namely manufacturing.

The papers by Haynie & Gohl and Stonebraker & Guttman describe efforts to extend the relational model with more powerful operations and support structures to handle CAD requirements. Finally, Wilkins & Wiederhold address the performance issues of using a relational database for CAD applications.

With this issue, I step down as an associate editor of the Database Engineering Newsletter. The Newsletter has grown over the last few years to become the primary vehicle for rapid dissemination of recent results to the database system community. This is primarily due to the dogged determination of fearless leader, Won Kim, without whom this newsletter could not possibly have existed. Thanks and credit for our success go to all the associate editors, past and present, and our contributing authors. The future editors will have some mightly large shoes to fill.

Yours truly,



Randy H. Katz
Berkeley, CA

A Selected Bibliography with Keywords on Engineering Databases

F. Vernadat

National Research Council of Canada
Ottawa, Canada, K1A 0R8

The use of computers in the wide range of CAD/CAM activities have introduced database technology in application domains such as mechanical engineering, civil or architectural design, electronic engineering, chemical engineering, automotive industry, aircraft industry, etc., and in more restrictive areas of computer graphics such as geometric modelling, image processing, and pictorial databases. We refer to those database systems as engineering databases. However, over the years it has become recognized that engineering databases differ slightly from classical administrative databases directly issued from database theory. However, people concerned by the design, implementation, or use of engineering databases must not neglect the knowledge of the general theory of database which can provide solutions to numerous problems (storage structures, access methods, concurrency control mechanisms, distributed systems, ...).

Since more and more attention is given to this domain, it is the purpose of the present bibliography to gather together selected major contributions on database theory and on engineering databases for helping interested people to get a rapid state-of-the-art of the field or to get acquainted with a specific topic of the field. In order to guide the reader, a few keywords accompany each reference. (Since all entries not classified in database theory are supposed to deal with database and CAD, these keywords have not been indicated).

The following list of references is a digest of an extensive bibliography available from the author (A Commented and Indexed Bibliography on Data Structuring and Data Management in CAD/CAM. 1970-mid 1983. Res. Rep. ERB-956, National Research Council, Ottawa, CDN, 1984). The original list of references contains 909 entries and the number of keywords is over 180. The bibliography includes a keyword index and an author's index and it contains a bibliographical survey. Both the references listed below and the references listed in the report are written according to the Harvard system.

Some abbreviations have been used as keywords. Their meanings are indicated below:

CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CIM	Computer-Integrated Manufacturing
CODASYL	Conference on Data System Languages
DBMS	Data Base Management System
DDL	Data Definition Language

NRCC 23170

DBMS	Data Base Management System
DDL	Data Definition Language
DML	Data Manipulation Language
E-R model	Entity-Relationship Model
IGES	Initial Graphics Exchange Specification
IPAD	Integrated Programs for Aerospace-Vehicle Design
MRP	Material Requirements Planning
NC	Numerical Control
PCB	Printed Circuit Board
QL	Query Language
VLSI	Very Large Scale Integration;

Many references come from the following publications:

- Computer (monthly)
- Computer-Aided Design (monthly)
- Computer-Aided Design (monthly)
- Computer Graphics (quarterly)
- IEEE Computer Graphics and Applications (monthly)

and also from proceedings of regular technical meetings such as:

- the Design Automation Conference (annual)
- CAD Conference and Exhibition in England (bi-annual)
- Autofact Conference (annual)

other sources of information include

- Working Conferences of Work Group W.G.5.2 and W.G.5.3 of IFIP
- technical papers available from CASA-SME

001. Aggarwal, S., and V. Rajaraman (1983). Computer-aided design of logical data base for information systems. *Comput. Sci. and Inf.*, 13(1), 3-12.
DBMS, automatic programming, logical database design
002. Appleton D.S. (1982). Planning a manufacturing data base. *Autofact IV Conf. Proc.* (Philadelphia, PA), Nov.
mechanical engineering, database planning, manufacturing
003. Armitage, B.S., and P.A.V. Hall (1977). Conceptual schema for CAD. *Comput. Aided Des.*, 9(3), 194-8.
conceptual schema, logical database design
004. Atkinson, M.P. (1980). Data management for interactive graphics. In *Computer Graphics, Infotech State of the Art Report*, 8(5).
Infotech, Maidenhead, UK. pp. 3-23.
computer graphics, DBMS, requirements, data models
005. Atkinson, M.P., and N. Wiseman (1977). Data management requirements for large scale design and production. *ACM SIGDA Newsl.*, 7(1), 2-16.
design engineering, DBMS, requirements
006. Bandurski, A.E., and D.K. Jefferson (1975). Enhancements to the DBTG model for computer-aided ship design. *Proc. ACM Workshop on Data Bases for Interactive Design* (Waterloo, Ont.), Sept., 17-25.
ship design, DBMS, CODASYL model, conceptual schema, subschema
007. Bandurski, A.E., and D.K. Jefferson (1975). Data description for computer-aided design. *Proc. ACM SIGMOD Conf. on Management of Data* (San Jose, CA), May, 193-202.
ship design, DBMS, CODASYL model, DML
008. Baron, N., E. Bornkessel, N. Cullmann, W.F. Klos, and L.F. Magalhaes (1982). An approach to the integration of geometrical capabilities into a data base for CAD applications. In J. Encarnacao, and F.L. Krause (Eds.), *File Structures and Data Bases for CAD*. North-Holland, Amsterdam. pp. 231-43.
geometric modeling, computer graphics, data structures
009. Beeby, W. (1982). The future of integrated CAD/CAM systems: the Boeing perspective. *IEEE Comput. Graphics and Appl.*, 2(1), 51-6.
aircraft industry, DBMS, communication, CAD/CAM
010. Beeby, W.D. (1983). Data-driven automation. The heart of integration: a sound data base. *IEEE Spectrum*, 20(5), 44-8.
aircraft industry, DBMS, communication, CAD/CAM, network
011. Bell, F.E. (1983). CAD to CAM - Using a shared data base. *Autofact V Conf. Proc.* (Detroit, MI), Nov.
mechanical engineering, CAD/CAM, DBMS, manufacturing
012. Bennett, J. (1982). A database management system for design engineers. *Proc. ACM IEEE 19th Design Automation Conf.* (Las Vegas, NV), June, 268-73.
electronic engineering, DBMS, relational model
013. Bittner, J. (1983). Data independence in CAD/CAM data bases. In E. A. Warman (Ed.), *Computer Applications in Production and Engineering*. North-Holland, Amsterdam. pp. 573-587.
CAD/CAM, DBMS, data independence, data modeling, E-R model, DDL, QL
014. Blaser, A. (Ed.) (1980). *Data Base Techniques for Pictorial Applications*. Springer-Verlag, Berlin.
pictorial databases, computer graphics, DBMS, image processing
015. Blaser, A., and U. Schauer (1977). Aspects of data base systems for computer-aided design. *Informatik-Fachberichte*, 11, 78-119.
engineering design, DBMS, requirements, tutorial
016. Bo, K. (1980). Data base design. In J. Encarnacao (Ed.), *Computer-Aided Design, Modelling, Systems Engineering, CAD-Systems*. Springer Verlag, Berlin. pp. 227-61.
logical database, DBMS, data independence, data models, tutorial
017. Bray, O.H. (1983). Data base management in CAD/CAM. *Autofact V Conf. Proc.* (Detroit, MI), Nov.
mechanical engineering, CAD/CAM, DBMS
018. Buchmann, A.P., and A.G. Dale (1979). Evaluation criteria for logical database design methodologies. *Comput. Aided Des.*, 11(3), 121-6.
database theory, logical database design, design methodologies
019. Buchmann, A.P., and T.L. Kunii (1979). Evolutionary drawing formalization in an engineering database environment. *Proc IEEE COMPSAC 79 3rd Int. Computer Software and Applications Conf.* (Chicago, IL), Nov., 732-7.
computer graphics, chemical engineering, process plants
020. Burger, W.F. (1983). MLD: a language and data base for modeling. In E.A. Warman (Ed.), *Computer Applications in Production and Engineering*. North-Holland, Amsterdam. pp. 559-571.
mathematical programming, DBMS, modeling language, user interaction
021. Challis, M.F. (1982). Typing in data base models. In J. Encarnacao, and F.L. Krause (Eds.), *File Structures and Data Bases for CAD*. North-Holland, Amsterdam. pp. 265-79.
data modelling, data types, DDL
022. Chang, N.S., and K.S. Fu (1981). Picture query languages for pictorial data-base systems. *Computer*, 14(11), 23-33.
computer graphics, pictorial databases, relational model, QL
023. Chang, S.K., and K.S. Fu (Eds.) (1980). *Pictorial Information Systems*. Springer-Verlag, Berlin.
computer graphics, image processing, pictorial databases, DBMS
024. Chang, S.K., and T.L. Kunii (1981). Pictorial data-base systems. *Computer*, 14(11), 13-21.
image processing, DBMS, pictorial databases, relational model
025. Cheng Chao Wang, P. (Ed.) (1981). *Automation Technology for Management and Productivity Advancement Through CAD/CAM and Engineering Data Handling*. Prentice-Hall, Englewood cliffs, NJ.
CAD/CAM, data management, information management, applications
026. Cholvy, L., and J. Foisseau (1983). Representation of information in a design process. In E.D. Warman (Ed.), *Computer Applications in Production and Engineering CAPE'83*. North-Holland, Amsterdam. pp. 545-558.
data modeling, semantic integrity, integrity control

076. Kawano, I., H. Fukushima, and T. Numata (1978). The design of a database organisation for an electronic equipment DA system. Proc. ACM IEEE 15th Design Automation Conf. (Las Vegas, NV), June, 167-75. electronic engineering, data structures, requirements
077. Kimura, F., Y. Yamaguchi, Y. Sasaki, K. Kido, and M. Hosaka (1982). Construction and uses of an engineering data base in design and manufacturing environments. In J. Encarnacao, and F.L. Krause (Eds.), File Structures and Data Bases for CAD. North-Holland, Amsterdam. pp. 95-116. CAD/CAM, mechanical engineering, DBMS, geometrical modeling
078. Koller, H., and K. Fruhauf (1981). A data base management system for industrial process control. Comput. Ind., 2, 171-7. reproduction control, process control, DBMS, relational model
079. Korenjak, A.J., and A.H. Teger (1975). An integrated CAD data base system. Proc. ACM IEEE 12th Design Automation Conf. (Boston, MA), June, 399-406. electronic engineering, commercial database, network model, IDMS
080. Kung-Chao Chu, J.P. Fishburn, P. Honeyman, and Y.E. Lien (1983). VDD - A VLSI design database system. Proc. ACM IEEE Annu. Meeting Database Week: Engineering Design Applications (San Jose, CA), May, 25-37. electronic engineering, VLSI, DBMS, relational model
081. Kunii, T.L., and H.S. Kunii (1979). Architecture of a virtual graphic database system for interactive CAD. Comput. Aided Des., 11(3), 132-5. computer graphics, DBMS, interactive system, system architecture
082. Kunii, T.L., S. Weyl, and J.M. Tenenbaum (1974). A relational data base schema for describing complex pictures with color and texture. Proc. 2nd Int. Joint Conf. on Pattern Recognition (Lyngby, Copenhagen, DK), Aug., 310-6. computer graphic, pictorial databases, relational model
083. Kutay, A.R., and C.M. Eastman (1983). Transaction management in engineering databases. Proc. ACM IEEE Annu. Meeting Database Week: Engineering Design Applications (San Jose, CA), May, 73-80. design engineering, DBMS, transactions, integrity control
084. Lacroix, M., and A. Pirotte (1981). Data structures for CAD object description. Proc. ACM IEEE 18th Design Automation Conf. (Nashville, TN), June, 653-9. electronic engineering, semantic data modeling, DDL, schema
085. Lafue, G.M.E. (1978). Design data base and data base design. Proc. CAD 78 3rd Int. Conf. and Exhib. on Computers in Engineering and Building Design (Brighton, UK), March, 254-62. design engineering, DBMS, relational model, integrity control
086. Lafue, G.M.E. (1979). An approach to automatic maintenance of semantic integrity in large design databases. Proc. Nat. Comput. Conf., AFIPS-48. AFIPS Press, Montvale, NJ. pp. 713-6. design engineering, integrity control, semantic integrity
087. Lafue, G.M.E. (1979). Integrating language and database for CAD applications. Comput. Aided Des., 11(3), 127-30. design engineering, languages, integrity control
088. Lafue, G.M.E. (1979). An Approach to Automatic Checking of Semantic Integrity in Design Databases. Ph.D. Dissertation, School of Urban and Public Affairs, Carnegie-Mellon Univ. (Pittsburgh, PA), Dec. design engineering, DBMS, integrity control, semantic integrity
089. Lafue, G.M.E. (1982). Semantic integrity dependencies and delayed integrity checking. Proc. 8th Int. Conf. on Very Large Data Bases (Mexico-City, Mex.), Sept., 292-9. DBMS, integrity control, semantic integrity
090. Lafue, G.M.E., and T.M. Mitchell (1982). Data Base Management Systems and Expert Systems for CAD. Res. Rep. LCSR-TR-28, Lab. for Computer Science Research, Rutgers Univ. (New Brunswick, NJ), May. DBMS, artificial intelligence, expert systems, integrity control
091. Lee, Y.C., and K.S. Fu (1983). Integration of solid modeling and data base management for CAD/CAM. Proc. ACM IEEE 20th Design Automation Conf. (Miami Beach, FL), June, 367-73. CAD/CAM, geometric modeling, DBMS, relational model, grammars
092. Lee, Y.C., and K.S. Fu (1983). A CSG based DBMS for CAD/CAM and its supporting query language. Proc. ACM IEEE Annu. Meeting Database Week: Engineering Design Applications (San Jose, CA), May, 123-30. solid geometry, DBMS, relational model, grammars, languages, QL
093. Leesley, M.E., and A.P. Buchmann (1980). Databases for computer-aided process plant design. Comput. and Chem. Eng., 4(2), 79-83. chemical engineering, DBMS, process plant
094. Leinemann, K. (1982). GRIMBI - A combination of interactive graphics methods and CAD database techniques for functional modelling. Proc. Graphics Interface '82 (Toronto, Ont.), 153-60. computer graphics, data modeling, DDL, DML, interactive systems
095. Leyking, L.W. (1979). Data base consideration for VLSI. Proc. CALTECH Conf. on Very Large Scale Integration (VLSI) (California Institute of Technology, Pasadena, CA), Jan., 275-301. electronic engineering, VLSI, DBMS, data models, hierarchical model
096. Liewald, M.H., and P.R. Kennicott (1982). Intersystem data transfer via IGES. IEEE Comput. Graphics and Appl., 2(3), 55-63. CAD/CAM, systems communication, IGES, graphics databases
097. Lillehagen, F.M. (1981). CAD/CAM systems. In J. Encarnacao, O.F.F. Torres, and E.A. Warman (Eds.), CAD/CAM as a Basis for the Development of Technology in Developing Nations. North-Holland, Amsterdam. pp. 367-414. CAD/CAM, specifications, mechanical engineering, DBMS, TORNADO
098. Lillehagen, F.M., and T. Dokken (1982). Towards a methodology for constructing product modelling databases in CAD. In J. Encarnacao, and F.L. Krause (Eds.), File Structures and Data Bases for CAD. North-Holland, Amsterdam. pp. 59-91. design engineering, DBMS, logical database design, DDL, DML,
099. Lorie, R.A., R. Casajuana, and J.L. Becerril (1979). GSYSR: A Relational Database Interface for Graphics. Res. Rep. RJ2511, IBM Res. Lab. (San Jose, CA), Apr. computer graphics, relational model, interface

100. Lorie, R.A. (1982). Issues in databases for design applications. In J. Encarnacao, and F.L. Krause (Eds.), File Structures and Data Bases for CAD. North-Holland, Amsterdam. pp. 213-29.
design engineering, DBMS, computer graphics, relational model
101. Lorie, R.A., and W. Plouffe (1983). Complex objects and their use in design transactions. Proc. ACM IEEE Annu. Meeting Database Week: Engineering Design Applications (San Jose, CA), May, 115-21.
design engineering, DBMS, relational model, transactions, System R
102. Ludlam, D., and E.J. Purslow (1981). An integrated CAD/CAM system using a relational data base. Proc. European Conf. on Electronic Design Automation (Brighton, UK), Sept., 5-8.
electronic engineering, DBMS, relational model, integration, PCB
103. Ludham, D. (1983). The design database. Comput. Syst., 3(7), 55-7.
electronic engineering, DBMS, relational model, PCB, manufacturing
104. Managaki, M. (1982). Multi-layered database architecture for CAD/CAM systems. In J. Encarnacao, and F.L. Krause (Eds.), File Structures and Data Bases for CAD. North-Holland, Amsterdam. pp. 281-94.
CAD/CAM, DBMS, system architecture, implementation, integrity
105. Matsuka, H., S. Uno, and T. Sata (1981). Application of advanced integrated designer's activity support system. In E.A. Warman (Ed.), Man-Machine Communication in CAD/CAM. North-Holland, Amsterdam. pp. 251-69.
geometric modeling, integrated system, DBMS, relational model
106. Matsuka, H. (1982). Data base in CAD system. Inf. Process. Soc. Jpn. (Joho Shori), 23(10), 1000-7 (in Japanese).
CAD/CAM, design documentation, numerical control, process control
107. McIntosh, J.F. (1978). The interactive digitizing of polygons and the processing of polygons in a relational database. Comput. Graphics, 12(3), 60-63.
computer graphics, DBMS, relational model, interactive system
108. McLeod, D., K. Narayanaswamy, and K.V. Bapa Rao (1983). An approach to information management for CAD/VLSI applications. Proc. ACM IEEE Annu. Meeting Database Week: Engineering Design Applications (San Jose, CA), May, 39-50.
electronic engineering, VLSI, DBMS, semantic data modeling
109. Meder, H.G., and F.P. Palermo (1977). Data base support and interactive graphics. Proc. 3rd Int. Conf. on Very Large Data Bases (Tokyo, J), Oct., 396-402.
computer graphics, DBMS, relational model
110. Miller, R.E., J. Southall, and S. Wahlstrom (1979). Requirements for management of aerospace engineering data. Comput. and Struct., 10, 45-52.
aerospace industry, data management, requirements, IPAD
111. Nash, D. (1978). Topics in design automation data bases. Proc. ACM IEEE 15th Design Automation Conf. (Las Vegas, NV), June, 463-74.
design engineering, DBMS, data models, requirements, survey
112. Nash, J.H. (1982). Graphics interaction with database systems. Proc. CAD 82 5th Int. Conf. and Exhib. on Computers in Design Engineering (Brighton, UK), March, 107-18.
design engineering, DBMS, user interaction, graphics display
113. Neumann, T. (1980). CAD data base requirements and architectures. In J. Encarnacao (Ed.), Computer-Aided Design, Modelling, Systems Engineering, CAD-Systems. Springer-Verlag, Berlin. pp. 262-92.
DBMS, data modeling, data consistency, data integrity, tutorial
114. Neumann, T., and C. Hornung (1982). Consistency and transactions in CAD databases. Proc. 8th Int. Conf. on Very Large Data Bases (Mexico-City, Mex.), Sept., 181-8.
DBMS, data consistency, integrity control, transactions
115. Neumann, T. (1983). On representing the design information in a common database. Proc. ACM IEEE Annu. Meeting Database Week: Engineering Design Applications (San Jose, CA), May, 81-7.
CAD/CAM, manufacturing control, DBMS, E-R model, logical database
116. Niida, K., H.H. Yagi, and T. Umeda (1977). An application of data base management system (DBMS) to process design. Comput. and Chem. Eng., 1(1), 33-40.
chemical engineering, DBMS, plant design
117. Noon, W.A., K.N. Robbins, and M.T. Roberts (1982). A design system approach to data integrity. Proc. ACM IEEE 19th Design Automation Conf. (Las Vegas, NV), June, 699-705.
electronic engineering, VLSI, data integrity
118. Okino, N., Y. Kabazu, H. Kubo, and N. Hashimoto (1980). Geometry data-base for multi-parts in CAD/CAM systems, TIPS/CDB. In P. Blake (Ed.), Advanced Manufacturing Technology. North-Holland, Amsterdam. pp. 71-86.
CAD/CAM, mechanical engineering, geometric modeling
119. Oyake, I., H. Mizuno, and M. Yamagishi (1982). A graphic database for interactive CAD. Proc. CAD 82 5th Int. Conf. and Exhib. on Computers in Design Engineering (Brighton, UK), March, 133-42.
mechanical engineering, geometric modeling, graphic data
120. Palermo, F.P., and D. Weller (1979). Picture building systems. Proc. IEEE COMPOON Spring 79 (San Francisco, CA), Feb.
computer graphics, DBMS, relational model, DDL, DML
121. Palermo, F.P., and D. Weller (1980). Some data base requirements for pictorial applications. In A. Blaser (Ed.), Data Base Techniques for Pictorial Applications. Springer-Verlag, Berlin. pp. 555-67.
computer graphics, DBMS, requirements, relational model, DDL, DML
122. Patnaik, L.M., and N. Ramesh (1982). Implementation of an interactive relational graphics database. Comput. and Graphics, 6(3), 93-6.
computer graphics, DBMS, relational model, DDL, DML
123. Peled, J. (1982). Simplified data structure for "mini-based" turnkey CAD systems. Proc. ACM IEEE 19th Design Automation Conf. (Las Vegas, NV), June, 636-42.
design engineering, DBMS, mini-computers, turnkey systems

124. Phillips, R.J., M.J. Beaumont, and D. Richardson (1979). AESOP: an architectural relational database. *Comput. Aided Des.*, 11(4), 217-26. architectural design, DBMS, relational model, fuzzy relations
125. Phillips, R.J., M.J. Beaumont, D. Richardson, and J. Bartley (1981). Geometry for CAD. *Comput. Aided Des.*, 13(2), 89-97. architectural design, computer graphics, relational model
126. Prior, H., and H. Fuchs (1980). Integrated production of manufacturing documentation. *Ind.-Anz.*, 102(82), 120-7 (in German). CAD/CAM, mechanical engineering, process planning, NC programs
127. Quinlan, K.M., and J.R. Woodwork (1982). A spatially-segmented solids database - Justification and design. *Proc. CAD 82 5th Int. Conf. and Exhib. on Computers in Design Engineering* (Brighton, UK), March, 126-32. mechanical engineering, geometric modeling, implementation
128. Rasdorf, W.J., and A.R. Kutay (1982). Maintenance of integrity during concurrent access in a building design database. *Comput. Aided Des.*, 14(4), 201-7. architectural design, DBMS, relational model, integrity control
129. Roberts, K.A., T.E. Baker, and D.H. Jerome (1981). A vertically organized computer-aided design data base. *Proc. ACM IEEE 18th Design Automation Conf.* (Nashville, TN), June, 595-602. electronic engineering, PCB, DBMS, commercial databases, schema
130. Romberg, F.A. (1981). A Logical Design Methodology for Complex Databases such as a Manufacturing Operations Database. Ph.D. Dissertation, Southern Methodist Univ. (Dallas, TX), Sept. logical database design, methodology, manufacturing
131. Roussopoulos, N. (1979). Tools for designing conceptual schemata of databases. *Comput. Aided Des.*, 11(3), 119-20. database theory, logical database design, methodology
132. Ruiz-de-Molina, E. (1983). The role of data base management and simulation in engineering projects. *Autofact V Conf. Proc.* (Detroit, MI), Nov. CAD/CAM, DBMS, simulation, engineering projects
133. Sanborn, J.L. (1982). Evolution of the engineering design system data base. *Proc. ACM IEEE 19th Design Automation Conf.* (Las Vegas, NV), June, 214-8. electronic engineering, data files, database, file structures
134. Scheffer, L. (1979). Database considerations for VLSI design. In W. M. Van Cleemput (Ed.), *Design Automation at Stanford*. Stanford Univ., Computer Systems Lab., Stanford, CA. electronic engineering, VLSI, hierarchical design, requirements
135. Scott, M. (1980). A data base for small computers. *Prod. Eng.*, 27(3), 50-4. CAM, DBMS, data files, manufacturing operations
136. Shaw, G.W. (1980). The use of database techniques in production control - A practical example from the aerospace industry. *Comput. Ind.*, 1(4), 245-9. production control, aerospace industry
137. Shenoy, R.S., and L.M. Patnaik (1983). Data definition and manipulation languages for a CAD database. *Comput. Aided Des.*, 15(3), 131-4. geometric modeling, DBMS, relational model, DDL, DML
138. Sidle, T.W. (1980). Weaknesses of commercial data base management systems in engineering applications. *Proc. ACM IEEE 17th Design Automation Conf.* (Minneapolis, MN), June, 57-61. engineering databases, commercial databases, DBMS, comparison
139. Smolin, R. (1981). DBMS for bill of materials processing. *Interface Age*, 6(12), 88-91. CAM, bill of materials, DBMS
140. Snook, S. (1979). The database for controlling work-in-progress at Perkins engines. *Database J.*, 9(4), 8-15. CAM, production control, DBMS, process control
141. Sorokin, V.K. (1982). Database organization in computer-aided design systems. *Avtom. and Telemekh.*, 43(9), 122-6. engineering databases, file organization, CAD, design quality
142. Sparr, T.M. (1982). A language for a scientific and engineering database system. *Proc. ACM IEEE 19th Design Automation Conf.* (Las Vegas, NV), June, 865-71. engineering databases, query language, relational model
143. Spoonamore, J.H. (1982). CAEADS - Computer-Aided Engineering and Architectural Design System. Tech. Rep., US Army, Construction Engineering Research Ltd. (Champaign, IL), Aug. architectural design, geometric modeling, DBMS, CAEADS
144. Stonebraker, M., B. Rubenstein, and A. Guttman (1983). Application of abstract data types and abstract indices to CAD data bases. *Proc. ACM IEEE Annu. Meeting Database Week: Engineering Design Applications* (San Jose, CA), May, 107-13. DBMS, abstract data types, abstract indices, boxes, wires, polygons
145. Taraman, S.R. (1981). Machining data bank structure. Tech. Paper MS81-490, CASA-SME, Dearborn, MI. CAM, mechanical engineering, machining, file structures
146. Throop, J.W. (1981). A common data base for metal cutting data. Tech. Paper MS81-183, CASA-SME (Dearborn, MI), April. CAM, logical database design, machining, standardization
147. Ulfsby, S., S. Meen, and J. Oian (1982). TORNADO: a database management system for graphics applications. *IEEE Comput. Graphics and Appl.*, 2(3), 71-9. CAD/CAM, mechanical engineering, DBMS, network model, CODASYL
148. Ulfsby, S., S. Meen, and J. Oian (1982). TORNADO: a DBMS for CAD/CAM systems. In J. Encarnacao, and F.L. Krause (Eds.), *File Structures and Data Bases for CAD*. North-Holland, Amsterdam. pp. 335-50. CAD/CAM, mechanical engineering, DBMS, network model, CODASYL
149. Valle, G. (1975). Relational data handling techniques in integrated circuit mask layout procedures. *Proc. ACM IEEE 12th Design Automation Conf.* (Boston, MA), June, 407-13. electronic engineering, IC masks, mask layout, relational model

150. Valle, G. (1977). Relational data handling techniques in computer aided design procedures. In J.J. Allan (Ed.), CAD Systems. North-Holland, Amsterdam. pp. 309-25.
design engineering, database, relational model
151. VanCleemput, W.M., and J.G. Linders (Eds.) (1975). Data Bases for Interactive Design. University of Waterloo, Waterloo, CDN.
design engineering, computer graphics, data structures, DBMS
152. Vernadat, F. (1983). Communication: a key requirement in computer integrated manufacturing. Proc. IEEE 2nd Annu. Phoenix Conf. on Computers and Communications (Phoenix, AZ), March, 193-8.
CAD/CAM, CIM, activities, communication, requirements
153. Vernadat, F. (1983). New requirements for user interaction with CAD/CAM databases. Proc. Graphics Interface '83 (Edmonton, CDN), May, 271-9.
CAD/CAM, DBMS, user interaction, data types, data models, DDL, DML
154. Vernadat, F. (1983). Manufacturing Databases. Res. Rep. ERB-955, Div. of Electrical Engineering, National Research Council of Canada, Ottawa, CDN.
CAD/CAM, DBMS, logical database design, manufacturing applications
155. Waters, M.A. (1978). A Data Base for Efficient VLSI Checking and Artwork Generation. Intren. Rep., ECI Div. of E-Systems, Inc. (St. Petersburg, FA), June.
electronic engineering, VLSI, circuit layout
156. Weller, D., and F. Palermo (1979). Database requirements for graphics. Proc. IEEE COMPCON Spring 79 (San Francisco, CA), Feb., 231-7.
computer graphics, DBMS, relational model, requirements
157. White, C. (Ed.) (1980). The nature of graphics databases. In Computer Graphics, Infotech State of the Art Report. Infotech, Maidenhead, UK. pp. 139-70.
computer graphics, graphics databases, requirements
158. Wiederhold, G. (1981). Research in knowledge base management systems. ACM SIGMOD Newsl., 11(3).
electronic engineering, digital circuits, DBMS
159. Wiederhold, G., A.F. Beetem, and G.A. Short (1982). A database approach to communication in VLSI design. IEEE Trans. Comput. Aided Des., CAD-1(2), 57-63.
electronic engineering, VLSI, commercial database, DBMS-20
160. Williams, R., and G.M. Giddings (1976). A picture-building system. IEEE Trans. Software Eng., SE-2(1), 62-6.
computer graphics, image processing, relational model, DDL
161. Wilmore, J.A. (1979). The design of an efficient data base to support an interactive LSI layout system. Proc. ACM IEEE 16th Design Automation Conf. (San Diego, CA), June, 445-51.
electronic engineering, mask layout, DBMS, interactive system
162. Wong, C.S., and E.R. Reid (1982). FLAIR - User interface dialog design tool. Comput. Graphics, 16(3), 87-98.
computer graphics, user interface, DBMS, relational model
163. Wong, S., and W.A. Bristol (1979). A computer-aided design data base. Proc. ACM IEEE 16th Design Automation Conf. (San Diego, CA), June, 398-402.
electronic engineering, DBMS, centralized database
164. Wood, C., E.B. Fernandez, and R.C. Summers (1980). Data base security: requirements, policies, and models. IBM Syst. J., 19(2), 229-52.
database theory, DBMS, data security
165. Yasky, Y. (1981). A Consistent Database for an Integrated CAAD System: Fundamentals for an Automated Design Assistant. Ph.D. Dissertation, Dept. Architecture, Carnegie-Mellon Univ. (Pittsburgh, PA).
architectural design, DBMS, data consistency, data integrity
166. Zdeblick, W.J., J. Lindberg, and L.J. Hawkins (1981). Machinability Data Base for End Mill Application. Tech. Paper MS81-184, CASA-SME (Dearborn, MI), April.
CAM, process planning, machining
167. Zintl, G. (1981). A CODASYL CAD data base system. Proc. ACM IEEE 18th Design Automation Conf. (Nashville, TN), June, 589-94.
electronic engineering, network model, CODASYL, DML

CAD/CAM Database Management*

By

Michael L. Brodie, Barbara Blaustein, Umeshwar Dayal,
Frank Manola, Arnon Rosenthal

Computer Corporation of America

1. The CAD/CAM Database Management System

The design and manufacture life cycle of a product involves many distinct engineering disciplines, each with its own specialized computer systems. Generally these systems have massive data repositories handled by inadequate data managers that cannot communicate with each other. As a result, there are significant problems for data management and for the efficient coordination of hundreds of complex heterogeneous systems.

Data provides a basis for integration and coordination. Design and manufacturing data consists of all types of product descriptions including requirements, drawings, parts hierarchies, geometries, analyses, and manufacturing processes, as well as administrative data for controlling, monitoring and planning. Although the same data is frequently used in different forms by many systems, it seldom is shared or exchanged automatically.

Ideally, a DBMS would be used to integrate current and future CAD/CAM systems by providing means to store, manipulate, and manage all CAD/CAM data. However, current database technology does not provide means to solve such problems as representing engineering data semantics (e.g., parts hierarchies and geometry), version control, data exchange, distributed processing over heterogeneous databases, and suitable interfaces.

A CAD/CAM DBMS (CCDBMS) which solves the above problems is being designed and developed to integrate all CAD/CAM systems. Each system will continue to operate autonomously, but some measure of global control and access will be imposed. The components of the CCDBMS architecture fall into three functional groups. First, a set of user interface components will provide uniform access to all CCDBMS facilities, including those of the individual CAD/CAM systems. Second, a Global Data Manager will provide distributed processing for CCDBMS requests that require access to more than one system within the CCDBMS. Third, a new DBMS will provide a global view of all data needed to support queries against the whole CCDBMS, distributed processing, and version control. The global view will include an abstract or extract of all data in the CCDBMS as well as a global dictionary and directory.

*This research is funded by General Dynamics, Data Systems Division.

This paper summarizes the research results on which the CCDBMS is based. Early research results indicated that two approaches to the problem, as we considered it, were not feasible. First, a centralized database for all CAD/CAM data would make the integration of current and future CAD/CAM systems economically infeasible. Second, the problem of determining one or more standard data representations for all CAD/CAM data is currently intractable due to mathematical problems with translation between representations and the real need for specialized representations.

2. Data Model and Languages

The CCDBMS uses the functional data model Daplex [Shipman81]. Characteristics that suit it for this application are:

- Like the relational model, it provides high-level set-oriented operations, and relatively user-friendly query capabilities.
- Again like the relational model, it provides the basis for the construction of powerful DBMSs [Chan81, Chan83].
- It supports the interaction of heterogeneous data collections as in Multi-base [Smith81, Dayal83].
- Daplex permits a straightforward modelling of "complex objects", which are important constructs in design-oriented applications [Lorie81], via entity-valued functions (attributes).
- Daplex allows explicit declaration of ISA hierarchies, which is an important semantic construct for this application.

It is interesting to note that Daplex has definitional capabilities similar to those of IDEF-1, which is being used in design and manufacturing applications for conceptual modelling [IDEF-1]. IDEF-1 models map to Daplex schemas in a straightforward way.

The current conceptual model being developed consists of information about parts and related documents, such as drawings, specifications, and change notices (the ISA relationship is useful here). Extensions to this model also have been investigated for including manufacturing data (e.g. group technology and planning data) and analysis data such as finite element models, test cases, and results of analysis programs.

The information about parts in the schema is a generalization of the usual parts-explosion (or bill-of-materials) structure used in databases in that particular instances of a (next-level) subcomponent part within a higher-level assembly must be represented. For example, this generalization is necessary to hold the orientation data that defines the geometric relationship of each individual part instance to the assembly.

Extensions to Daplex are currently being investigated in order to meet the special demands of CAD/CAM applications. Geometric data handling extensions are discussed in Section 4. Another important example is support for

computation of transitive closures within the parts hierarchy. Although Daplex allows explicit looping and if-then testing, traversing a parts-hierarchy is not much easier in Daplex than it is in other languages. As a result, it appears desirable to have special syntax to support the particular problems of querying parts hierarchies and other similar cyclic structures.

We also are investigating special constructs for defining parts hierarchies. Just as the generalization hierarchies (defined using the ISA construct) include special semantics (such as inheritance), so the parts hierarchy (defined using new constructs) might include its own special semantics (such as acyclicity). Further extensions also might be justified since the conceptual schema is extended to deal with additional types of design and manufacturing data, as well as individual processes that use it.

3. Screen Oriented Interface

A uniform screen oriented interface is being designed to provide access to all CCDBMS functions including database query and update, data dictionary operations, definition and execution of distributed transactions, system commands, and help facilities. It must be appropriate for thousands of design and manufacturing engineers whose level of skill and familiarity with computer systems (such as database systems and query languages) vary widely.

Most CCDBMS functions including query, update, and dictionary commands, will be available via an interface based on concepts from OBE [Zloof82] and DACOS [Kaufman83]. Separate screen interfaces will be provided for specialized functions such as system commands. The research issues involved the extension of OBE to support Daplex.

The syntax and semantics of OBE had to be extended for data definition, query, and manipulation. The major extensions, which concern the object and set orientation of Daplex, involved additional domain and data type capabilities. Concepts for values, variables, and operations were added for entity types, entity subtypes, set types, and some engineering data types. Daplex constraints such as overlap of type hierarchies and entity uniqueness were also added. QBE's predicates have been used to integrate operations and predicates for entities, sets, and geometry. Extensions were made for the expression and display of queries that involve transitive closure, including the ability to express queries over any recursively defined structure and to retrieve the relationships between the nodes in the structure. QBE and OBE function and program invocation mechanisms have been generalized to permit any function or program to be invoked. This is particularly important for predefined transactions over the CCDBMS. Mechanisms for arbitrary view definition and maintenance are now being considered.

Significant extensions have been made to the QBE display syntax. Screens can contain multiple tables, each representing one entity or a user defined view. Relationships between tables are displayed via entity and other

variables as well as by views. To simplify the potentially complex display, graphic devices such as scrolling, zooming, windowing, editing, and overlaying have been introduced. Mechanisms have been integrated for entity and set valued variables, predicates, and functions. New windows have been introduced to display queries and errors. As a screen query is entered, the CCDBMS can automatically display the corresponding Daplex query for tutorial and checking purposes. Many other features have been added such as specialized displays of results to hierarchic queries, and search broadening and narrowing.

Future investigations are planned for the definition, update, and browsing of local and global schemas, schema mappings, local systems capabilities, and version control rules. More sophisticated graphical concepts based on VIEW [Barnett82] are being investigated.

4. Geometric Query Capabilities, Processing, and Representation

Geometric information currently is generated and displayed on workstations. The CCDBMS will store this information centrally and provide access to parts information via geometric conditions. For example, in considering a design change, all parts close to the point of change can be found and their properties (e.g., low melting point) accessed.

The parts hierarchy shows the component parts of each part in the database. Queries to a part hierarchy, particularly geometric queries, raise difficult issues in query language behavior and data management, in addition to issues of computational geometry. Special semantics are used for queries to part hierarchies to enable the system to return results at the proper level of detail. For example, a query requesting parts within 4 feet of an airplane instrument panel normally should not return the parts "airplane" or "cockpit", not every bolt in the pilot's seat. Instead, "pilot's seat" could be returned.

A system of rules addresses these problems by categorized geometric predicates according to their behavior when going down the parts hierarchy. The rules prevent unwanted output and speed the search process. The output limitation rules can be informally expressed as:

- On predicates that easily are satisfied by large (small) parts, return only the smallest (largest) satisfying parts.
- Return parts at approximately the same level of decomposition.
- Do not decompose unimportant parts.

The geometric predicates supported are "contained" and "intersects". Other geometric operators change coordinate systems (parts are stored relative to their immediate superassembly), define regions relative to a part, calculate distances, etc.

Rather than doing sophisticated geometric calculations, the DBMS works with approximations to actual shapes. We chose rectangular boxes, since these permit refinements (sets of boxes), allow useful regions to be defined (e.g., the region directly to the left of x), and are easily manipulated. Since the geometric representation of a part is its containing rectangular box, we sometimes obtain false positives.

In our environment, we cannot use an index structure for geometric information. The difficulty is that moving a large assembly (e.g., a pump) potentially changes the position of hundreds of parts and could cause hundreds of database updates. Instead, the search algorithm uses the part hierarchy itself as the branching structure. The search down a branch terminates when subparts are reached that cannot satisfy the search predicate or when an output-limitation rule halts further decomposition.

5. Version Control

A number of issues often are thrown together under the heading "version control" or "configuration management". This function is defined as: "The systematic approach to identifying, controlling, and accounting for the status of the parts and assemblies required in a product and/or design from the point of its initial definition throughout its entire life" [Knox83] (emphasis added).

There are three main components of version control, each of which imposes a set of requirements on a CAD/CAM DBMS: 1) Schema definition (discussed above): recording status and descriptive information about products, numerically-controlled tool programs, schedules, etc.; 2) Access control: restricting access of various users to data at various stages in the life of a product; and 3) Change control: monitoring updates, recording change requests and approvals, and changing data availability as a product moves from one design or manufacturing stage to another.

An effective CAD/CAM access control system should support four functions: 1) value-based access privileges, 2) definition of access privileges at the attribute (function) level, 3) linking of some access privileges to particular pre-defined transactions, and 4) flexibility granting and revoking of privileges (including privilege to granting and revoking).

Value-based access privileges work together with the schema definition since database values are used to determine user access to data at a particular time. Attribute-level access privileges are essential to provide the degree of control required in the CAD/CAM environment. However, supporting attribute-level access privileges becomes complicated when a schema allows entity-valued attributes (or functions) needed to capture complicated relationships in CAD/CAM. The CCDBMS can control the kinds of updates allowed by requiring the use of predefined transactions to update sensitive data. Different types of privileges are needed for different products and for different

stages within the life of a single product, and different (relatively autonomous) groups involved in design and manufacturing may wish to follow different access control policies. The CCDBMS must therefore support a closely controlled system for defining and changing access privileges.

Change control is perhaps the most difficult aspect of version control. "The basic change control process consists of six major functions: (1) determination of the need for a change, (2) identification and logging of change, (3) description and documentation of change, (4) evaluation and approval of change, (5) incorporation of change in hardware, and (6) verification and documentation of change incorporation" [Samaras].

The role of an automated system in supporting change control is chiefly to monitor various types of changes and, when possible, to initiate appropriate actions. CCDBMS support for change control could be partially embedded in predefined transactions. Conditions to be tested could be encoded in the appropriate transactions, and if necessary one predefined transaction would invoke another.

Access and change control share three fundamental components: the abilities to activate an access or change control procedure when a particular event occurs, to evaluate a boolean formula and, based on the evaluation, to invoke an appropriate action. Because of the great variability in the types of access privileges and change monitoring needed for different products and at different stages within the life of a product, the CCDBMS must be able to store access and change control information (or rules) and to enforce these rules using general techniques. General techniques developed for rule-based expert systems may be useful in enforcing these rules.

6. Distributed Processing

The CCDBMS is a heterogeneous distributed DBMS. Its salient capabilities include: High-level data definition facilities for a tight integration of part, drawing, and version control information (which currently is scattered in many different repositories under the control of different DBMSs, file systems, and application systems); global version and access control; efficient processing of ad hoc global queries over the global view; and centralized management of global predefined transactions.

The design of the CCDBMS applies and extends MULTIBASE [Smith81, Landers83] technology. As in MULTIBASE, the problems of heterogeneity and data integration are handled separately by two different kinds of software modules: the former by the Local System Interfaces (LSIs), and the latter by the Global Data Manager (GDM). The GDM is the key component of the distributed processing architecture. It presents a unified view of the heterogeneous repositories; it accepts, plans, and monitors the execution of user requests; and it returns results to the user. The role of the LSIs is to present a uniform interface between the GDM and the heterogeneous host systems

by hiding the idiosyncracies of the host systems from the GDM. The GDM can thus communicate with all the sites via a common protocol, expect data returned from all the sites in a common format, and make consistent assumptions about the transaction management capabilities of all the sites.

Data in each repository is first described by a Local Schema (LS) in a common language. Then a logically integrated view, the Global Schema (GS), is defined over the LSs. Global query processing in the CCDBMS is similar to the one used in MULTIBASE. (See [Dayal81, Dayal82a, Dayal82b, Dayal83, Goldhirsch84] for details.) Global queries over the GS are input to the GDM from the user interfaces in an internal form called a query processing envelope (QPE). The GDM uses the definition of the GS to transform the global QPE into QPEs over the LSs. It then constructs an efficient global query processing strategy that consists of local QPEs (each of which represents a query over a single LS, and is executable at a single site), move steps that specify data transfer between sites, and a partial order that specifies precedence constraints on the execution of the local QPEs and move steps. The local QPEs are shipped to the appropriate LSIs, which translate them into efficient programs in the host DMLs. The results of executing the host programs are formatted and returned to the GDM, which combines them into a final answer that is returned to the user.

Unlike MULTIBASE, the CCDBMS must process update transactions in addition to queries. This raises two sets of issues. First, updates on the GS must be mapped into equivalent updates on the LSs. But automatic view update mapping is a notoriously difficult problem. Hence, we initially avoid the problem by insisting that all global transactions be predefined (i.e., the mapping of every global transaction into local transactions must be supplied to the GDM before the global transaction is invoked).

The second set of issues pertains to global transaction management: concurrency control, commitment, and recovery. These problems are more complicated here than in "conventional" distributed DBMSs for two reasons. First: the host systems might not provide a uniform set of capabilities. For simplicity, we use centralized two-phase locking at the GDM to synchronize global transactions, centralized deadlock detection at the GDM, and centralized two-phase commitment with the GDM as coordinator. In addition, we require that each site be capable of synchronizing purely local transactions, detecting local deadlocks, and supporting atomic commitment. Capabilities that are lacking in the host system must be compensated for by the LSI. Second: CAD/CAM transactions typically are long. Using an entire transaction as the unit of commitment would result in poor resource utilization. Our solution is to (a) treat each transaction as though it were composed of several atomically committable units; and (b) use special version control procedures for synchronizing special groups of predefined transactions (e.g., use approval, release, and check-out procedures for synchronizing design, redesign, and manufacturing transactions).

Future extensions will consider the processing of ad hoc global transactions, decentralized transaction management, enhanced robustness through replication, and issues of autonomy.

7. References

- [Barnett82] Barnett, J., M. Friedell, and D. Kramlich, "Context-Sensitive, Graphic Presentation of Information," Computer Graphics, Vol. 16, No. 3, 1982.
- [Chan81] Chan, A., S. Fox, K. Lin, D. Ries, "The Design of an Ada Compatible Local Database Manager", Technical Report CCA-81-09, Computer Corporation of America, 1981.
- [Chan83] Chan, A., U. Dayal, S. Fox, N. Goodman, D. Ries, D. Skeen, "Overview of an Ada Compatible Distributed Database Manager", Technical Report CCA-83-01, Computer Corporation of America, 1983.
- [Dayal81] Dayal, U., et al., "Local Query Optimization in MULTIBASE: A System for Heterogeneous Distributed Databases," Technical Report CCA-81-11, Computer Corporation of America, September 1981.
- [Dayal82a] Dayal, U., T. Landers, and L. Yedwab, "Global Query Optimization in MULTIBASE: A System for Heterogeneous Distributed Databases," Technical Report CCA-82-05, Computer Corporation of America, 1982.
- [Dayal82b] Dayal, U., and N. Goodman, "Query Optimization for CODASYL Database Systems," Proceedings ACM SIGMOD Conference, June 1982, pp. 138-150.
- [Dayal83] Dayal, U., "Processing Queries over Generalization Hierarchies, in a Multidatabase System", Proc. Ninth VLDB Conference, Oct. 1983.
- [Goldhirsch84] Goldhirsch, D., and L. Yedwab, "A Hybrid Approach for Handling Generalized Entities in Views," Computer Corporation of America, January 1984 (submitted for publication).
- [IDEF-1] Softech, Inc., Integrated Computer-Aided Manufacturing (ICAM) Architecture Part II, Volume 5--Information Modeling Manual (IDEF-1), AFWAL-TR-81-4023, Wright-Patterson AFB, Ohio, June 1981 (NTIS AD-B062-458).
- [Kaufman83] Kaufman, C., J. Barnett, and B. Blaustein, "The DACOS Forms Based Query System," Journal of Telecommunications Networks, Computer Science Press, Rockville, MD, winter issue, 1984.
- [Knox83] Knox, Charles S., CAD/CAM Systems: Planning and Implementation, Marcel Dekker, Inc., New York, 1983.
- [Landers83] Landers, T., and R.L. Rosenberg, "An Overview of MULTIBASE." In H.J. Schneider (ed.), Distributed Databases, North-Holland Publishing Company, 1982.

CAD/CAM Database Management

- [Lorie81] Lorie, R.A., "Issues in Database for Design Applications", IBM Research Report RJ3176, IBM Research Lab., San Jose, CA, July 1981.
- [Samaras] Samaras, T.T., Engineering Graphics Desk Book, Prentice-Hall.
- [Shipman81] Shipman, D., "The Functional Data Model and the Data Language Daplex," ACM Transactions on Database Systems, Vol. 6, No. 1, March 1981.
- [Smith81] Smith, J.M., et al., "MULTIBASE--Integrating Heterogeneous Distributed Database Systems", Proc. National Computer Conference, Chicago, May 1981.
- [Zloof82] Zloof, M.M., Office-by-Example: A Business Language that Unifies Data and Word Processing and Electronic Mail, IBM Systems Journal, Vol. 21, No. 3, 1982.

Database Concepts in the Vdd System

Kung-Chao Chu
Y. Edmund Lien

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

The VLSI design database (Vdd) system is a set of programs targeted to assist a circuit designer in layout design, verification, and simulation. As a subsystem in a package of integrated design aids, it also provides interface to other higher-level tools. We state the major problems in designing Vdd and outline how these problems can be solved using existing database techniques. The problems are: the need to have two distinct representations of a chip - a language description and a database representation, the desirability to support chip design in different silicon processing technologies, and the need to treat a design session as a database transaction. The database techniques used include database modeling by schemas, secondary indexing, swapping of data in and out of the main storage according to data semantics, and atomic transaction commitment.

1. INTRODUCTION

The Vdd system [Chu] is part of a design package called *Ida*, for Integrated design aids, developed in the research area at AT&T Bell Laboratories. *Ida* is designed to support individual or a small team of chip designers in an integrated design environment. Integration, one of today's industry buzzwords, means in our context a set of programs that is sufficient to aid the designer in the entire chip design cycle, nicely packaged with clean interface between programs, and not overly complicated to intimidate the designer.

At present, *Ida* includes the layout design subsystem Vdd, logic and circuit simulators, routers, and layout generators. It has been operational for more than a year. A typical design environment to run *Ida* is a "personal" work station with color graphics in a UNIX† operating system.

The Vdd software assists the designer in design activities ranging from layout editing, layout programming, design-rule checking, plotting, access to cell libraries, and interface to simulators.

2. DESIGN CONSIDERATIONS IN Vdd

We set out in the beginning three goals that the Vdd software should achieve. First, it has to support a syntax-oriented layout representation, i.e., a layout description language. We will discuss the pros and cons of a description language shortly. Second, the Vdd system should be technology independent as much as feasible, and this independence should be achieved without requiring the designer to work with primitive geometric components. Since MOS technology is significantly different from others and is the main force in the silicon industry, we decided that Vdd should support all varieties of the MOS family. Third, Vdd should be complete in its own right for a layout designer. In other words, a person skillful in chip layout should be able to use Vdd to design, verify, and simulate a chip and to produce a mask representation ready for chip fabrication. This goal, if couched in database terms, is the ability to support a design transaction.

† UNIX is a Trademark of AT&T Bell Laboratories.

2.1 The Role of Description Languages

Using a language to describe a chip is fairly common. The language approach has been used at different levels of abstraction. We have seen languages like XYMASK [Fowler], *cif* [Mead], *i* [Johnson], LSL [Bose], and PMS and ISP [Bell]. For chip layouts, a language offers a concise and powerful way to describe the components and the interconnection of components in a chip. Geometric constraints and electrical connectivity can be easily stated in such a way that "binding" to real physical mask layers and actual coordinates can be done at the compilation time.

A strong argument for having a powerful layout language in a VLSI design system is to ease the task of automatic layout generation. Tools that generate layouts from high-level specifications need a target language. The more flexible the layout language is, the simpler the task of implementing such a generator.

On the other hand, a layout program can be time and space consuming to translate. A typical 8-bit microprocessor takes about 3 million characters to describe in the *i* language and after translation, it occupies about 5 million bytes of storage in the address space in the UNIX system (Berkeley version 4.1c). If a minor editing of the top-level cell requires the complete translation of this chip, it can take at least half an hour on a lightly-loaded VAX† system. Moreover, the updates have to be translated into the language description.

A language based design system would also require all tools to have this translator as their front-end. While many tools only need partial information of a chip, for example, a logic simulation need not know the detailed mask layout, lack of a means to build an index structure into the chip description results in a heavy time and space penalty in these tools.

2.2 Technology Independence

As we stated earlier, technology independence can be accomplished easily if we require the designer to deal only with rectangles. The interpretation of the geometric primitives in the context of VLSI and the composition of these primitives to form electrical primitives such as transistors and contacts, will then be left with a circuit extractor.

One alternative is to define a set of electrical primitives at the layout level and to support all these primitives in every tool. Of course the trick is that these primitives should not be limited to one particular variety of MOS technology. This approach makes the life of the chip designer easier at the expense of the life of a tool designer.

2.3 Design Transactions

The layout of a chip is often created in one of the three means: it is generated by another tool, like a PLA generator, it is created by layout programming using a text editor, or it is created by interactive graphics editing. The first two describe the end result in a layout language. The last does not necessarily lead to a layout program.

In fact, the last approach can best be thought of in terms of a database transaction. A design transaction involves a sequence of layout display, layout changes, and commitment of changes to the design. The layout information can also be retrieved for the purpose of simulation or verification. Two most obvious needs in a design transaction are to provide random access to parts of a chip and to implement atomic transaction commitment.

Once we start to parallel a circuit design activity and a conventional business database transaction, we can take advantage of the well understood techniques such as the B-tree data structure and page shadowing. Katz and Weiss provides a more thorough coverage of the issues concerning design transaction management [Katz].

† VAX is a Trademark of Digital Equipment Corporation.

3. DATABASE TECHNOLOGY APPLIED TO VLSI DESIGN

In the rest of this article, we describe the particular approaches we took in Vdd to resolve the problems listed above. Our main contributions are two: We use a relational schema to describe a silicon processing technology, therefore, tools are made independent of the specifics of a technology. This is essentially the manifestation of data independence in design tools. The other main idea is to organize a chip database for every chip design. The description of a chip is stored in several crash-resistant B-trees, which provide efficient indexing as well as the capability to perform atomic transaction commitment.

3.1 Data Model of Processing Technology

The information about the MOS technology family can be represented by a relational database. Each technology is represented by a database instance. We envision that there will be an expert who knows the details of a specific technology and can specify them for the general user community. This *technology administrator* will prepare a technology database to match the local processing capability.

We started out with a generic model of MOS technology. A taxonomy was then developed, providing a means for the designer to define wires, how wires are grouped to form connections and transistors. All tools were designed to understand this generic MOS technology, but the specific details of a technology were left in the technology database.

In designing the schema of the technology database, we follow a few principles. We want to hide the processing details from the circuit designer. For instance, one should be able to describe transistors and connections between transistor terminals without specifying the layers of the transistors. It is also desirable to treat a combination of related layers as one composite wire.

Our generic MOS circuit layout model consists of three conceptual object types: a layer type, a contact type, and a transistor type. For example, a wire type is represented by the following relation

wire(name, minwid, color, inner, outer, rim, subslevel, subsrim, xymask)

A wire instance has a name, a minimum width, and a color representation. A wire is a composition of at most three layers. The attributes inner, outer, and subslevel refer to the individual layers. These attributes can have other wire names as values. The attribute rim and subsrim give the extensions of the outer layer and the substrate layer relative to the inner layer, respectively. Finally, the name of the mask in XYMASK is also given.

Likewise, an instance of a contact type must contain information about its component layers, its substrate if any, their sizes, the locations and names of reference points in the contact. Each instance of a transistor type is assumed to be a four-terminal MOS transistor with an end-user definable channel size, reference points, and four functional parts (a gate, two symmetrical pieces for source and drain, and a substrate). The layer names of the four parts are specified by the technology administrator.

Similarly, the details of design rules can be coded in a relation. This allows our design-rule checker to be independent of the technology.

The information in the technology database is retrieved via the database schema. All tools are then bound to the specific technology at run time.

What we have demonstrated is that it is possible to elevate the level of sophistication of design tools by permitting certain generic MOS information in the software while leaving other out and making it updatable by the design community.

By going to a set of higher level primitives such as variable-sized transistors, we have to give up some flexibility found in conventional systems. For now, we can only deal with MOS transistors with rectangular channel area. More complicated transistor types can be handled as cells.

3.2 A Family of Layout Languages

With the generic MOS model in place, we designed a family of MOS layout languages, each of them tailored to a specific technology. For example, two members of the family are layout languages for CMOS and NMOS. They differ in the keywords for the basic components and the semantics of the wires. In the CMOS language, the construct of composite wires are used to model the three-layered complex of N-diffusion, thinox, and P-substrate. In the NMOS situation, all wires have only one layer. Therefore, the language translators for CMOS and NMOS differ in the way the electrical connectivity is determined. These differences are reflected in the two technology databases.

The UNIX tool *yacc* was used to generate a translator for all languages in the family. Since all technologies share the same relational schema, we only need one syntactic specification for all languages in the family. The translator retrieves information from a technology database and uses it to perform the necessary computation such as determining electrical connectivity, determining sizes of components, and resolving geometric constraints.

3.3 A Database Representation for Chip Design

A chip layout can be specified as a layout program using a layout language. An alternate representation is to organize the chip information as a list of components in the chip.

In Vdd, we model the chip information by a set of relations, each of them representing a component type. There are *transistors*, *wires*, *contacts*, *externs* *interns* calls, and *cells*. Calls refer to the inclusions of subcells in a cell.

Each relation stores information about the components; for example, names of transistors, orientations, sizes, and locations are kept in the *transistors* relation. Each relation is implemented as a B-tree. Secondary indexing by attributes can also be constructed.

The B-tree package is due to Peter Weinberger. He uses the scheme of shadow pages to obtain atomic updates to a B-tree. In case of a soft disk crash, the contents of a B-tree will not be corrupted.

3.4 Storage Management

A tool in the Vdd package needs to retrieve information from the design database. Let's consider the case of the graphics editor in Vdd. It allows the designer to edit a circuit interactively. The designer chooses a cell to edit, therefore, the software needs to display the components of the cell. In addition, the designer may remove, relocate, or add a component. The Vdd editor also allows the designer to verify the circuit against the design rules interactively.

The information in the design database is moved into the main memory on a demand basis. As far as the information contents in the main memory are concerned, a cell can be in one of the three states: STRIPPED, HALF, and FULL. Each cell has its "summary" information stored in the *cell* relation. A cell is STRIPPED, if it has only this record in the main memory. Since the record contains the name of the cell, its bounding box, and status, it is the minimal information to bring into the main memory. This is particularly useful at the beginning of an editing session.

A HALF cell has all its calls and externs in the main memory. When a cell is added into another cell as a call, it is necessary to check if the calls run into a recursion. Therefore, all cells directly or indirectly called by the first cell have to be brought into the main memory. But they only need to be HALF.

A FULL cell has all its components stored in the main memory. In Vdd, we keep the cell being edited FULL. The design-rule check of a cell keeps all its sub cells FULL.

When chips are getting very large, the amount of main memory usable by a design tool becomes crucial. Our storage management strategy tends to keep only a few cells in the main memory. When the main storage runs scarce, Vdd starts to purge the cells that are not critical from the main memory. Each cell is timestamped to reflect its currency. This information is used by the

design-rule checker to ensure that only updated cells are checked. For example, a cell needs to be checked only if it or one of its subcells has been changed after the last check.

Timestamps and the ability to provide quick access to partial information of a cell as well as to swap cell information in and out of the main memory as the need arises allow us to support quick iterations of layout editing and design-rule checking.

4. CONCLUSIONS

In our effort to integrate VLSI design tools, we have found several useful applications of database concepts. Modeling the processing technology by a relational schema and retrieving the information via relational queries makes the tools independent of changes to database contents. As the Vdd system evolved, we had the need to augment the technology database. These changes were made without disturbing the existing tools.

The designer is given two options to represent a chip. It can be represented by a layout language or a design database. Translators are provided to map between the two representations.

The language representation is useful for tools that generate layouts automatically. The database representation is useful if the designer engages in interactive layout design. In this case, quick response time, efficient storage management, and reliable updates to the chip description can be achieved by implementing a design session as a database transaction.

Acknowledgement

We would like to thank the other members of the Vdd design team: Jack Fishburn, Peter Honeyman, and Paul Rubin. Fishburn implemented the interactive design-rule checker and also built several technology databases. Honeyman implemented the interface to Weinberger's B-tree package. Rubin programmed the relational software for the technology database.

REFERENCES

- [Bell] C. G. Bell and A. Newell, *Computer Structures: Readings and Examples*, McGraw-Hill computer science series, McGraw-Hill, New York, 1971.
- [Bose] A. K. Bose, B. R. Chawla, H. K. Gummel, "A VLSI Design System," IEEE 1983 International Symposium on Circuits and Systems Proceedings, pp. 734-739, 1983.
- [Chu] K.-C. Chu, J. P. Fishburn, P. Honeyman, Y. E. Lien, "Vdd - A VLSI Design Database System," Proceedings, 1983 ACM-SIGMOD Database Week, pp. 25-37, May 1983.
- [Fowler] B. R. Fowler, "XYMASK," Bell Laboratories Records, Vol. 47, No. 6, pp. 204-209, July 1969.
- [Johnson] S. C. Johnson, private communication.
- [Katz] R. H. Katz and S. Weiss, "Transaction Management for Design Databases," Technical Report, Computer Science, University of Wisconsin, 1983.
- [Mead] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass., 1980.

Revision Relations
Maintaining Revision History Information

Mark Haynie
Karl Gohl
Amdahl Corp.

ABSTRACT

This paper describes the use and implementation of revision relations. Revision relations allow revision history information to be maintained in a relation. The design presented here has several advantages over similar ideas in both speed and storage requirements.

1. INTRODUCTION

The need for revision history information and versioning control in certain applications is described in [HAYN81]. In that paper, the Relational/Network Hybrid data model describes how several instances of a particular relation definition can exist as separate entities. Instances of relations all contain the same type of information (say, interconnection information in a computer system) but the data is divided on functional boundaries (say, the chip level of that system). This method is fine when storing information about different chip designs but may not be optimal for keeping track of versions of the same chip. Revision two of an integrated circuit will, in all probability, look the same as revision one with a few interconnect changes. Revision relations are proposed to allow many revisions (versions) of a relation to be stored in a single physical relation.

Hypothetical relations described in [STON81, WOOD83] are unsuitable for this task in that they are geared primarily for "what if" changes to a database. They can be used to debug applications on live data without fear of corrupting the database. Used in this way, one usually works with level one hypothetical relations (based on actual relations) or, possibly, level two hypothetical relations (based on a level one hypothetical relation) but, rarely much further. To use this construct for version control each level 1..n of a hypothetical relation set must be applied in order to archive the nth version of the relation.

Two requirements for revision relations are fast retrieval at any revision level and that control constructs in the form of extra attributes defined on a relation be kept at a minimum. The usage pattern of revision relations, unlike hypothetical relations, is that changes are performed at the highest

revision only and queries are performed at any level. This restriction is not strictly enforced, however. Like hypothetical relations, once a tuple has been changed at level n , changes at levels $< n$ cannot affect tuples at levels $\geq n$. A final requirement is the ability to view all revisions as a single composite relation. This may be important to the application that wishes to see the changes between revisions as opposed to the data at a particular revision.

2. REVISION RELATION STRUCTURE

2.1 COMPONENTS

All revisions of a particular relation are stored in a single relation. All tuples of a revision relation have two new column definitions, @revid and @revout. @revid contains the revision number for each tuple at which it became effective. @revout contains the revision number of when the tuple has been obsoleted. @revid and @revout are normally invisible to the user: selecting "all" the columns of a relation in a query will not print these. Data manipulation operations on revision relations will automatically modify the @ columns appropriately and queries will use the @ columns to retrieve only those tuples requested by the user at a certain revision level.

The figure below depicts a personnel database at revision one.

emp			
name	dept	@revid	@revout
Diana	16	1	NULL
Mark	16	1	NULL

Each tuple has a @revid of 1 (the revision at which it was added) and a @revout of NULL (to signify that it is applicable for all further revisions). If new employees are hired during revision 2 of the database the relation may look like:

emp			
name	dept	@revid	@revout
Diana	16	1	NULL
Mark	16	1	NULL
Ken	16	2	NULL
Hanfei	16	2	NULL

2.2 RETRIEVAL

Queries made to the database are qualified with the revision number.

```
select *                                (1)
from emp using <1>;
```

will retrieve only those tuples in effect at the time revision one modifications were made (<Diana,16>, <Mark,16>). [The <1> is a symbolic representation of revision number specification. Actual format in our system is explained later.]

```
select *                                (2)
from emp using <2>;
```

will print the name and dept fields of the emp relation reflecting all modifications up to and including those performed at revision two (<Diana,16>, <Mark,16>, <Ken,16>, <Hanfei,16>). Retrieval operations on revision relations are implemented like view expansion on base relations. Expression (2) is actually modified by the system to

```
select name, dept                        (3)
from emp
where (@revin <= 2) and
      ((@revout > 2) or
       @revout = NULL));
```

2.3 DELETES AND UPDATES

Deletes to revision relations are actually performed by changing the @revout column. The operation of firing employee Mark

```
delete emp using <2>                     (4)
where name = 'Mark';
```

and transferring Diana from department 16 to 12

```
update emp using <2>                     (5)
set dept = 12
where name = 'Diana';
```

during the revision two time frame would cause the composite relation to look like:

emp			
name	dept	@revin	@revout
Diana	16	1	2
Diana	12	2	NULL
Mark	16	1	2
Ken	16	2	NULL
Hanfei	16	2	NULL

Queries on revision one will continue to see the old revision one data, expression (1) will still retrieve (<Diana,16>, <Mark,16>). Expression (2) will still be modified to (3) but now retrieves (<Diana,12>, <Ken,16>, <Hanfei,16>).

The ability to view the composite revision relation is possible by turning off the query modification which normally occurs when a particular revision is referenced. This allows a user to look at only the changes between revisions or all data in all revisions. Applications must be aware of the @revin and @revout columns when using revision relations in this way. To print out all the tuples (the old values) that were changed at revision two, one may code:

```

select *                               (6)
from emp
where @revout = 2;

```

3. IMPLEMENTATION

Revision relations are drastically different from regular relations and therefore the four basic database operations select, insert, delete and update must be enhanced to handle them.

3.1 INTEGRATION WITH THE HYBRID MODEL

The Taco database management system is an implementation of the Hybrid data model [HAYN83b]. In this model, access attributes are the mechanism for determining which of possibly many relational instances are to be used for a particular database operation. Access attributes for a relation are stored in another relation (a "control" relation), which usually contains history information. The access attribute and control relations constructs are used also for differentiating between revisions. Further, revision relations are dynamic relation instances in Taco so many instances of revisioned tables may exist simultaneously.

The following database contains information on chip interconnection (net) information.

control			
pn	rev	owner	netsaa
123	1	harry	@1-1
123	2	harry	@1-2
124	1	george	@2-1
124	2	george	@2-2

@1: nets				@2: nets			
netno	cpname	@revin	@revout	netno	cpname	@revin	@revout
1	blk1.cpl	1	2	1	blk4.cpl	1	NULL
2	blk2.cp2	1	NULL	2	blk9.cp2	2	NULL
3	blk3.cp3	1	NULL	3	blk8.cpl	2	3
4	blk4.cp4	2	NULL	3	blk8.cpl	2	3

Figure 1. Sample Interconnect Database using Revision Dynamic Relations

3.2 DATA DEFINITION

To define a revision dynamic relation one includes the revision keyword in the Taco define statement:

```
define revision table nets(netno(integer),                (7)
                           cpname(char(9)));
```

An instance of a revision table is created the same way as a dynamic table -- the create keyword being inserted into the access attribute column.

```
insert into control(pn, rev, owner, netsaa):             (8)
  <126, 1, 'fred', create>;
```

The create keyword will create revision one of the instance. Deleting all the access attributes to a particular revision relation will remove the instance. The next revision (two in this case) can be created using a built in function. The insert statement

```
insert into control(pn, rev, owner, netsaa):
  select 126, 2, 'fred', nextrev(netsaa)                (9)
  from control
  where <pn,rev> = <126,1>;
```

will take the access attribute of revision one and make a new access attribute with the next revision (via the nextrev built in function) by inserting a new tuple into control. Modification of revision two of the relation will be seen by revisions two and higher -- revision one will continue to see the old data.

3.3 INSERTION

Taking the example database in fig. 1, any references using the access attribute corresponding to <pn,rev> of <123,1> will reference revision one. For example, data can be entered into revision one using normal data manipulation

statements:

```
insert into nets using (select netsaa
                        from control
                        where <pn,rev> = <123,1>) :
( <1, 'blk1.cpl'>, <2, 'blk2.cp2'>, <3, 'blk3.cp3'> );
```

 (10)

The access attribute used to reference the revision table instance holds the revision number as well. That number is inserted into the @revin column of each tuple inserted. The @revout column always is set to null for new tuples. The statement:

```
insert into nets using (select netsaa
                        from control
                        where <pn,rev> = <123,2>) :
<4, 'blk4.cp4'>;
```

 (11)

results in the tuple <4, 'blk4.cp4', 2, null> actually being inserted.

3.4 RETRIEVAL

Retrieval of data from revisions is performed by modifying the query to view only a single revision. The revision number is encoded in the access attribute so a query entered as

```
select *
from nets using (select netsaa
                 from control
                 where <pn,rev> = <123,2>)
where netno > 3;
```

 (12)

is modified to look like:

```
select *
from nets using instance#
where (netno > 3) and
      (@revin <= rev# and
       (@revout > rev# or @revout = NULL));
```

 (13)

Where **instance#** is the dynamic instance number portion of the access attribute returned by the using clause query and **rev#** is the revision number of the access attribute. In our implementation, NULL is represented by a large positive integer when stored in integer columns such as @revout. This means that the clause (@revout > **rev#**) will produce a true value when @revout is NULL, so the clause (@revout = NULL) may be eliminated.

Referencing all revisions of a relation is implemented using a built in function to turn an access attribute of a particular revision into an access attribute of all revisions.

```
select *
from nets using (select allrevs(netsaa)
                 from control
                 where <pn,rev> = <123,1>);
```

 (14)

The allrevs function simply modifies the rev# portion of the access attribute to 0. An additional expression in the where clause must check for this case. The expression (14) is, therefore, modified to be

```
select *
from nets using instance#           (15)
where (netno > 3) and
      ((@revin <= rev# and @revout > rev#) or
       rev# = 0);
```

3.5 DELETION

Deletion of tuples in a revision relation is actually a two step process. The statement

```
delete nets using (select netsaa           (16)
                   from control
                   where <pn,rev> = <123,2>)
where cname = 'blk2.cp2';
```

will be modified as with queries to select only the tuples in effect at the time of revision two. Then an update operation rather than a delete operation is performed to change the @revout column to the value of the revision number of the access attribute used to access the nets relation. The one exception to this is when the @revin and @revout columns are the same -- in which case the tuple is really removed. (When a tuple is added and deleted during the same revision there is no way to access it since the where clause modification performed during queries will never select that tuple. Thus, it might as well be removed from the table.)

3.6 MODIFICATION

An update operation on a revision table translates to an update and an insert. A statement such as

```
update nets using (select netsaa           (17)
                   from control
                   where <pn,rev> = <123,2>)
set cname := 'blk17.22'
where netno = 1;
```

first gets modified such that the revision two is acted upon. For each tuple that is a candidate for modification, the tuple (before modification) is reinserted into the relation but, with the @revin field set to the current revision (two in this case) and the @revout field set to NULL. The original tuple is then modified according to the set clause and the @revout column is set to the current revision number (two in this case). As with deletion, a special case is made for updates to tuples inserted or updated previously at the same revision level. A standard update operation with no modification of the @revin or @revout columns is performed in this case since to do otherwise would prevent the tuple before modification to be seen by the user. (In

standard revision relation retrieval mode, that is. "Allrevs" retrieval mode could see these changes but it was felt the current solution made better sense).

4. OBSERVATIONS -- ALL REVISION PROCESSING

For applications operating on the composite relation ("allrevs" mode) certain guidelines must be followed in order for applications to determine the old and new tuples of an update set. Following an update of a single tuple in a revision relation we have two tuples stored in the composite relation, the data before modification (the @revout column indicates the revision number) and the data after modification (the @revin column contains the same revision level). In order for querying processes to later match up which new tuples belong with which old tuples an unchanging primary key must exist for the relation.

In our first set of employee examples above we assume that **name** is an unchanging primary key. However, if changes to the primary key of revision relations are allowed to occur (if an employee marries and changes his/her name, for instance) then an additional mechanism is needed to match old and new tuples. A tuple number column may be added to the relation for this purpose. A tuple number is unique for each logical tuple in the relation (although in the composite relation there may be several tuples with the same tuple number, all at different revision levels). Taco, however, makes no attempt to maintain these numbers automatically.

REFERENCES

- [ASTR76] Astrahan, M.M., et. al. "System R: Relational Approach to Database Management." ACM Trans. on Database Systems. 1:3, September, 1976, pp. 189-222.
- [GOHL83] Gohl, K.W. DA Architecture, Amdahl Working Document P/N D3 114668, 1983.
- [GOHL82] Gohl, K.W. Reducing Storage Requirements for DA, Amdahl Technical Reference Memorandum, 1982.
- [GRAY81] Gray, J., et. al., The Recovery Manager of the System R Database Computing Surveys 13:2, June 1982.
- [HAYN81] Haynie, M.N. "The Relational/Network Hybrid Data Model." Proc. 18th Design Automation Conf., 1981, pp. 646-652.
- [HAYN83a] Haynie, M.N. "The Relational Data Model for Design Automation

Databases." Proc. 20th Design Automation Conf., 1983.

- [HAYN83b] Haynie, M.N., Taco User's Guide, Amdahl Software Specification P/N 819043-600, 1983

- [STON81] Stonebraker, M., "Hypothetical Data Bases as Views," University of California - Berkeley Memorandum No. UCB/ERL M81/27, 1981.

- [WOOD83] Woodfill, J. and M. Stonebraker, "An Implementation of Hypothetical Relations," University of California - Berkeley Memorandum No. UCB/ERL M83/2, 1983.

Database Management and Computer-Assisted VLSI Fabrication

Randy H. Katz
Computer Science Division
Electrical Engineering and Computer Science Department
University of California, Berkeley
Berkeley, CA 94720

Abstract: We discuss the data management issues for a computerized manufacturing facility. Berkeley's new research facility for integrated circuit fabrication, capable of manufacturing VLSI complexity circuits, is being used as a testbed for computer-assisted manufacturing. Projects include: applied expert systems for fabrication line diagnosis and repair, automated laboratory notebook, real time computer-based machinery control, man-machine interaction in the manufacturing environment, and automatic scheduling of machine use and maintenance times. The INGRES relational database management system is currently being used for the management of some of the fabrication line data.

1. Introduction

As engineers (software and hardware), we typically think of "design" as the activity that turns an idea into an engineering prototype. An equally important activity is manufacturing: taking an object's design specification and reproducibly creating instances of it, usually in large quantity.

In this paper, we describe the data management requirements for the integrated circuit fabrication line. Surprisingly enough, the very machinery that has made the microelectronics revolution possible has yet to become directly controllable by computer! However, the situation is beginning to change. The goal is to improve circuit yield, quality, and throughput for VLSI circuits by improving the control of the process and making it more predictable. The first step is have the plant description available on-line, and to be able to track the state of the line at any point in time.

The information about a manufacturing facility includes machine descriptions (what they are and how to use them), machine histories (usage and maintenance), process descriptions (the set of machine control sequences), inventories, environmental monitoring, and personnel information. An objective is to understand the correlations of the many variables with circuit yield. The interrelationships among the data are complex, and lead to a challenging database design problem.

The organization of the paper is as follows. In the next section, we give a more detailed description of what constitutes the computer-assisted fabrication line. A description of the kinds of data found in this environment is given in section 3. With the description of the manufacturing machinery and processes on-line, more intelligent "adaptive" applications are possible, and one such application for fault diagnosis is described in section 4. Section 5 briefly describes extensions to database systems that should be incorporated in the next generation systems to better support such applications. Section 6 contains the summary.

2. The Computer-Assisted Fabrication Line

The integrated circuit fabrication line is a marvel of technology. The "tooling" of the manufacturing process is a collection of masks containing geometric shapes. The masks specify the patterning of a three dimensional structure on the silicon wafer. Current technology is approaching one micron feature sizes, with research efforts striving for reproducible submicron resolution. Special equipment is used for both mask making and wafer growth and preparation. The wafers are submitted to a complex sequence of chemical and physical processes to transfer the specification of the circuit from the masks to the silicon surface. The control of the process

completion.

The final step is to attempt to learn from the faults. Newly discovered correlations should be recognized by the system. This may involve such things as more frequent maintenance or tighter environmental controls.

5. Implications for Next Generation Databases

For forecasting and for discovering correlations among laboratory variables, the data must be organized for time-series analysis. The notion of time must be embedded in the database system. Much of the analysis is based on pattern recognition techniques, and support for statistical queries will be necessary.

Monitors/trigger mechanisms must become adaptive. Triggering conditions can change over time as new relationships among the variable are discovered. For example, if overdue maintenance is correlated with reduced yield, the trigger that schedules maintenance should be changed to trigger more frequently.

Since the comparison of observed and expected results is frequent, the database should support convenient encodings of the expected results in the form of *simulations*. Thus, some data is stored in the form of programs that can generate the needed data on demand.

These systems will have to be made highly available. An automated factory is expected to be kept busy around the clock, and outages of service will not be tolerated. Much of the environmental monitoring is related to safety, further emphasizing the need for twenty-four hour availability.

6. Summary

Engineering databases are more than design databases, they also include manufacturing data: machine descriptions, inventories, environmental monitoring, process descriptions, and personnel. An accurate description of the process line, the process, and the laboratory environment in order to control the process line, including the ability to adapt to faults or unexpected conditions. The computer-assisted fab line provides an excellent testbed environment for coupling intelligent systems, such as fault diagnosis, with very large databases. These problems are currently under study at UC Berkeley and several other universities.

7. References

- [OSSH83] Ossher, H. L., B. Reid, "Fable: A Programming Language Solution to IC Process Automation Problems," Proc. Sigplan '83 Symp. on Prog. Lang. Issues in Software Systems, San Francisco, CA, (June 1983). Available as SIGPLAN Notices, V 18, N 6, (June 1983).

Engineering Data Management Activities
Within the
IPAD Project

H. R. Johnson

Boeing Computer Services
Seattle, Washington

ABSTRACT

This paper summarizes current research and development activity in engineering data base management systems by the IPAD* Project at The Boeing Company.

1.0 INTRODUCTION

In 1976, NASA awarded The Boeing Company a contract to develop IPAD. The specific goal of IPAD was to increase productivity in the United States aerospace industry through the application of computers to manage engineering data. An Industrial Technical Advisory Board (ITAB) was established to guide the development of IPAD [1]. Members of ITAB represent major manufacturing (aerospace and other) and computer companies. More recently, IPAD has also received funding from the Navy to investigate computer-aided manufacturing.

IPAD has considered applying data base management (DBM) technology to all phases of the product life cycle: design (CAD), manufacturing (CAM), and operations/maintenance. IPAD has also investigated the application of data base management technology to the integration of processes within and between all phases of the life cycle through access to a common data base management facility.

In this connection, IPAD has analyzed engineering design methodologies, identified requirements for integrated, computer-based systems for managing engineering data, and developed software to demonstrate these concepts. Results have included development of the RIM and IPIP data base management systems (DBMSs) and a network facility for multi-host, intertask communication. See [2] for a comprehensive discussion of IPAD objectives and products.

An earlier report 3 published in this journal provided brief descriptions of the RIM and IPIP DBMSs and documented briefly some of the requirements identified for engineering data base management. Consequently, this paper treats these subjects in less detail and considers, as well, current activities which make use of RIM and IPIP.

These include work on the following:

- o Distributed processing (including DBMS file transfer) in a heterogeneous hardware/software (DBMS) environment.
- o Distributed data base management in the same heterogeneous environment.

*IPAD (Integrated Programs for Aerospace-Vehicle Design) development is performed by The Boeing Company under NASA Contract NAS1-17555. IPAD software and documentation may be obtained from the IPAD Program Management Office, The Boeing Company, P.O. Box 24346, Seattle, WA 98124, M/S 73-03.

- o Further geometry applications utilizing IPIP structure (complex object) handling capabilities.
- o A stand-alone geometry management utility supported by the SQL DBMS which incorporates structure processing capabilities currently embedded in IPIP.

2.0 REQUIREMENTS FOR ENGINEERING DATA MANAGEMENT

Early in the IPAD contract, the engineering design process was analyzed, and a number of requirements for engineering data management were identified [3,4,5,6]. The following briefly describes just a few of these requirements.

The system must support definition and manipulation of geometry data and provide for interfacing this data with design drafting and graphic display systems.

Multiple levels and styles of data description are required to provide for differing data requirements and for data independence to support a variety of users in a dynamic environment. Specifically, the network and relational data models are required.

The system must support logical partitioning of a data base into "datasets" which figure in restricting data access, concurrency control, configuration control (including versioning), and data archival.

The system must support distribution of engineering data across a heterogeneous network of computers encompassing multiple DBMSs supporting the data models noted above.

3.0 THE IPAD/RIM DATA BASE MANAGEMENT SYSTEM

The IPAD/RIM (Relational Information Management) DBMS was developed on the IPAD Project to explore relational data base concepts prior to the development of IPIP. RIM has been enhanced by the University of Washington and The Boeing Company in cooperation with ITAB and the IPAD Project.

A RIM data base may be accessed in read mode by multiple users. Access to the data base is restricted to a single user when it is opened for update.

RIM supports a single level of data definition. Relations are organized into schemas. Schema definition may be entered and modified interactively through a menu interface. Rules on data relationships within and between relations may be declared. Rules can be used as constraints, although the user may turn rule checking off and on. RIM supports matrices, vectors, and real data types. It also supports a tolerance capability which supports qualification by user-specified approximation of equality.

RIM offers both algebra-level (including join) and calculus-level (excluding join) data manipulation commands through an interactive interface, along with facilities for formatting retrieved data. RIM supports the calculus-level data manipulation commands for FORTRAN programs via subroutine calls.

RIM is written primarily in FORTRAN 66, but will compile in FORTRAN 77. It is available on several hosts. More than 300 copies of RIM have been supplied to

universities and corporations. RIM is used on a daily basis in many of these organizations.

Development of IPAD/RIM has ceased. However, it has served as the basis of commercial DBMSs (see [7] for example), for mainframes, minis and micros, and has been incorporated in turnkey graphic systems.

4.0 THE IPIP DATA BASE MANAGEMENT SYSTEM

The IPIP (IPAD Information Processor) DBMS is intended to manage engineering data in CAD and CAM environments. To this end, IPIP supports multiple data models, multiple levels of schemas, and concurrent, multiple-user, multi-thread access through multiple application interfaces in a distributed environment. A single-user, single-thread version of IPIP is also available.

Scientific data types and arrays are supported. Composite objects called structures (complex objects), which may consist of multiple tuples from multiple relations, may be declared and manipulated as a single relation to manage geometry and other scientific data. Data may be partitioned logically into datasets to support concurrency control, access restriction, versioning, releasing, and archival procedures. See [8] for a general description of IPIP.

IPAD has developed a general purpose network facility for intertask communication in a heterogeneous, distributed processing environment. The IPAD network has been implemented in accordance with ISO specifications of layered protocol. Access to IPIP and IPIP-managed data is via this network. See [9] for a general description of the network facility. IPIP and the network are written primarily in Pascal. The IPAD network is being replaced in this configuration by recently developed NETEX from Network Systems Corporation.

Some intended IPIP features are not available at this time. For example, the latest version of IPIP does not support versioning, releasing, or archival aspects of datasets nor the MODIFY command for structure-defined relations. Application interfaces are limited to FORTRAN programs. IPIP proper, and CDC and DEC FORTRAN precompilers execute on CDC CYBER series machines (operating under the NOS operating system). DEC VAX 11 FORTRAN programs against the CYBER-resident data base may be submitted via the IPAD network from a VAX 11/780, and then executed on the VAX, calls to IPIP being forwarded to the CYBER for execution and results being returned via the network to the VAX.

CDC has announced IPIP as a class 3 product. Migration of IPIP to other hosts is under consideration.

4.1 Data Architecture

There are three types of IPIP schemas: internal, logical, and mapping. The IPIP internal schema corresponds to the internal schema of the ANSI DBSG [10] and the storage schema of the CODASYL DDLC [11]. IPIP logical schemas correspond to ANSI conceptual and external schemas and also to CODASYL schemas and subschemas. The IPIP mapping schema is used for mapping between schemas of the other two types.

An IPIP data base is described by a single level of internal schemas and one or more (i.e., $n, n \geq 1$) levels of logical schemas mapped to underlying logical and/or internal schemas. An application program may be written against logical schemas at any level.

Logical schemas can be configured in the ANSI and CODASYL tree structure arrangement to provide for centralized definition of a data base through a comprehensive, base-level logical (conceptual) schema. Multiple tree configurations of logical schemas can be coupled by mapping one logical schema to multiple logical schemas or by invoking multiple logical schemas in a single application program or session. This provides for decentralized definition of a data base or of a federation of data bases [12]. This coupling capability may be used for a variety of purposes ranging from integration of existing data bases with minimal change to data definition, to decentralization of data administration over multiple clusters of shared and/or private data.

IPIP provides for preruntime binding of programs and logical schemas. Programs may be bound at runtime regardless of whether underlying schemas have been bound.

4.2 Support for Multiple Data Models

A single Logical Schema Language (LSL) and a Data Manipulation Language (DML) [13,14] support both the relational and network data models. These languages are based on subsets of the 1978 CODASYL DDL [11] and 1979 FORTRAN DML [15] specifications. Included are those CODASYL constructs supporting value-based sets; i.e., those for which relationships are determined by states (value or null) of corresponding attributes in owner and member records. Constructs specific to other set selection criteria are excluded.

The CODASYL INSERTION and RETENTION clauses which specify constraints on association/disassociation of members with/from owners are retained and extended with respect to the handling of null attributes and dovetailed with an IPIP extension (MEMBERSHIP clause), which provides for member records to be put into ownerless 'potential' set occurrences and to be associated automatically by IPIP with an owner when it is created as well as providing for the CODASYL option where owner must exist whenever member does (referential integrity in relational terminology). IPIP extensions include explicit clauses governing IPIP propagation (both from owner to member and member to owner) of record deletion. The CODASYL SOURCE clause which provides for system propagation of attribute state from owner to member is extended to provide for bidirectional propagation.

For more direct support of the relational data model, an optional FOREIGN KEY clause is included in the LSL using syntax which retains the relational flavor of the concept, but parallels set syntax. Clauses were included to specify insertion, retention, and membership options in terms of attribute states. Propagation of record (tuple) deletion and attribute state may be specified relative to foreign keys as well as to sets.

The IPIP DML provides for operations relative to foreign keys (by name) paralleling CODASYL operations relative to sets. A WHERE phrase was incorporated into the IPIP FIND, FETCH, MODIFY, and DELETE commands to support specification of more general conditions for many-record-at-a-time operations. Record name in a DML command may be qualified by a cursor name to support program definition of multiple relations, concurrently, over a single schema-defined relation.

A DATA MODEL clause was included in the LSL to govern which data model dependent constructs (i.e., set, foreign key, or both) may be used in a particular schema. 'RELATION' and 'RECORD' are treated as synonyms in IPIP languages, and may be used interchangeably regardless of data model specified. A DOMAIN clause, which is included in the LSL and ISL (internal schema language) to provide for user-

declaration of data types, may be used with either data model. It is intended that in future releases of IPIP that regardless of data model specified, DML commands across relations (records) may be expressed either in the CODASYL style of referencing schema-declared relationships (e.g., FETCH items of A, items of B VIA (name of) schema-declared set or foreign key relating A and B; where $a_1=b_1, \dots, a_n=b_n$ is the criteria defining that relationship) or in the relational style of inline specification of relationship criteria (e.g., FETCH items of A, items of B WHERE $a_1=b_1, \dots, a_n=b_n$). The full relational join is not available at this time.

The ISL resembles the LSL as nearly as possible. This unified approach to support for the network and relational data models is described more fully in [8,16].

4.3 Datasets and Structures

An IPIP data base is partitioned into datasets. A dataset is declared implicitly on first-user access. Dataset intersections with relations are the units of locking data for read/update. Access by dataset is supported by IPIP indexing (B-tree). IPIP indexing and address conversion structures have been designed to support access to versions of datasets while minimizing physical redundancy of data across versions. Datasets may be used in specifying data to be processed by a prototype facility for transferring data between IPIP and RIM data bases.

Composite objects called structures (complex objects) are supported to manage geometry and other scientific data. A structure is defined in a logical schema to consist of tuples from a tree or network of relations as related by foreign keys. A structure may also be defined in terms of records and sets. A relation/record in one logical schema may be mapped to a structure in another schema. Such a relation/record is said to be structure-defined. A structure is manipulated (retrieval and update) as an entity through operations on a structure-defined relation; the same commands used on nonstructure-defined records.

A user accesses structure-defined data as an entity (e.g., surface, curve, segment). A single-user command may result in IPIP processing of multiple underlying tuples as specified by schema-declared constraints and propagated actions for relations and foreign keys within the structure. On store, IPIP generates values for unique keys when the user does not. IPIP sequences retrieved data according to schema specification for inclusion in the structure-defined record. User productivity is enhanced through support of entities which are natural to his application. Data integrity is enhanced through definition of structure processing in the schema, as opposed to complicated command sequences being embedded in numerous other applications.

The IPIP structure processing facility was developed originally to support management of geometry data. The semantics of a particular geometry representation may be declared via structure declarations. Thus, IPIP is aware of and can maintain the semantics of the geometry. The schema may be revised or extended as appropriate. The structure processing facility is independent of any particular geometry representation. It is also applicable to numerous applications including parts explosion and financial data. See [17,18,19] for more details on structure processing and its application to geometry data.

5.0 IPAD NETWORK EXECUTIVE SYSTEM (INES)

IPAD is currently implementing a network executive system. INES provides an environment on a heterogeneous network of computers in which a task closely

reflects the user's concept of a unit of work. This environment includes a common, uniform, friendly, man-machine interface; common task definition; concurrent task execution control; and design architecture supportive of task or hardware changes. Code will be developed at both layers 6 and 7 of the ISO model (presentation and application layers).

The prototype environment includes a CYBER 835, a VAX 11/780, and an IBM 4341 connected by NSC NETEX software and Hyperchannel hardware. The RIM DBMS is resident on the VAX, IPIP on the CYBER, and SQL on the IBM.

Plans for fiscal year 1984 calls for functionality to be in place for sending and receiving files, transferring relational data bases between the machines and DBMSs noted above, and network configuration control. Follow-on work includes development of a task definition language along with appropriate processors. Users will then be able to define tasks to be performed, but execution details will be transparent to them. Tasks may be performed on machines different from their origin.

6.0 IPAD DISTRIBUTED DATA BASE MANAGEMENT FACILITY (IDF)

Work has begun on a distributed data base management facility, IDF, which provides for data base management in a CAD/CAM environment consisting of a heterogeneous network of computers encompassing multiple DBMSs supporting a variety of data models. Initial prototyping will be conducted in the IPIP/CYBER-SQL/IBM environment described in Section 5.0. Work on IDF addresses the following objectives. See [20,21,22] for more details on IDF.

6.1 Support Requirements of the CAD/CAM User Environment

The system must provide facilities to support traditional engineering requirements such as scientific data types, geometry, datasets, versions, and configuration control.

6.2 Encompass Heterogeneity of Hardware and Software

The typical product life cycle is supported by a heterogeneous mix of computers supporting multiple DBMSs and a variety of data models. A data management facility spanning these sites provides for integration of the applications that support the life cycle.

6.3 Incorporate Existing Enterprises

As time passes, the perception of a computing enterprise changes. What has been viewed as several enterprises, each supported by a centralized data base and an attendant application suite, comes to be viewed as a single, more encompassing enterprise. In this view, the multiple coexisting data bases and application suites together form a basis for the expanded enterprise. Existing data and applications must continue to be supported with minimal change (i.e., data or program conversion). It must be possible to redistribute and /or replicate existing data to better support the expanded enterprise--again, with minimal impact; and it should be possible to write new applications which span what were formerly distinct data bases.

6.4 Support of Relational, Network, and Hierarchical Data Models

Most DBMSs commercially available or in the public domain support either the relational, network, or hierarchical data models. Many CAD/CAM data bases have

been or will be implemented using these products. To support existing enterprises, IDF must support these three data models.

6.5 Provide a High Degree of Site Autonomy

The facility must respond to local users even when remote sites are unavailable, providing access to data on those sites which remain available. This requires that all centralized dependencies be avoided in the distributed environment. Such things as dictionaries, scheduling, and deadlock detection must be implemented in a decentralized manner.

6.6 Provide Transparency of Data Location/Replication

In a dynamic organization, access patterns change over time. Users must be insulated from the need to be aware of where data is located and/or replicated. This provides data administration the freedom to relocate and replicate data to enhance system performance without impacting the user or his programs.

6.7 Provide User-Friendly Interfaces

Ease of use has been and will continue to be paramount in obtaining widespread acceptance of any new technology. Users must be able to invoke IDF facilities in a convenient manner. Aspects of convenience include simplicity of use, familiarity, and uniformity. Simplicity involves easy-to-use interfaces, such as forms or menus, along with help facilities. Such facilities should accommodate novices as well as experienced users. Familiar interfaces require no additional training of users, and support existing applications without modification. Uniformity allows users to work at multiple sites without having to deal with multiple data models. User friendliness is relative to particular user/application mixes. Requirements may vary from user to user of a single system.

6.7.1 Support DBMS-Native Interface

In accordance with the user-friendly aspect of familiarity, IDF must support, with minimal change, the DDL/DML interfaces of those DBMSs which may be linked into an IDF network.

6.7.2 Support Configurable User Interfaces

In accordance with the user-friendly aspects of familiarity and uniformity, IDF must allow user interfaces to be configurable on an individual user basis. It should provide facilities to configure user interfaces from a common data model (DDL/DML), to be used at every site, to multiple data models, to be used at designated sites. Thus one user might, for example, use the SQL DDL/DML at any site, while another user might use the IPIP DDL/DML across the network. Alternatively, user interfaces might be configured so that the DDL/DML available to users might vary from site to site.

6.8 Support Configurable Data Administration

To accommodate various organizational approaches to management, the system should provide facilities for configuring data administration from a centralized to a federated function. Configurability provides for various degrees of local administrative autonomy. This capability complements the facility to incorporate existing enterprises (see Section 6.3). Each existing enterprise has an established data administration function. One attractive approach to data administration for the merged enterprise is a federation of these data

administration functions. Another approach would be to merge these into a single, centralized data administration function.

6.9 Provide an Open System Architecture

A multitude of data base management systems and data models are available. Initially, IDF can incorporate only limited subsets of these. IDF must provide an open system interface to allow future inclusion of additional DBMSs/data models.

6.10 Support Homogeneous Distributed Systems as a Special Case

Special support should be provided in the case where an IDF facility encompasses instances only of a single DBMS. Support of the DDL/DML native to this DBMS is one example of this. In the area of performance, data model translation might be circumvented.

7.0 FURTHER GEOMETRY APPLICATIONS

The IPIP structure processing facility is independent of any particular geometry representation. However, IPAD has developed schemas to model geometric objects using the rational Hermite representation. These schemas support points, segments, and objects (groups of entities). Segments supported by these schemas include lines, conics, rational cubics, and quintics. These schemas are being enhanced to support composite curves, patches, tabulated cylinders, ruled surfaces, and limited engineering drawing layouts.

IPAD has written translators to retrieve geometry stored in an IPIP data base, write it to a file in IGES format; and conversely, to store data from an IGES file to an IPIP data base. Currently, the translators are limited to points, lines, circular arcs and objects, though they are being enhanced to handle additional entities.

8.0 STAND-ALONE GEOMETRY MANAGEMENT UTILITY

IPIP facilities for manipulating geometry data (maintaining owner-member relationships, propagating attribute states, and tuple deletion, etc.) are being packaged in a utility which uses the SQL DBMS. These facilities are hard coded, in that they are driven by a utility-supplied schema. Design of the facility is modular to facilitate its migration to other DBMSs and its extension to support user-supplied data description.

Activities in this area will include comparison of this DBMS-external utility approach with the DBMS-embedded approach taken by IPIP.

REFERENCES

1. Swanson, W. E., "Industry Involvement in IPAD Through the Industry Technical Advisory Board," Proceedings of IPAD National Symposium, NASA Conference Publication 2143, September 1980, pp. 21-26.
2. Miller, R. E., "IPAD Products and Implications for the Future," Proceedings of IPAD National Symposium, NASA Conference Publication 2143, September 1980, pp. 219-234.
3. Johnson, H. R. and Bernhardt, D. L., "Engineering Data Management Activities Within the IPAD Project," IEEE Quarterly Bulletin on Database Engineering, Vol. 5, No. 2, June 1982, pp. 2-8.

4. Meyer, D. D., "Reference Design Process," IPAD Document D6-IPAD-70010-D, March 1977.
5. Southall, J. W., "Integrated Information Processing Requirements," IPAD Document D6-IPAD-70012-D, June 1977.
6. Fisher, T. R., McKenna, E. G., Meyer, D. D., and Schweitzer, J. E., "Manufacturing Data Management Requirements," IPAD Document D6-IPAD-70038-D, December 1981.
7. "BCS RIM--Relational Information Management System User Guide," Version 6.0, RIM Document 70101-03-017, July 1983.
8. Comfort, D. L., Johnson, H. R., and Shull, D. D., "An Engineering Data Management System," Proceedings of IPAD National Symposium, NASA Conference Publication 2143, pp. 145-178, September 1980.
9. Ives, F. M., Kirkwood, D. M., and Tanner, J. G., "Executive and Communication Service to Support the IPAD Environment," Proceedings of IPAD National Symposium, NASA Conference Publication 2143, September 1980, pp. 95-144.
10. Klug, A. and Tsichritzis, D., Editors, "The ANSI/X3/SPARC DBMS Framework," Report of the Study Group on Data Base Management Systems, AFIPS Press, 1977.
11. CODASYL Data Description Language Committee, "Journal of Development," January 1978.
12. Heimbigner, D. and McLeod, D., "A Federated Architecture for Data Base Systems," Proceedings of the National Computer Conference, 1980, pp. 283-289.
13. "Data Base Administration User Guide, IPAD Information Processor (IPIP)," Version 5.0, CYBER/VAX configuration, IPAD Document UM-REL5-200.
14. "Application Programming User Guide, Interfacing and Integrating Application Programs," Version 5.0, CYBER/VAX configuration, IPAD Document UM-REL5-300.
15. CODASYL FORTRAN Data Base Committee, CODASYL FORTRAN Data Base Facility Journal of Development, August 1979.
16. Johnson, H. R., Larson, J. A., and Lawrence, J. D., "Network and Relational Data Modeling in a Common Data Base Architecture Environment," Sperry Univac Research Report TMA00720, Roseville MN, March 1979.
17. Dube, R. P., Herron, G. J., Schweitzer, J. E., and Warkentine, E. R., "An Approach for Management of Geometry Data," Proceedings of IPAD National Symposium, NASA Conference Publication 2143, September 1980, pp. 179-202.
18. Johnson, H. R., Schweitzer, J. E., and Warkentine, E. R., "A DBMS Facility for Handling Structured Engineering Entities," Proceedings of Annual Meeting, SIGMOD 83, Engineering Design Applications, May 1983.
19. Dube, R. P. and Smith, M. R., "Managing Geometric Information with a Data Base Management System," IEEE Computer Graphics and Applications, October 1983, pp. 57-62.
20. Balza, R. M., Beaudet, R. W., and Johnson, H. R., "A Distributed Data Base Management Facility for the CAD/CAM Environment," Proceedings of IPAD Symposium II on Advances in Distributed Data Base for CAD/CAM," April 1984.
21. Johnson, H. R., Beaudet, R. W., Balza, R. M., Baum, L. S., and Nelson, B. W., "IPAD Distributed Data Base Management Facility--Functional Specification," IPAD Document, February 1984.
22. Johnson, H. R., Beaudet, R. W., Balza, R. M., Baum, L. S., and Nelson, B. W., "IPAD Distributed Data Base Management Facility--Architectural Specification," IPAD Document, March 1984.

The map introduces a level of indirection that has the following advantages:

- it implements stable identifiers that are not affected by database reorganization;
- it permits the use of small 2-byte identifiers;
- it permits copying a complex object without any relocation other than adjusting the physical addresses in the map;
- it actually implements an index on non-root identifiers in a very compact way; and
- it permits getting all the tuples in an object without traversing any lists.

2.3 Query Optimization

The map also offers an opportunity to expedite the execution of queries involving the complex object. We have extended the System R optimizer [13] to consider the map as an additional access path when generating a plan for a statement involving a complex object.

One way in which the map can be effective is in evaluating predicates of the form "X = value" where X is the name of an IDENTIFIER column. The value consists of a 10-byte identifier. To locate a tuple that satisfies this type of predicate, the system first retrieves the map identified by the 8-byte identifier of the complex object and then finds the desired map entry using the 2-byte internal identifier. The map entry provides the TID of the desired tuple.

A second way to use the map is for predicates of the form "Y = value" where Y is the name of a COMPONENT OF column. The system first locates the map associated with the complex object, and then, using the 2-byte internal identifier in the value as the index into the map, finds the TID of the parent tuple. It fetches the parent tuple, which in turn contains the internal identifier of the first component tuple which satisfies the "Y = value" predicate. The first component tuple in turn contains the internal identifier of the next component tuple, and so on.

A third way of using the map is for predicates of the form "Z = value" where Z is the name of an INTERNAL REFERENCE column (all references must be to tuples in the same object, otherwise the EXTERNAL REFERENCE column type would be used). The system first locates the map. It then sequentially scans all map entries. For each entry corresponding to the relation of interest, it fetches the tuple to see if the INTERNAL REFERENCE column contains the value shown in the predicate.

2.4 Fetching Complex Objects

Engineering applications tend to require a large volume of data to be moved from the database into the application's data areas for use in the design and analysis work. The usual method to accomplish this with a complex object is to code several nested loops that fetch the root tuple, each of its component tuples, each of their component tuples, and so on, one tuple at a time until the necessary data have been retrieved. Since

the system knows the structure of a complex object, the application should be able to declare which data it wants from an object. Then the system can return all the desired data in one request, which in turn can considerably simplify data area management in the application program; see Figure 2.

```

EXEC SQL DECLARE CC COMPLEX CURSOR FOR
  SELECT CID, CDATA, ->FIRST(INSTANCES), ->FIRST(PATHS)
    FROM CELL;
  SELECT ISUBCELL, IDATA, ->NEXT, ->PREVIOUS
    FROM INSTANCES;
  SELECT PDATA, ->NEXT, ->FIRST(RECTANGLES)
    FROM PATHS;
  SELECT RDATA, ->NEXT
    FROM RECTANGLES;
EXEC SQL END COMPLEX;

EXEC SQL BEGIN DECLARE SECTION;

  DCL 1 CELLSTR BASED(CELLPTR),
    2 CELLID CHAR(10), /* IDENTIFIER */
    2 CDATA ...,
    2 IFIRST FIXED BIN(15), /* Index FIRST INSTANCES */
    2 PFIRST FIXED BIN(15); /* Index FIRST PATHS */
  DCL 1 INSTSTR BASED(INSTPTR),
    2 ISUBCELL CHAR(10), /* Ref id of another CELL */
    2 IDATA ...,
    2 INEXT FIXED BIN(15), /* Index NEXT INSTANCES */
    2 IPREV FIXED BIN(15); /* Index PREV INSTANCES */
  DCL 1 PATHSTR BASED(PATHPTR),
    2 PDATA ...,
    2 RFIRST FIXED BIN(15), /* Index FIRST RECTANGLES */
    2 PNEXT FIXED BIN(15); /* Index NEXT PATHS. */
  DCL 1 RECTSTR BASED(RECTPTR),
    2 RDATA ...,
    2 RNEXT FIXED BIN(15); /* Index NEXT RECTANGLES */

  DCL CHAR_VAR1 CHAR(32760) VARYING; /* Buffer 1 */
  DCL CHAR_VAR2 CHAR(32760) VARYING; /* Buffer 2 */

EXEC SQL END DECLARE SECTION;

EXEC SQL OPEN CC
  TEMPLATES CELLSTR,INSTSTR,PATHSTR,RECTSTR;
IF SQLCODE = 0 THEN DO;
  EXEC SQL FETCH CC INTO :CHAR_VAR1, :CHAR_VAR2;
  IF SQLCODE != 0 THEN CALL HANDLE_ERROR;
END;
EXEC SQL CLOSE CC;

```

Figure 2: Complex Fetch, Declarations and Code.

3. LONG FIELDS

A database system that manages engineering data requires the ability to store items of arbitrary length. This imposes several requirements on the database system:

- Access Methods: For engineering applications, it is often necessary to store long, unformatted items such as raster images or large matrices in fields of arbitrary length. However, if the data is extremely long (e.g., megabytes), retrieving it in one chunk may be impractical or even impossible. This requires the ability to deal with long fields piecewise, in a manner analogous to file operations.

- Recovery: System R uses logging superimposed on a shadow page mechanism to provide recovery [2]. Among other things, the log contains both the old and new record contents after each update. Logging imposes a great space and performance penalty on the use of long fields.
- Secondary Storage Space Management: "Long" fields can vary widely in size making it impractical to allocate space in multiples of one fixed-size unit (page). A large page size wastes too much space at the end of partially full pages. Small pages require many I/O's to read or write a field. Small pages could be allocated contiguously to reduce the number of I/O's but this would lead to fragmentation, requiring frequent compaction of the database. Different size pages must be available for different size fields.

We have implemented a storage system for long fields that satisfies these requirements. This long field manager is called by the data manager component (RDS) of the system and uses the storage manager component (RSS) to manage storage allocation information, thus easily obtaining transaction management and recovery functions. Long field storage is maintained using pools of different size pages; a best-fit algorithm is used to allocate part of a long field value to the appropriate size page or pages. Finally, updates to a long field value are carried out using a shadow page mechanism [9]; this removes the need for logging updates while maintaining the ability for transaction backout.

4. LONG TRANSACTIONS

Transaction management has been extensively studied in the context of classical database applications [3]. In such an environment, a transaction is generally defined as the unit of both consistency and recovery. In a design environment, one needs the notion of transaction for controlling the consistency. But in design applications, the time needed to arrive at a new consistent state of the data is much longer, maybe days or weeks. Therefore, classical use of locks, waits, and deadlock resolution techniques is not suitable.

One proposed approach [6, 11, 7] is to emulate what engineers have done manually in the past; that is, to make copies of the data so that they can work independently of others. Thus, an engineer would CHECK OUT an object from the shared (public) database and store it in a private database, leaving a nonvolatile lock in the shared database. This is the start of a long transaction. When the design has reached a consistent state, the engineer CHECKS IN the changed consistent data, releasing the nonvolatile lock. Each engineering transaction may encompass several such CHECK OUT and CHECK IN actions.

This model is easy to understand, but assumes a very rigid design environment; in particular, it does not support the complex design environment where a hierarchy of designers must complete a complex design involving many design objects by passing incomplete objects back and forth among them in a controlled manner. We have extended the model to allow a designer to CHECK OUT a partial design from another designer and complete the design for the first designer or use it in his own design. These extensions combine and generalize concepts found in both the existing

models of engineering transactions and the model of nested transactions for conventional business applications [12].

For each engineering transaction, we define a "semi-public" database. A transaction may CHECK OUT an object from the public database, or the semi-public database of another transaction. The transactions which check out objects from another transaction's semi-public database become its dependent transactions; that is, they are its children in a transaction hierarchy.

For each CHECK IN, the transaction may commit the object to its own semi-public database, or to the semi-public database of its parent transaction (the public database if the transaction does not have a parent transaction). When a transaction ends, all objects in its semi-public database are committed to the semi-public database of its parent.

Using the semi-public database of a parent transaction, several engineers communicate and share objects that are partially consistent while maintaining the consistency of the public database.

5. VALIDATION

We briefly describe one validation effort for our prototype system. We have implemented a VLSI layout editor [4] which makes extensive use of complex objects. Each VLSI cell is represented by one or more complex object instances in a structure similar to that in Figure 1.

Cells in the database are stored hierarchically, yet should be displayed as flat objects. This is achieved by recursive interpretation; the algorithm draws the rectangles of a cell, and then calls itself to draw the cells which are instantiated in it.

Because of replication, there is a good chance that an object, i.e., a cell, once used, will be accessed again. This justifies the use of an object buffer. The object buffer manager provides access and manipulation capabilities for objects and their components. Modification of a record triggers a transparent, synchronous modification to the database. A simple request for a record or an entire object causes the object buffer manager to return a type code and a pointer to the record or object in main memory. Thus only if the object is not in the buffer will the database be accessed. We believe this methodology to be essential in providing adequate performance.

The use of our database system made the implementation easier because of three factors: the ability to change the schema without loss of data or appreciable down-time of the database system, the availability of an ad hoc query language which can be used as sophisticated peek and poke commands to evaluate the results of transactions, and the ability to back-out a transaction when testing a piece of code.

In addition, several aspects of complex objects have proved to be extremely valuable in the database design for VLSI data management. The hierarchical structure of complex objects supports well the inherent

hierarchical structure of the several records that describe the components of a VLSI object, the availability of identifiers makes it easier to maintain the relationships between tables, and the inherent integrity constraints helped to simplify the programming task.

6. FUTURE WORK

The results we have seen so far are promising. The approach provides the user with valuable functions. It should also improve the performance because of

- special access paths for complex objects,
- special long field management, and
- high level support for long transactions.

But we believe that a greater improvement in performance should come from the fact that, in an engineering environment, much of the work is done on private data. This suggests the development of engineering workstations that would support the private processes and provide full relational capability. However, because the size of the workstation database is smaller, and because capabilities like full multi-user support, authorization, and extensive transaction recovery are not necessary in a private environment (in communication with the central, shared database), one can expect to be able to speed up the processing substantially.

Our current efforts are focussed in three areas: implementing communication between the workstation databases and the public database; developing and implementing the protocols for CHECK OUT and CHECK IN, including nonvolatile locks on the public database, and nested environments for maintaining semi-public databases on the public database system; and developing a single-user workstation database system.

ACKNOWLEDGEMENTS

Gary Hallmark implemented the VLSI layout editor mentioned in the section on validation, and in the process helped us test major sections of the database system.

REFERENCES

- [1] Chamberlin, D. D., et. al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control," IBM J. Res. Dev. 20, 6 (November 1976), pp. 560-575.
- [2] Gray, J., et. al., "The Recovery Manager of the System R Database Manager," ACM Computing Surveys 13, 2 (June 1981), pp. 223-242.
- [3] Gray, J., "The Transaction Concept: Virtues and Limitations," Proc. 7th Intl. Conf. on Very Large Data Bases (ACM), September 1981, pp. 144-154.

- [4] Hallmark, G., and Lorie, R., "Towards VLSI Design Systems Using Relational Databases," Proc. Spring Comcon 84 (IEEE), February 1984.
- [5] Haskin, R., and Lorie, R., "Using a Relational Database System for Circuit Design," Database Engineering 5, 2 (June 1982), pp. 10-14.
- [6] Haskin, R., and Lorie, R., "On Extending the Functions of a Relational Database System," Proc. Intl. Conf. on Management of Data (ACM), June 1982, pp. 207-212.
- [7] Katz, R., and Weiss, S., "Transaction Management for Design Databases," Working Paper, Computer Sciences Dept., Univ. of Wisconsin, Madison, Wisconsin, 1983.
- [8] Kim, W., et. al., "Nested Transactions for Engineering Design Databases," IBM Research Report RJ 3934, IBM Research Laboratory, San Jose, California, June 1983.
- [9] Lorie, R., "Physical Integrity in a Large Segmented Database," ACM Trans. on Database Systems 2, 1 (March 1977), pp. 91-104.
- [10] Lorie, R., "A Project on Design Systems," Database Engineering 4, 1 (September 1981), pp. 5-9.
- [11] Lorie, R., and Plouffe, W., "Complex Objects and Their Use in Design Transactions," Proc. Annual Meeting - Database Week: Engineering Design Applications (IEEE), May 1983, pp. 115-121.
- [12] Moss, J. E., "Nested Transactions and Reliable Distributed Computing," Proc. 2nd Symp. on Reliability of Distributed Software and Database Systems (IEEE), October 1982, pp. 33-39.
- [13] Selinger, P. G., et al., "Access Path Selection in a Relational Database System," Proc. Intl. Conf. on Management of Data (ACM), May 1979, pp. 23-34.

USING A RELATIONAL DATABASE MANAGEMENT SYSTEM
FOR COMPUTER AIDED DESIGN DATA - AN UPDATE

by

Michael Stonebraker
Antonin Guttman

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
UNIVERSITY OF CALIFORNIA
BERKELEY, CA.

I INTRODUCTION

Two years ago [GUTT82] we reported on the result of an experiment comparing the performance of a special purpose CAD editor, KIC [KELL81] with a general purpose data base system, INGRES [STON80] on a data base consisting of a VLSI circuit. KIC dramatically outperformed INGRES, and we indicated some of the factors influencing the outcome. These included the inability of a general purpose data base system to accomplish efficient two dimensional search, the lack of a transitive closure command to efficiently expand design trees and the absence of support for geometric constructs as primitive objects in the data base.

In this paper we sketch a collection of solutions to these problems that we have been investigating. These include a new multi-dimensional access method appropriate for spatial data, the use of abstract data types (ADTs) in a DBMS, the addition of commands in the query language as a new data type, and syntactic and algorithmic solutions to expressing and solving transitive closure queries. We briefly summarize our research on these topics in the remainder of this paper.

II R-TREES

We have devised an index structure called an R-tree which allows efficient access to spatial data according to its location [GUTT84a]. Leaf nodes in an R-tree contain index entries, each consisting of a pointer to a spatial object and a rectangle that covers it. Higher nodes contain similar entries, with pointers to lower nodes and rectangles covering those in the lower nodes. This hierarchy of covering rectangles is built and maintained dynamically in a manner similar to a B+tree.

To search for all data overlapping a given rectangle, we examine the root node to find which entries have rectangles overlapping the search area. The corresponding subtrees can have data in the search area, therefore we apply the search algorithm recursively to each one. In this way we find all qualifying data but avoid searching parts of the tree corresponding to objects that are far from the search area.

This research was sponsored by the National Science Foundation Grant ECS-8300465, by the Air Force Office of Scientific Research Grant AFOSR-83-0254 and by a Grant from ESL, Inc.

R-trees can be built for any number of dimensions, and in addition they are useful for overlapping objects of non-zero size, a characteristic not shared by most multi-dimensional indexing schemes, for example quad trees [FINK74], k-d trees [BENT75], and K-D-B trees [ROBI81].

We have implemented R-trees, and in spatial search tests using VLSI data, only about 150 usec. of CPU time was required per qualifying item. This indicates that the structure effectively restricts processing to qualifying or near-qualifying data.

III ABSTRACT DATA TYPES

We have suggested allowing new types of columns to be added to a data base system and new operators on these columns to be defined. Basically, a knowledgeable user must write a collection of procedures which will be called by the DBMS as necessary. For example, a user could define a polygon ADT and create the following POLYGON relation

```
create POLYGON (pid = i4, p-desc = polygon)
```

Then an overlap operator (!!) could be defined for the new type, and a user could find the polygons overlapping the unit square as follows:

```
retrieve (POLYGON.all) where POLYGON.p-desc !! "0,0,1,1"
```

Support for user defined types and new operators has been constructed in about 2500 lines of code for the INGRES relational data base system. Implementation details are addressed in [FOGG82, ONG82], and ADTs execute with a modest performance degradation [FOGG82]. Initial suggestions concerning how to integrate new operators into query processing heuristics and access methods are contained in [STON83, ONG83]. We are now attempting to cleanly support new operators throughout the query execution engine.

IV QUEL AS AN AN ABSTRACT DATA TYPE

There has been substantial discussion concerning data base support for complex objects. It is possible to support them as ADTs along the lines discussed in the previous section, and this position is advanced in [STON83]. This approach is conceptually clean because no facilities peculiar to CAD data are required. However, it has the disadvantage that one cannot easily "open up" an object and examine its component objects. A second possibility is to extend a relational data base system with specific facilities for complex objects. This is the approach taken in [LORI83]. It has the advantage that component objects can be addressed but requires special-purpose services from a DBMS.

In this section we propose a third approach which may offer the good features of each of the above proposals. It involves supporting commands in the query language as a data type in a DBMS. In our environment this means that a column of a relation can have values which are one (or more) commands in the data manipulation language QUEL. We explain our proposal using the following relations:

```
OBJECT (Oid, o-desc)  
LINE (Lid, l-desc)  
TEXT (Tid, t-desc)  
POLYGON (Pid, p-desc)
```

Suppose a complex object is composed of text, lines and polygons. For each such component object, a tuple would be inserted into the LINE, TEXT or POLYGON relation. For example:

```

append to LINE (Lid = 22,
                description = "(0,0) (14,28)")
append to POLYGON (Pid = 44,
                  description = "(1,10) (14,22) (6,19) (12,22)")

```

Then, the description field in OBJECT would be of type QUEL and contain queries to assemble the pieces of any given object from the other relations. For example, the following query would make object 6 be composed of line 22 and polygon 44.

```

append to OBJECT(
  oid = 6,
  o-desc = "retrieve (LINE.all) where LINE.id = 22
           retrieve (POLYGON.all) where pid = 44")

```

We have proposed extensions to QUEL which allow the components of an object to be addressed. For example, one could retrieve all the line descriptions making up object 6 which were of length greater than 10 as follows:

```

range of O is OBJECT
retrieve (O.o-desc.l-desc) where
  length (O.o-desc.l-desc) > 10

```

This notation has many points in common with the data manipulation language GEM [ZANI83], and allows one to conveniently discuss subsets of components of complex objects. In addition we can support clean sharing of lines, text and polygons among multiple composite objects by having the same query in the description of more than one object, a feature lacking in the proposal of [LORI83].

Materializing an object from the OBJECT relation will be slow since it involves executing several QUEL queries. Hence it is clearly desirable to precompute the value of frequently used objects and store the actual result in the OBJECT description field. We are investigating how to efficiently precompute values for complex objects represented by commands in the query language.

V TRANSITIVE CLOSURE OPERATIONS

In our earlier paper [GUTT82] we suggested syntax for adding transitive closure capability to INGRES. A * operator added to an *append* command indicates that the operation is logically repeated as long as new tuples are generated. For example, suppose the structure of a hierarchical VLSI circuit design is represented by tuples in a CELL-REF relation. Each tuple stands for the use of one circuit cell as a component in another:

```
CELL-REF (parent-cell, child-cell, location)
```

Then all cells used in the entire design for CKT-A can be collected by

```

retrieve into TREE (cell=CKT-A)
range of T is TREE
range of C is CELL-REF
append* to TREE (cell=C.child-cell)
  where C.parent-cell=T.cell

```

The * operator can be applied to *retrieve into*, *delete* and *replace* with a similar meaning.

We have added *append** to INGRES and have tested our implementation by expanding VLSI design trees [GUTT84b]. Tree expansion was faster using a depth-first

algorithm in our implementation, because we could use the stack to store TREE tuples currently undergoing processing, whereas during breadth-first expansion the current level of TREE was stored in a temporary relation. The depth-first method also used much less buffer space, because our VLSI trees were much wider than they were high, as is typical.

If a tree being expanded by *append** contains duplicate subtrees, time can be saved by eliminating duplicate tuples in order to avoid processing the redundant subtrees. We tested a variety of duplicate tuple elimination methods, and found that detecting duplicates in the entire tree or within one level is expensive but sometimes worthwhile. Eliminating duplicates on a vertical path from the root to a leaf is necessary for correct processing of some queries. The cost is negligible for the depth-first implementation, which is another advantage of this algorithm.

REFERENCES

- [BENT75] Bentley, J. L., "Multidimensional Binary Search Trees Used for Associative Searching", *Communications of the ACM* 18, 9 (September 1975), 509-517.
- [FINK74] Finkel, R. A. and J. L. Bentley, "Quad Trees - A Data Structure for Retrieval on Composite Keys", *Acta Informatica* 4, (1974), 1-9.
- [FOGG82] Fogg, D., "Implementation of Domain Abstraction in the Relational Database System, INGRES", Masters Report, EECS Dept, University of California, Berkeley, Sept. 1982.
- [GUTT82] Guttman, A. and M. Stonebraker, "Using a Relational Database Management System for Computer Aided Design Data", *Data Base Engineering*, 5, 2, June 1982.
- [GUTT84a] Guttman, A., "R-Trees: A Dynamic Index Structure for Spatial Searching", submitted for publication.
- [GUTT84b] Guttman, A., *New Features for a Relational Database System to Support Computer Aided Design*, Ph.D. thesis, University of California, Berkeley, in preparation.
- [HASK82] Haskins, R. and R. Lorie, "On Extending the Functions of a Relational Database System," *Proc. 1982 ACM-SIGMOD Conference on Management of Data*, Orlando, Fl, June 1982.
- [KELL81] Keller, K., "KIC, A Graphics Editor for Integrated Circuits", Masters thesis, Dept. of EECS, University of California, Berkeley, June 1981.
- [LORI83] Lorie, R. and W. Plouffe, "Complex Objects and Their Use in Design Transactions," *Proc. Engineering Design Applications of ACM-IEEE Data Base Week*, San Jose, Ca., May 1983.
- [ONG82] Ong, J., "The Design and Implementation of Abstract Data Types in the Relational Database System, INGRES," Masters Report, EECS Dept, University of California, Berkeley, Sept. 1980.
- [ONG83] Ong, J., et. al., "Implementation of Data Abstraction in the Relational Database System INGRES," to appear in *SIGMOD Record*.

- [ROBI81] Robinson, J. T., "The K-D-B Tree: A Search Structure for Large Multidimensional Dynamic Indexes", *ACM-SIGMOD Conference Proc.*, April 1981, 10-18.
- [STON76] Stonebraker, M. et al., "The Design and Implementation of INGRES," *ACM Transactions on Database Systems* 2, 3, September 1976.
- [STON83] Stonebraker, M. et. al., "Application of Abstract Data Types and Abstract Indices to CAD Databases," *Proc. Engineering Design Applications of ACM-IEEE Database Week*, San Jose, Ca., May 1983.
- [ZANI83] Zaniola, C., "The Database Language GEM," *Proc. 1983 ACM-SIGMOD Conference on Management of Data*, San Jose, Ca., May 1983.

Relational and Entity-Relationship Model Databases and VLSI Design

Marianne Winslett Wilkins and Gio Wiederhold
Stanford University, Computer Science Dept.

Abstract. Databases have a number of advantages over specialized design files for use in the VLSI design process. Balanced against these advantages is the presumed extra run-time cost of accessing the database. In particular, ordinary relational databases appear to be too slow to be used on-line by engineers in the design process [Eastman 80, Haynie 81, Sidle 80, Stonebraker 82].

The requirements for design access include the retrieval of major units of data at one time, implying joins of many attributes with their libraries, and projections to obtain the relevant attributes for a given study. Previously, in order to test if databases could achieve acceptable performance at all, we experimented using a CODASYL database system [Beetem 82, Wiederhold 82]. This appeared to provide adequate performance, but at a high cost of software maintenance and lack of flexibility due to CODASYL limitations.

Our hypothesis is that the relational data model per se is not the crucial factor in performance; rather, the internal access mechanisms of the database significantly affect performance. To test this hypothesis, we ran experiments using the Cypress database system in the Cedar environment at Xerox Palo Alto Research Center. The program used is a design macro expander, more fully described in Section 3. The tests show that colocating pointers to all relations that a VLSI component appears in will reduce CPU time spent in the main database access routines by 25%. However, the largest component of execution time is the time spent to write the expansion results to a file.

1. Motivation

Database systems provide a number of attractive features for use in VLSI design efforts. For example, database systems usually automatically provide features for data sharing, concurrency control, and automatic crash recovery. Database systems are often geared to handle the large amounts of data common in VLSI applications. The use of a database system has the potential to free its users from routine data management tasks.

On the other hand, a general-purpose database system may complicate the design task through cumbersome user interfaces, rigid schemas and access paths, sluggish response time, and lack of support for long transactions. Researchers have considered the suitability of commercially available databases [Beetem 82, Sidle 80, Wiederhold 82, Zintl 81] and the appropriateness of various data models [Haskin 82, Katz 82, Lorie 81, Stonebraker 82, 83] for good performance in the VLSI design environment. Relational databases typically do not suffer from rigid schemas and access paths and cumbersome interfaces; in this paper, we address the issue of relational database response time in a VLSI application.

We believe that it is not so much the higher-level data model as the underlying implementation structures that determine the suitability of a database system for use in a design application. We also believe strongly that the implementation structure of the database system can be isolated from its client interface [Steel 75]. With effective mappings from the interface to the internal structure, one can provide a design tool that combines clarity and performance. Through use of the Cypress database system at Xerox Palo Alto Research Center (PARC) [Brown 81, Cattell 83], we were able to measure the differences in performance in VLSI macro expansion resulting from use of two different internal access structures: a purely relational data model with internal indexing via B-trees, and an entity-relationship data model with internal B-trees and colocated links to all tuples associated with each entity. The internal access structures of the entity-relationship model database can be implemented for purely relational databases that allow typed attributes.

2. The Experiment Environment

The experiments were conducted in a "workplace of the future," the Computer Science Laboratory (CSL) of Xerox PARC, using Dorados [Clark 81, Lampson 80, 81] interconnected by a 3 megabit Experimental Ethernet [Metcalf 76]. The Dorados run Cedar [Teitelman 84], an experimental programming environment developed

at Xerox PARC. The different components of this environment are discussed in more detail below.

2-1. The Dorado

The Dorado is a high-performance personal computer developed at Xerox PARC. Dorados have been the hardware of choice at PARC for building prototype systems over the last several years. [Pier 83] gives a quick summary of Dorado features:

...[The Dorado is a 16-bit machine] designed to be used by a single (expert) user running multiple cooperating processes in an integrated programming environment. It has a microprogrammed processor with a 60 nanosecond microinstruction cycle time, a high-speed cache, memory map, large main memory, an instruction fetch/decode unit, and high-resolution monochrome and color displays. The processor is shared among priority-ordered microcoded tasks, performing microcode context switches on demand with no overhead. The memory subsystem is controlled by a seven-stage pipeline. It can deliver a peak main-storage bandwidth of 530 million bits per second ... The IFU is implemented with a six-stage pipeline, and under favorable conditions can deliver instructions at a peak rate of 16 million instructions per second. The machine is implemented using standard ECL 10K technology.

From this description of existing facilities, we extrapolate and say that the Dorado presents a model for the engineering workstations that will become common in industry.

The Dorado on which the VLSI macro expander runs is equipped with an 80 megabyte disk and a mouse.

2-2. The Software Environment: Cedar

Cedar is an integrated computing environment developed at PARC over the last five years. The system revolves around a single language—Cedar Mesa [Mitchell 79]—and includes facilities for document typesetting, color graphics, performance monitoring, remote and local debugging, automatic garbage collection, networking, print servers, file servers, mail servers, tape servers, and many other features. Cedar Mesa is an object-oriented, modular, strongly typed language with a PASCAL-like syntax. The experiments were run under Cedar 5.1 of February 1984.

2-3. The Database System: Cypress

Cypress is an entity-relationship-datum model database [Chen 76, Cattell 83] system designed to run under Cedar. From the user's viewpoint, a Cypress database consists of a set of domains, a relation schema, sets of entities in the defined domains, relations representing properties of those entities, and other relations among entities and string-, integer-, boolean-, and date-valued fields. Cypress allows the user to define a hierarchy of

entity types and subtypes. Stored Cypress values are not restricted to conventional underlying data types, and may be extended to include large pieces of text and graphical data.

For the purpose of this experiment, the most important Cypress feature is that a Cypress database without user-defined domains and entities is a purely relational database using B-trees as its only indexing mechanism. Thus, Cypress allows a controlled comparison of the performance of a purely relational implementation and an entity-relationship database implementation. We are able to measure the effect of the entity access method while keeping other internal and external factors constant—an unusual opportunity.

The programming language interface to Cypress allows for tuple-at-a-time access. Access path selection is done at each database call; if the access path were chosen at the time of initial result-query definition, as in System R [Astrahan 76] and its commercial relative SQL-DS, we would expect a substantial improvement in the execution times given below.

2-4. The Remote File Server: Alpine

Alpine is a general-purpose transactional file server package developed at Xerox PARC. CSL uses one Alpine server, equipped with several large disks. The server is heavily used for mail storage and as a repository for large files.

2-5. The Measurement Tool: The Cedar Spy

We used the Cedar Spy to monitor the performance of the VLSI programs. With the Spy, a user can examine execution parameters for both specific routines and programs and in the system as a whole. The user may watch CPU usage or one of five other interesting parameters within selected sections of code. When turned on, the Spy operates by keeping track of the state of all interesting programs and routines. After turning off the Spy, the user may obtain a statistical summary of the monitored information at a specified degree of precision.

3. The VLSI Macro Expander and Data

The main program used in these experiments is a Cedar version of a hierarchical VLSI macro expander originally written in PASCAL [Payne 80] for use on a DEC-10. The PASCAL macro expander and some associated FORTRAN routines were used by [Bectem 82] to evaluate the performance in a VLSI design application of a commercially available network database, DBMS-20. The macro expander has been further developed [?] and is now a commercial offering from Silvar-Lisco. For the current experiment, this PASCAL program was translated to Cedar Mesa, with a substantial

number of changes made for the new environment.

The macro expander takes a given hierarchical design component name—the ALU for example—and produces a description of that component in terms of lower-level components, such as transistors. The user controls the level of detail in the expansion by telling the macro expander which hierarchical levels to expand.

In the Cedar environment, the macro expander runs on the user's local Dorado and uses a local version of the Cypress database package to access the VLSI design macros. Due to the large size of these macros, they are kept on a remote file server running Alpine. This setup is appropriate for a large design project, where engineers have their own personal machines, and large files of general interest are kept in a central location to facilitate sharing and insure consistency. However, there is no programming requirement that the VLSI macros be kept on a remote disk; the physical location of the data is of interest only to the database access routines, and not to any higher level code.

The file resulting from the expansion is stored on the local disk. Again, for a larger result file the expansion could be kept on the remote file server with no program changes. We store the result file as a flat file with no special index structures. While not the recommended method for an integrated environment, this corresponds to current-day practice where the design engineer is confronted with a set of incompatible design tools for use at different phases of the design process [CADTEC 83].

The most interesting feature of the macro expander is the inclusion of an additional level of indirection to allow for the explicit storage of multiple representations of the same object. For example, the database may contain earlier and later versions of the same component. In a VLSI application, one expects to store multiple, logically equivalent representations of the same object, such as sticks, geometric layouts, and the more conventional component-pin-net representations.

For data, this experiment uses a hierarchical description of a PDP11 CPU [Slutz 79]. These macros take up approximately two megabytes of storage, though not all of the database relations are used in the expansion process. We expand the conventional component-pin-net representation of the PDP11, as opposed to its geometric representation.

Table 1. Expansion sizes

ALU	components	193
	nets	295
	pins	1026
CPU	components	504
	nets	1238
	pins	4183

4. Experiments and Results

We will consider the effects of varying two basic parameters: first, whether the on-going expansion is kept in main memory or written out to a temporary file; and second, whether the internal access mechanism is B-trees alone or a combination of B-trees and other pointers.

4-1. Temporary Files

The intermediate results of a VLSI expansion require a great deal of storage space. In the absence of virtual memory, or in a small virtual address space, it will be necessary to keep intermediate results in a temporary file. This version of the macro expander creates a temporary file during each expansion pass at a given level of the design hierarchy, then reads that file on the next expansion pass. Tables 2 and 3 give a summary of CPU time requirements for complete expansions of the PDP11 ALU and CPU. As these tables show, the reading and writing of temporary files is very expensive: it takes 80% more CPU time to expand the PDP11 CPU with temporary files than to do an in-core expansion.

Table 2. Expansion CPU Times for the PDP11 ALU

Entities	Temporary Files	CPU Seconds
y	y	34.2
n	y	39.7
y	n	27.4
n	n	33.8

Table 3. Expansion CPU Times for the PDP11 CPU

Entities	Temporary Files	CPU Seconds
y	y	265.2
n	y	281.2
y	n	141.1
n	n	153.1

In fact, writing out the results of an expansion is always a very expensive operation. According to Table 4, over half the time of all types of expansions is spent in writing the results to disk. A considerable savings would result if the in-memory structures could be put to direct use, as is possible in an integrated system like Cedar.

Table 4. Sample of Where Execution Time Is Spent For a PDP11 CPU Expansion

Entities	Temp. Files	Writing Files		Main Database Access Routine	
		Sec.	%	Sec.	%
y	y	147.0	55.2	42.6	16.0
n	y	145.5	51.9	53.8	19.2
y	n	84.4	60.9	42.4	30.6
n	n	83.0	55.0	54.3	35.9

4-2. Entities

We examine the effects of two different access structures: a purely relational B-tree approach, and an entity-relationship method. In the latter approach, all VLSI components are declared as Cypress entities. This might be regarded as a first step along the lines of recent proposals for semantic additions to the relational model for VLSI design [Johnson 83, Lorie 83, Stonebraker 83]. Internally, the declaration of an entity corresponds to a B-tree lookup that returns an object of type *Entity*. Internally, this object corresponds to a database record that includes pointers to the first tuple referencing that entity, for each relation in which that entity might possibly appear. Within a given relation, all tuples referencing the same entity are chained together. Therefore, when using entities only one B-tree index lookup is required for referencing any or all of the relations in which an entity appears. Index lookups are particularly expensive in Cypress due to a small page size, so minimization of their occurrence should substantially decrease the time spent in database calls.

The techniques described above are similar to ones used in System R's RSS [Astrahan 76], the major model for Cypress's storage level implementation. An additional feature of Cypress, not yet implemented, that would speed database access, is the provision for the collocation of tuples referencing entities with the records defining those entities. This is also an unimplemented feature of the CODASYL DBMS-20 used in earlier experiments [Bectem 82].

Tables 2 and 3, for ALU and CPU expansions respectively, show that the use of entities gives a noticeable decrease in running time. Table 4 shows that for the CPU expansion, most of the CPU savings can be accounted for by the main database access routine in the macro expander, where the great majority of database calls are made.

The use of entities is not without its cost, however. Getting the print name of an entity requires a Cypress call which is not always balanced by a corresponding reduction in tuple access time. The running time of

the macro expander could be improved by only having entity values in those domains that are used to look up particular tuples. For example, the database treats nets as entities, though in our tests database tuples are almost never looked up via net name. Thus the macro expander incurs an extra cost in looking up the net name, with no corresponding savings in tuple access. The cost of using entities can be seen in Table 4, where approximately 1.5 seconds more are required to write files when entities are being used. This is a small penalty, however, if there is any chance that another application might need to make use of nets as entities.

Table 5. Sample Wait Times For a PDP11 CPU Expansion Without Temporary Files Or Entities

Type Of Wait	Seconds
Condition variable	93.7
Page fault	19.8
Preempted	14.3
Other	3.7
Total	131.5
Total running time = wait time + CPU time:	
286.7 seconds	

5. Summary and Conclusion

Databases have a number of advantages over specialized design files for use in the VLSI design process. Balanced against these advantages is the presumed extra run-time cost of accessing the database. In particular, ordinary relational databases appear to be too slow to be used on-line by engineers in the design process.

Our hypothesis is that the relational data model per se is not the crucial factor in performance; rather, the internal access mechanisms of the database significantly affect performance. To test this hypothesis, we ran experiments using the Cypress database system and a design macro expander in the Cedar environment at Xerox Palo Alto Research Center.

We found that the use of temporary files, as opposed to doing an in-core VLSI macro expansion, imposes an 80% overhead for our larger expansions. In all expansions, the expense of writing result files to disk accounts for over half of the CPU time, with an additional third of the execution time being spent in the main database access routines.

The use of entities reduces expansion time by an amount that is independent of whether temporary files are being used. The use of entities gives an 8% savings in CPU time for an expansion of a PDP11 CPU without temporary files. If we restrict our attention to the

macro expander routine that is responsible for most of the database calls, we find that the use of entities gives an approximate 25% savings in CPU time.

We conclude that the choice of internal indexing structure has significant repercussions for the performance of relational and entity-relationship databases in VLSI design applications. However, the database is not necessarily the weakest link in a design expansion; in our case, the CPU demands of our database system were overshadowed by the costs of writing the expansion results out to a result file.

6. Acknowledgements

Our appreciation goes to Xerox Palo Alto Research Center for the use of their equipment; to Rick Cattell for his help with database problems; to John Maxwell for help with the Cedar Spy; and to Dan Swinehart for general aid and assistance. This work was supported in part by contract N00039-82-G-0250 (the Knowledge Base Management Systems Project, Prof. Gio Wiederhold, Principal Investigator) from the Defense Advanced Research Projects Agency of the United States Department of Defense, and by an AT&T Bell Laboratories Doctoral Fellowship. The views and conclusions contained in this document are those of the authors and should not be interpreted as representative of the official policies of DARPA or the US Government.

7. Bibliography

- [Astrahan 76] M. M. Astrahan, *et al.*, "System R: Relational Approach to Database Management," *Trans. on Database Systems*, 1:2, ACM, June 1976.
- [Beetem 82] A. Beetem, J. Milton, and G. Wiederhold, "Performance of Database Management Systems in VLSI Design", *Database Engineering*, 5:2, June 1982.
- [Brown 81] M. Brown, R. G. G. Cattell, and N. Suzuki, "The Cedar Database Management System: A Preliminary Report", *Proceedings ACM SIGMOD Conference 1981*, Ann Arbor, 1981.
- [CADTEC 83] "Series 8000 Design System", Technical Introduction, CADTEC Corporation, April 1983.
- [Cattell 83] R. G. G. Cattell, "Design and Implementation of a Relationship-Entity-Datum Data Model", Technical Report CSL-83-4, Xerox Palo Alto Research Center, May 1983.
- [Chen 76] P. Chen, "The Entity-Relationship Model—Towards a Unified View of Data", *ACM Transactions on Database Systems*, 1:1, January 1976.
- [Clark 1981] D. W. Clark, B. W. Lampson, and K. A. Pier, "The Memory System of a High-Performance Personal Computer", *IEEE Transactions On Computers*, 30:10, October 1981. Also Technical Report CSL-81-1, Xerox Palo Alto Research Center, January 1981.
- [Eastman 80] C. Eastman, "Systems Facilities for CAD Databases", *Proceedings of the 17th Design Automation Conference*, Minneapolis, 1980.
- [Haskin 82] R. Haskin and R. Lorie, "On Extending the Functions of a Relational Database System" *Proceedings of the ACM SIGMOD Conference on Management of Data*, Orlando, June 1982.
- [Haynie 81] M. Haynie, "The Relational/Network Hybrid Data Model for Design Automation Databases", *Proceedings of the 18th Design Automation Conference*, Nashville, 1981.
- [Johnson 83] H. R. Johnson, J. E. Schweitzer, and E. R. Warkentine, "A DBMS Facility for Handling Structured Engineering Entities", *Proceedings of Database Week 1983*, San Jose, May 1983.
- [Katz 81] R. H. Katz, "A Database Approach for Managing VLSI Design Data", *Proceedings of the 19th Design Automation Conference*, Las Vegas, June 1982.
- [Lampson 80] B. W. Lampson and K. A. Pier. "A Processor for a High-Performance Personal Computer", *Proceedings of the 7th International Symposium on Computer Architecture*, La Baule, May 1980. Also in Technical Report CSL-81-1, Xerox Palo Alto Research Center, January 1981.
- [Lampson 81] B. W. Lampson, G. A. McDaniel, and S. M. Ornstein, "An Instruction Fetch Unit for a High-Performance Personal Computer", Technical Report CSL-81-1, Xerox Palo Alto Research Center, January 1981.
- [Lorie 81] R. Lorie, "Issues in Databases for Design Applications", *Proceedings of the IFIP Conference on File Structures and Databases for CAD*, Seeheim, September 1981.
- [Lorie 83] R. Lorie and W. Plouffe, "Complex Objects and Their Use in Design Transactions", *Proceedings of Database Week 1983*, San Jose, May 1983.
- [Metcalf 76] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks", *Communications of the ACM*, 19:7, July 1976.
- [Mitchell 79] J. Mitchell, W. Maybury, and R. Sweet, "Mesa Language Manual", Technical Report CSL-79-3, Xerox Palo Alto Research Center, April 1979. Unfortunately no later external reference on the Mesa language is available, and Cedar Mesa has diverged somewhat from the description in

this manual.

- [Payne 80] T. Payne, PASCAL Macroexpander Program, Stanford University Electrical Engineering Department, Computer Systems Laboratory, and Center for Integrated Systems, 1980.
- [Pier 83] K. A. Pier, "A Retrospective on the Dorado, A High-Performance Personal Computer", Technical Report ISI-83-1, Xerox Palo Alto Research Center, August 1983.
- [Sidle 80] T. Sidle, "Weakness of Commercial Data Base Management Systems in Engineering Applications", *Proceedings of the 17th Design Automation Conference*, Minneapolis, 1980.
- [Slutz 79] E. Slutz, "SDL Description of DEC PDP-11", Stanford University Computer Systems Laboratory and Center for Integrated Systems, 1979.
- [Steel 75] T. B. Steel, "Interim Report of the ANSI-SPARC Study Group", *ACM-SIGMOD FDT*, 7:2, September 1975.
- [Stonebraker 82] M. Stonebraker, and A. Guttman, "Using a Relational Database Management System for CAD Data", *Database Engineering*, 5:2, June 1982.
- [Stonebraker 83] M. Stonebraker, B. Rubenstein, and A. Guttman, "Application of Abstract Data Types and Abstract Indices to CAD Data Bases", *Proceedings of Database Week 1983*, San Jose, May 1983.
- [Teitelman 84] W. Teitelman, "A Tour Through Cedar", *Proceedings of the 7th Intl. Conference on Software Engineering*, Orlando, March 1984. To appear in *IEEE Transactions on Software Engineering*, April 1984.
- [Wiederhold 82] G. Wiederhold, A. Beetem, and G. Short, "A Database Approach to Communication in VLSI Design", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 1:2, April 1982.
- [Zintl 82] G. Zintl, "A CODASYL CAD Data Base System", *Proceedings of the 18th Design Automation Conference*, Nashville, June 1982.

Y. UDAGAWA and T. MIZOGUCHI

Mitsubishi Electric Corporation
325 Kamimachiya, Kamakura city, Kanagawa 247, Japan

1. INTRODUCTION

During the past quarter of a century, several data models have been developed, e.g. network, hierarchical and relational data models /1/. They have been designed for business data applications in which objects are represented by alphanumeric data and relationships among them are rather simple. In engineering data applications, however, pictorial data are essential in addition to alphanumeric data. Furthermore, relationships among data are so complex that they must be managed in multiple levels of abstraction. It has already been discussed that conventional database systems cannot effectively support engineering data management /2/. Much research has been done to investigate the use of relational data model for pictorial data management /3, 4, 5/. Describing pictures requires some extensions to the model. For example, Becerril et al./4/ developed a system to handle relations in which tuples are ordered and duplicated.

In this paper, we describe a database system for engineering applications, called ADAM (Advanced Database system with Abstraction Mechanism). ADAM is based on the relational model. The relational model was chosen because (1) it provides set-oriented data processing on a simple data structure, i.e. a relation, and (2) it provides a more effective way of representing the complex relationships among data than the other data models. ADAM data model uses the relational model for representing attributes information and relationships among data. As for pictorial information, ADAM provides figure-specification capabilities where pictorial notations of instances are represented in terms of built-in functions of computer graphics. Objects are modeled by multi-level constructions of these frameworks. Roughly speaking, the single-level construction without the figure specifications of the ADAM model corresponds to the relational model. The correspondence between Codd's relational model and ADAM is illustrated in Fig.1.

The main features of ADAM are as follows.

- (1) ADAM can deal with pictorial data as well as alphanumeric data.
- (2) ADAM has a facility called abstraction mechanism for managing a group of data as a unit.
- (3) ADAM uses instances with arguments (called abstract instances hereafter) for efficient description, storage and manipulation of repeated appearances of similar objects.
- (4) Objects are modeled as a hierarchical structure of instances. This allows the users to model objects step by step.
- (5) ADAM allows users to view a pictorial notation in any detail desired.
- (6) Updates of data affect a pictorial notation immediately.

2. SOME OBSERVATIONS ABOUT ENGINEERING DATA

By engineering data we mean all the data pertaining to industrial activities. Among these, we shall focus our attention upon electric circuit diagrams and system architecture diagrams, etc. Fig.2 shows a typical engineering data. Table 1 gives the identifiers and their notations for the parts in Fig.2. The followings are the main features of engineering data.

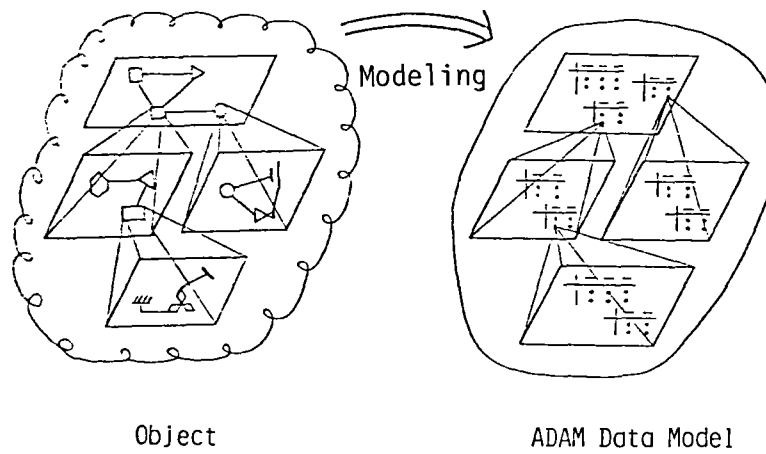
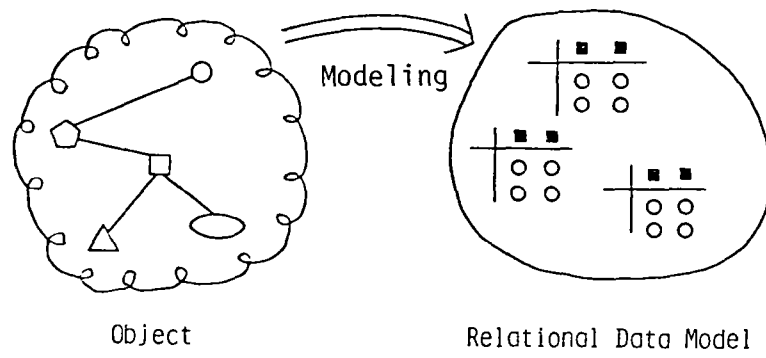


Fig.1 Correspondence between the relational data model and ADAM data model.

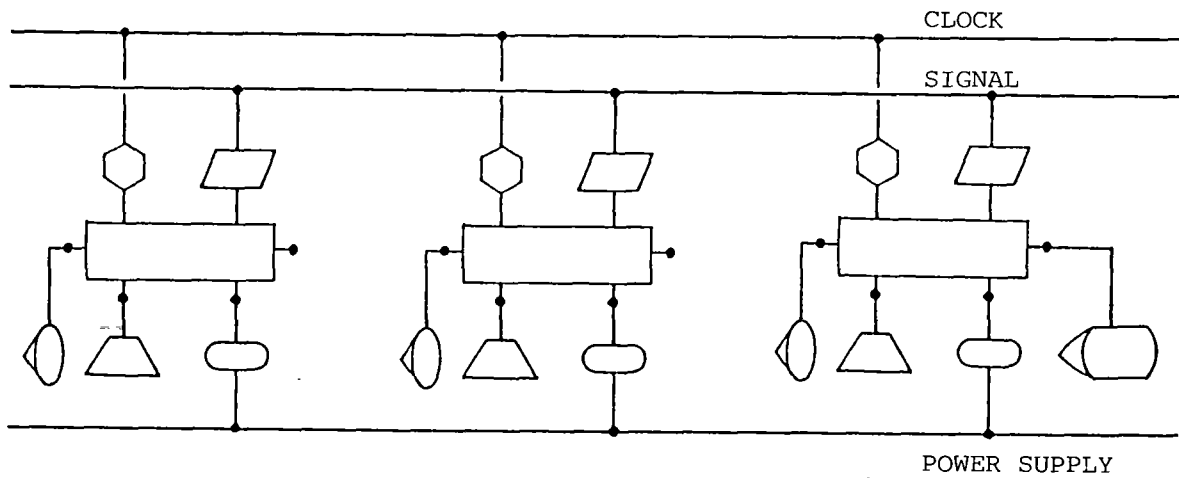






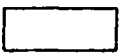


Fig.2 An example of engineering data.

Table 1. Parts in Fig. 1 and their notations.

Part	Notation	Part	Notation
Terminal	•	Synchronizer	
I/O Latch		Tone Controller	
Tuner		Timer	
Display		Signal Processor	

(1) Engineering data consist of pictorial information as well as alphanumeric information.

Each part has attribute information, e.g. for tone controller, maximum power of output, location installed and date manufactured. This kind of information is usually represented by fixed-format alphanumeric data in the same manner as in conventional databases. On the other hand, parts in Fig.2 have pictorial notations.

(2) Attributes relevant to objects are different one another.

For parts in Fig.1, there are attributes which are common to all objects, e.g. date manufactured. However there are attributes which are not common. For example, a display is specified by the size of screen, whereas tone controller is specified by its output power.

(3) Engineering data have too complex a structure to be manipulated by only one level of abstraction.

In most engineering applications, an object to be manipulated is represented by more than one diagram. Further, these diagrams are structured in such a way that an element in a diagram is represented by other detailed diagram. For example, the electric diagram in Fig.2 may best be managed in four levels of abstraction as shown in Fig.3 rather than one level of abstraction as in Fig.2

(4) Engineering data include many repeated instances of similar objects.

As an example, a terminal indicated by the notation • appears twenty-one times in Fig.2. This aspect arises from the fact that many parts used in engineering applications are standardized.

(5) The amount of information contained in engineering data is so enormous and their structure so complex that we can not analyze all the information in advance.

As a result, we can only model engineering objects via many interactions with the engineering database. This requires a database system to be more interactive and more flexible.

The features (3) and (5) are also pointed out in Haskin and Lorie /5/.

3. OVERVIEW OF ADAM

3.1 Data Definition Facilities

ADAM data model includes two classes of instances. One is a class of instances which are represented by fixed-format data, e.g. character strings, inte-

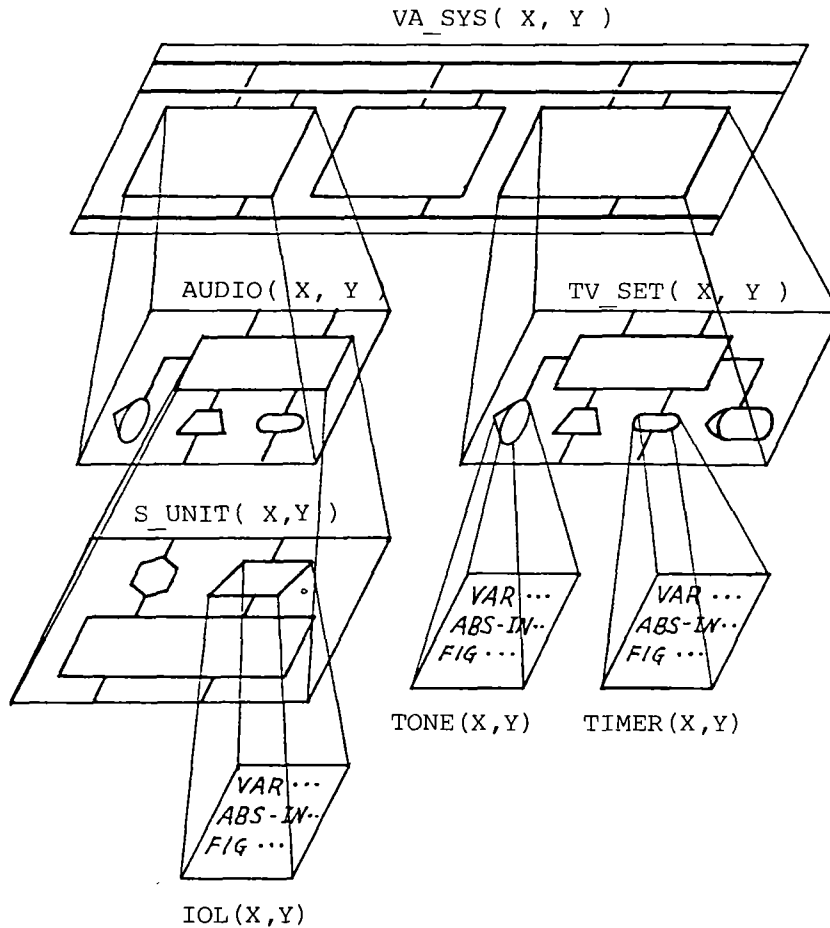


Fig.3 Hierarchical structure for the Fig.2.

gers and reals. Instances of this class are used in business-oriented databases. The other is a class of abstract instances (instances with arguments). An abstract instance, in turn, can be used to define other abstract instances. In this way abstract instances form a hierarchy.

ADAM data definition language consists of the following six parts.

- (1) variable declaration
- (2) abstract instance declaration
- (3) general figure specification
- (4) domain declaration
- (5) relation declaration
- (6) detailed figure specification

They are stored in four types of files as shown in Fig.4. For details see Udagawa and Mizoguchi /7/.

The most interesting feature of the data definition facilities of ADAM is that it includes two figure specification parts general and detailed. These two parts are provided, because abstract instances have two aspects of figure representations. Suppose an instance A is constructed from other instances, say B and C as in Fig.5. Now, if we view B and C from abstraction level A, we need only general notations which suggest detailed structures of A's constituents. That is, in dealing with constituents as units, their detailed structures are transparent

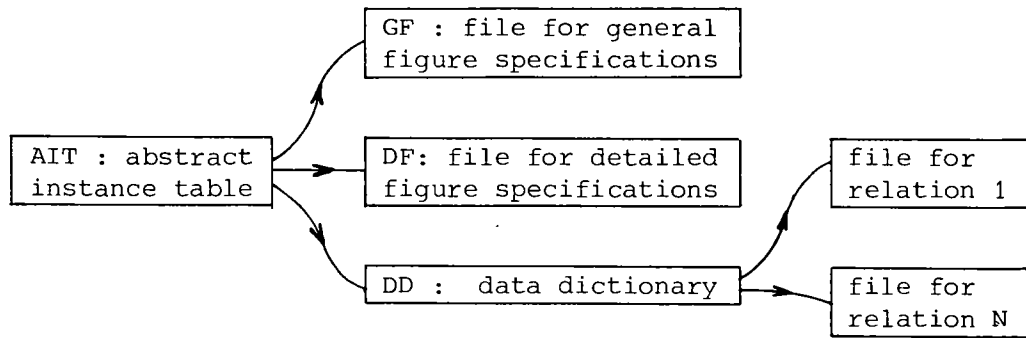


Fig.4 File organization of ADAM/X1.

to the general notations. The general figure specification part defines a general notation for the underlying instances. Intuitively, this part specifies the bold rectangle for instance A in Fig.5. A general figure is usually described in terms of built-in functions of computer graphics.

On the other hand, the detailed figure specification part specifies a pictorial notation for the detailed structure of an instance. A notation specified in this part is usually described in terms of graphic functions and relational operations. The notation allows users to get a precise pictorial representation, even if constituents of an abstract instance are frequently changed by updating relations. This part specifies the pictures in the bold rectangle for instance A in Fig.5.

Syntax of the figure specification parts are as follows.

```

< general figure specification > ::= $GENERAL_FIGURE ;
                                < figure specification list >
< detailed figure specification > ::= $DETAILED_FIGURE ;
                                < figure specification list >
< figure specification list > ::= < figure specification > ;
                                | < figure specification > ; < figure specification list >
< figure specification > ::= GRAPHICS ( < relational operation > )
                                | < temporary relation > := < relational operation >
                                | < graphics operation > < relational operation >
< graphics operation > ::= POINT | LINE | CIRCLE | ELLIPSE | GRID
                                | RECT | POLY | SYMBOL , etc.
  
```

Examples of the figure specification are shown in Fig.6.

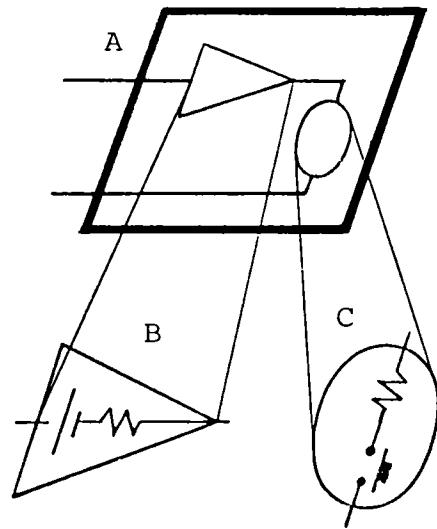


Fig.5 Correspondence between general and detailed figures.

```

$GENERAL-FIGURE ;
  LINE( X      , Y+16.0, X      , Y-16.0 ;
        X      , Y-16.0, X+36.0, Y-16.0 ;
        X+36.0, Y-16.0, X+36.0, Y+16.0 ;
        X+36.0, Y+16.0, X      , Y+16.0 ;
        X+34.0, Y+16.0, X+34.0, Y-16.0 ) ;

$DETAILED-FIGURE ;
  GRAPHICS ( FIG <! ELEM !> ) ;
  X ( H1, V1, DTE_ID, DINAME ) :=
    ( CONN <* STE_ID = TE_ID ,
      SINAME = INAME *> TERM )
    <! HCOOR, VCOOR, DTE_ID, DINAME !> ;
  LINE ( ( X <* DTE_ID = TE_ID ,
          DINAME = INAME *> TERM )
        <! H1, V1, HCOOR, VCOOR !> ) ;

```

Fig.6 Examples of figure specifications.

3.2 Data Manipulation Facilities

3.2.1 Equivalence and Ordering of Instances

Many data in engineering applications include similar instances. We introduced abstract instances to describe these instances efficiently. Introducing abstract instances lead to a basic mathematical problem, i.e. to define the equivalence and ordering among them. Ordering for abstract instances falls into two classes. One is the ordering for instances of different instance identifiers. The other is for the same identifiers.

As discussed already, instances in the ADAM data model form a hierarchy in which an instance is at a higher level than its constituents. For the former class, we define the ordering of abstract instances based on a hierarchy of instances.

Ordering Rule 1 : An instance is greater than its constituent instances.

Some examples of the ordering of instances in Fig.3 are given.

Ex.1 VA_SYS(*, *) > AUDIO(*, *) > S_UNIT(*, *) > IOL(*, *) , where * indicates any appropriate arguments.

Ex.2 Ordering among IOL(*, *), TONE(*, *) and TIMER(*, *) is not defined.

This ordering allows the users to traverse hierarchical structures of arbitrary levels.

Now we consider the ordering for instances of the same identifiers and therefore the same number of arguments. We define three ordering rules which are different from each other in processing of arguments.

Ordering Rule 2 : An ordering for instances is defined by the alphabetical and numerical ordering.

Ex.3 AUDIO(U, 3.0) < AUDIO(V, 3.0) < AUDIO(X, Y) , since U < V < X in EBCDIC code system.

Ordering Rule 3 : An ordering for instances is defined by the alphabetical and numerical ordering except for variables.

Ex.4 AUDIO(X, Y) < AUDIO(U, 3.0) = AUDIO(V, 3.0) , since Y < 3 in EBCDIC code system.

Ordering Rule 4 : An ordering of instances is defined by the following criteria.

- (1) If instances given are unifiable /6/, then they are equivalent.
- (2) If instances given are not unifiable because constants are unmatched, then ordering is defined by the alphabetical and numerical ordering of constants; otherwise it is not defined.

Ex.5 AUDIO(X, Y) = AUDIO(V+2.0, 5.0) since { V+2.0/X, 5.0/Y } is a legal substitution. Thus the set { AUDIO(X, Y), AUDIO(V+2.0, 5.0) } is unifiable.

Ex.6 AUDIO(U, 2.0) < AUDIO(5.0, 3.0), since { 5.0/U } is a legal substitution and 2.0 < 3.0.

Ex.7 Ordering between AUDIO(X, X+1.0) and AUDIO(U, U) is not defined, since { X/U, X+1.0/U } is an illegal substitution and ordering between X and X+1.0 is not defined.

Based on the ordering rules above, ADAM provides three evaluation modes for data manipulation.

(A) Alpha-evaluation mode

Expressions for data manipulation are evaluated according to Ordering Rules 1 and 2.

(B) Beta-evaluation mode

Expressions for data manipulation are evaluated according to Ordering Rules 1 and 3. In this mode, if identifiers are used systematically, we can efficiently distinguish whether a variable is substituted by a constant or not.

(C) Gamma-evaluation mode

Expressions for data manipulation are evaluated according to Ordering Rules 1 and 4. Gamma-evaluation mode is useful for manipulating abstract instances whose arguments are variables, constants and functions.

Fig.7 gives examples of natural join in each evaluation mode. Instances in ADAM are partially ordered in general. Thus "not-ordered join", "not-ordered restriction" operations, etc. are available in addition to the relational operations.

3.2.2 Data Retrieval Operations

As discussed in section 2, objects in engineering applications are modeled as multiple levels of abstraction.

Thus conventional data manipulation

facilities for the relational data model are insufficient for data manipulation in ADAM, because many essential data manipulations require a facility to traverse from one level to another through an arbitrary number of intermediate levels of abstraction. Data retrieval operations for the relational data model are extended so that the users can specify retrieval conditions at arbitrary (but adjacent) levels of hierarchy. For example,

\$MODE is gamma

< ELEM > FIG [ELEM] / (FIG [ELEM = 'TIMER(X,Y)']) [count (ELEM) > 1] specifies a query "find all the constituent modules which contains more than equal to one timer."

R1	INSTANCE	R2	INSTANCE	ATTR
	E(x,y)		E(x,y)	2x+y
	E(u,3)		E(2,3)	7
			E(x,3)	2x+3

(A) An example relational database.

R3	INSTANCE	ATTR
	E(x,y)	2x+y

(B) Alpha-natural-join :

R3 = R1 [INSTANCE = INSTANCE] R2

R4	INSTANCE	ATTR
	E(x,y)	2x+y
	E(u,3)	2u+3

(C) Beta-natural-join :

R4 = R1 [INSTANCE = INSTANCE] R2

R5	INSTANCE	ATTR
	E(x,y)	2x+y
	E(2,3)	7
	E(x,3)	2x+3

(D) Gamma-natural-join :

R5 = R1 [INSTANCE = INSTANCE] R2

Fig.7 Examples of natural-join.

3.3 Graphic facilities

As discussed in section 3.1, an entire pictorial notation for an instance is separately stored at different abstraction levels. So some sophisticated graphic algorithms have to be developed to retrieve a required pictorial notation.

Let $\langle\{\text{GFS}\},m\rangle$ denote general figure specifications at abstraction level m and $\langle\{\text{DFS}\},m\rangle \cup \text{GRAPHICS}(\langle\{\text{REL}\},m\rangle)$ denote detailed figure specifications at abstraction level m . $\langle\{\text{DFS}\},m\rangle$ indicates specifications using built-in graphic functions and $\langle\{\text{REL}\},m\rangle$, which is usually described in terms of relational operations, denoted an argument of the function GRAPHICS. The graphic algorithm in ADAM takes an integer, say n , as a parameter and produces the result as follows.

For $n = 0$, $\text{GRAPHICS}(\langle\{\text{GFS}\},m\rangle)$.

For $n = 1$, $\text{GRAPHICS}(\langle\{\text{DFS}\},m\rangle \cup \langle\{\text{GFS}\},m-1\rangle)$.

For $n = 2$, $\text{GRAPHICS}(\langle\{\text{DFS}\},m\rangle \cup \langle\{\text{DFS}\},m-1\rangle \cup \langle\{\text{GFS}\},m-2\rangle)$, etc.

Notice that the result is different from a simple overlay of figures at each abstraction level in that general figures of intermediate levels are not displayed. Applying the algorithm to the $\text{AUDIO}(X,Y)$ in Fig.3, we get the result shown in Fig.8.

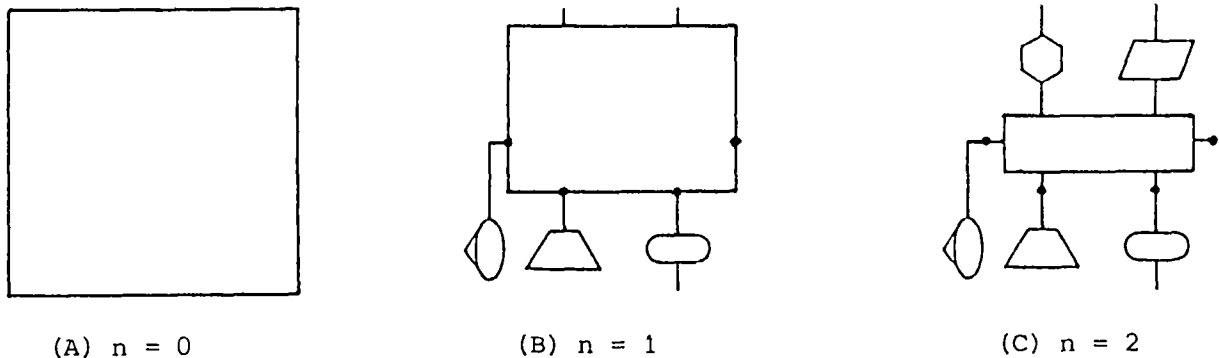


Fig.8 Results of the graphic algorithm in ADAM.

4. CONCLUDING REMARKS

This paper described an extended relational database system for the unified management of engineering data. Our approach is characterized

- (1) figure specification parts to augment the data definition facilities,
- (2) a facility called abstraction mechanism for dealing with a group of objects as a unit.
- (3) instances with arguments for manipulating repeated appearances of similar objects efficiently.

We have discussed overall facilities of ADAM, i.e. data definition, data manipulation and graphic facilities. Detailed discussions and examples are given in Udagawa and Mizoguchi /8/. A first version of ADAM is being implemented on MELCOM-COSMO 900 II computer system.

ACKNOWLEDGEMENT

The authors are grateful to Dr. M. Sudo, manager of Computer and Software Department at Information Systems and Electronics Development Laboratory for many helpful comments. We thank Dr. Won Kim, IBM Research Laboratory, San Jose, for his careful reading of the paper and comments on improving the presentation. We also thank Dr. Y. Masunaga, the University of Library and Information Science, for his encouragement to publish this article.

REFERENCES

- /1/ C.J.Date, "An introduction to database systems," 3rd. ed. Addison-Wesley, 1981.
- /2/ T.W.Sidle, "Weakness of commercial database management systems in engineering applications," Proc. 17th Design Automation Conf., 1980, pp.57-61.
- /3/ N.S.Chang and K.S.Fu "A relational database system for images," In "Pictorial Information Systems," Lecture Notes in Computer Science, Vol. 80, Springer-Verlag, pp.288-321, 1980.
- /4/ J.L.Becerril, R.Casajuana and R.A.Lorie "GSYSR : A relational database interface for graphics," In "Data Base Techniques for Pictorial Applications," Lecture Notes in Computer Science, Vol.81, Springer-Verlag, pp.459-474, 1980.
- /5/ R.L.Haskin and R.A.Lorie "On extending the functions of a relational database system," Proc. ACM-SIGMOD Int. Conf. Management of data, pp.207-212, June 2-4, 1982.
- /6/ J.A.Robinson "A machine oriented logic based on the resolution principle," J. Ass. Comput. Mach., Vol.12, No.1, pp.25-41, 1965.
- /7/ Y.Udagawa and T.Mizoguchi, "Implementation techniques of ADAM data definition language," Proc. 28th National Convention of Inf. Process. Society of Japan.
- /8/ Y.Udagawa and T.Mizoguchi, "An advanced database system ADAM --- towards integrated management of engineering data," Proc. IEEE Computer Data Engineering Conference, Apr. 1984.

INTEGRATION OF WORD PROCESSING AND DATABASE MANAGEMENT IN ENGINEERING ENVIRONMENT

Fumio Nakamura, Atsumi Kimura,
Sadasaburoh Kanai, Kazuhiko Ohmachi

Systems Development Lab., Hitachi, Ltd.
1099 Ohzenji, Asaoku, Kawasaki
215 JAPAN

1. Introduction

The Database technology for business applications is at a mature stage now. However, the database technology for engineering applications is only emerging. We can see the reason from the following differences between the histories of computerization in business and engineering applications:

(1) In business applications, data is transformed by sorting, merging, etc. and several reports are generated during the transformation. In an analysis of business systems, the design of input and output files and forms is important. Further, computerized business systems involve many simple repetitive applications.

(2) Engineering applications place heavy emphasis on algorithms (procedures), like complex numerical calculations. Subroutines are shared and managed in engineering systems. One evidence is FORTRAN. It has much poorer data or file manipulation functions than COBOL.

However, the environment of engineering or manufacturing companies is rapidly changing. More varieties of products must be designed and manufactured in shorter period. To accomplish this, integration of engineering applications is essential, and engineering databases hold the key and have drawn attention in recent years.¹⁻⁵⁾

We have addressed one of the problems in engineering database systems, support for design-document generation, and built a system called EASY-DOC (Engineering Activity support System -DOCument generation). Design documents, as well as drawings, are one of the most important outputs of the designers. Unlike drawings, however, computer aided design-document generation has not been frequently discussed. Design-document generation requires not only word processing but also database management, since design documents involve variable fields whose values vary from product to product. EASY-DOC is now being used for a pump design system at Hitachi.

In Section 2, we will review the features of engineering databases and technical problems to be solved to realize engineering database systems. We will clarify the features of design documents and describe the architecture of EASY-DOC in Section 3.

2. Features of Engineering Databases and Technical Problems

2.1 Features of Engineering Databases

A global view of an engineering database as a data center in a total design and production system is depicted in Figure 1. It includes three different types of data.

(1) Geometric data -- holds three-dimensional shape and attribute data of products, and drawing information which is a two-dimensional projection of shape data. It has highly complex relationships among data items and requires high speed data access. Therefore, current general purpose database management

systems (abbreviated to DBMSs) cannot fully manipulate the data structures with adequate performance. As a result, most present CAD systems move geometric data between main memory and secondary storage in simple ways (e.g. direct block access) and manage it as main-memory data during one job (e.g. processing of a drawing). This approach lacks flexibility in modifying data structures (e.g. addition of new attributes), since they do not provide even data definition facility which all DBMSs provide.

(2) Engineering data -- includes several kinds of data such as design data (other than geometric data) for the products being designed, design results and maintenance histories of completed products, reliability data, and design standards. Among the three types of data, this

is the least integrated into databases. Data access is made in an ad-hoc manner with a wide variety of data selection conditions. The most important factor in selecting DBMSs for this type of data is the ease of use, which is a significant advantage of relational DBMSs.

(3) Administration data -- is data required for managerial applications such as manufacturing control and cost management. This is already supported in on-line database systems using traditional DBMSs.

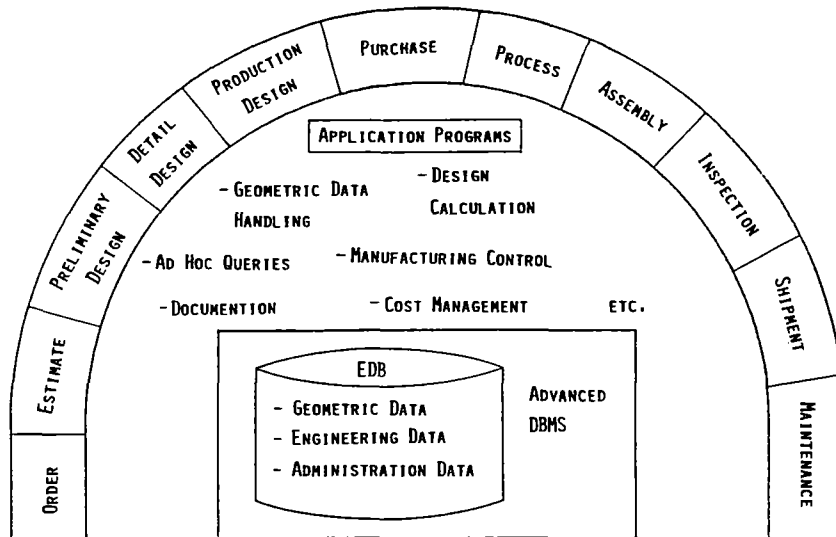


Figure 1. Global View of an Engineering Database System

2.2 Technical Issues in Engineering Database Systems

Since engineering applications have different data processing characteristics from business applications, The following technical problems must be solved to realize engineering database systems.

(1) Geometric Data Management

Geometric data has a complex data structure and is processed in the following manner:

- A block of semantically correlated data (e.g. data in one drawing) is retrieved as a unit.
- The retrieved data is interactively processed via a display terminal, over a relatively long duration.
- After completion of interaction, a new set of data is generated and replaces the old one.

Therefore, if geometric data structures are faithfully defined using data definition functions of current DBMSs, the performance will become intolerable because of lengthy database access time and locking overhead; thousands of records must be read in before the interaction and many of them must be updated during and after the interaction. On the other hand, database management concept is necessary for relating the geometric data with other attributes and product data.

(2) Numeric Data and Units

Engineering databases frequently require floating point numbers and arrays, unlike business databases. And such numeric fields usually carry units. Database processing must take units into consideration, and in some cases (e.g. exported products), unit conversion is required (e.g. between the metric system and the yard-pound system).

(3) Long Life of Data

In some cases (e.g. turbines in electric power plants), design data must be kept for tens of years. The old data is unloaded to magnetic tapes, and reloaded when required. The data structure may change during a long span of time. If it happens, the data reload could be unsuccessful. This is also true for products with rapid technical improvements like semiconductor products. For this, the system may have to allow multiple versions of data definition for one database.

(4) Program Management

Accumulation of programs and subroutines is an important aspect of engineering systems, and a method is required to manage these programs. The nucleus of a program management system is a program base. Although the notion of program base has not been established yet, a global view of a program base is "a set of data about programs for program development, execution, and maintenance". Engineering applications need a flexible program execution mechanism which enables to dynamically relate and execute independently developed programs.

(5) Design-Document Generation

This will be discussed in detail later.

3.3 Document Database

All information about defined documents and generated documents is stored in the document database and controlled by the document management facility. The principal entity sets and relationship sets are shown in Figure 5 as an Entity-Relationship diagram⁶⁾. The document database is directly maintained by EASY-DOC. Users access it through EASY-DOC.

(1) DOC-DEF holds overall document definition information like standard form format.

(2) FORM corresponds to defined forms and holds all information for each form specified via the document definition facility.

(3) VAR-FD holds information for each variable field specified during document definition such as length, internal data type, and display format.

(4) DOCUMENT holds information corresponding to each generated document from a defined document by setting variable field values. Further, more than one document of the same kind for each product (e.g. revisions) may exist.

(5) Each PAGE is constructed by setting variable field values in a form and by making minor changes to the form if necessary.

(6) FIGURE has information about figures and graphs to be inserted into forms and pages. It is extracted from graphic systems in plotter command form.

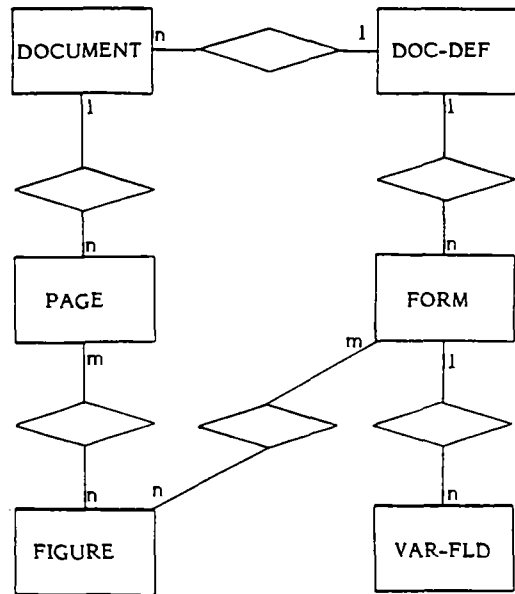


Figure 5. Main Data Structure of the Document Database

3.4 Program Generation

The program generation facility makes it easy for users to write procedures which set variable field values. Users can write procedures using the pseudo-codes the program generator provides. The pseudo-codes consist of FORTRAN, including database access calls, and form control statements. They may also include variable field names which are specified in the document definition. The program generator generates complete FORTRAN77 source programs from specified pseudo-codes by translating the form control statements and variable field names into appropriate FORTRAN procedural and declaration statements as shown in Figure 6. It refers to the document definition information in the document database during the translation process. An example pseudo-code program and the program generated from it is shown in Figure 7.

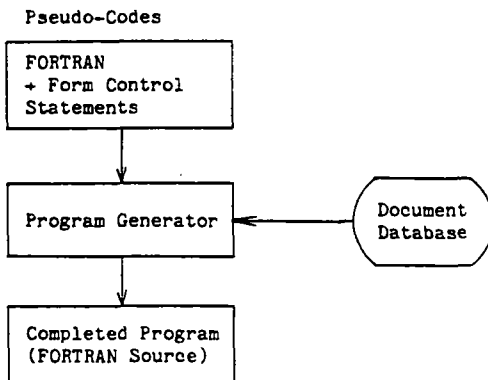


Figure 6. Program Generator

```

%OPEN  FORM1 ;
%GETF  ?A, ?B, ?C ;

(Read the values of the variable
fields ?D and ?E from the database
using the input values ?A, ?B and ?C.)
:
FORTRAN : statements
:
%LINK  PROG1 (?D, ?E, ?F) ;
      (Calculate the variable field ?F
      using ?D and ?E.)
%SETF  ?D, ?E, ?F ;
%DISPLAY PAGE1 ;
%SAVE  PAGE1 ;
%CLOSE FORM1 ;
END
    
```

(a) Pseudo-codes

```

CHARACTER*8 $FORM
CHARACTER*8 $FLD(3)
REAL $FA001
:
$FORM = 'FORM1'
CALL $OPEN ($FORM)
$FLD(1) = 'A'
$FLD(2) = 'B'
$FLD(3) = 'C'
CALL %GETF ($FLD(1), $FA001, $FLD(2), $FA002,
           $FLD(3), $FA003)

(Retrieval of ?D and ?E from the database.)
:
FORTRAN : statements
:
(Translation of %LINK, %SETF, %DISPLAY, %SAVE
omitted.)
CALL %CLOSE ($FORM)
END
    
```

(b) Generated FORTRAN Program

Figure 7. Example of Program Generation

(1) FORTRAN Statements

Pseudo-codes are primarily based on FORTRAN77 since most designers are familiar with FORTRAN, and FORTRAN77 can handle non-numeric data well.

(2) Variable Fields

When the program generator encounters a variable field whose name is preceded by "?", it takes the following actions:

- (a) It replaces the name by a FORTRAN variable name which begins with "\$F".
- (b) It generates a declaration statement for the variable by retrieving the variable field definition information from the document database.

(3) Form Control Statements

These are special statements provided by the program generator for form control such as open and close of forms, setting up variable field values, and display of pages on display terminals. The control statements begin with "%". There are ten different types of statements.

- (a) %OPEN makes a form ready for processing.
- (b) %CLOSE makes a form inaccessible.
- (c) %GETF is used to receive field values, from a display terminal in conversational mode or as card image data in batch mode.
- (d) %SETF sets up variable field values in the form. The values are edited based on the display format defined in the document database.
- (e) %DISPLAY outputs the generated page to the display terminal, for verification.
- (f) %SAVE stores the generated page to the document database.
- (g) %GETP returns the number of pages already generated in the document.
- (h) %SETP attaches a page number to the generated page.
- (i) %LINK dynamically links to another program specified.
- (j) %PUT outputs a message.

Acknowledgments

The authors wish to thank to Mr. K. Arai (Software Technology Promotion Center), Mr. T. Takanishi and Mr. S. Oyamada (Tsuchiura Works), and Mr. I. Yoshida (Systems Development Lab.) for their encouragements and cooperation with this work.

References

1. Survey Reports on Engineering Database systems (in Japanese), Joho-Shori Sinkoh Jigyoh Kyohkai (IPA) (March 1976, 1977, 1978).
2. A Survey Report on Computer Aided Engineering (in Japanese), Nihon Joho-Shori Kaihatsu Kyohkai (JIPDEC) (March 1982).
3. Computer-Aided Design, Vol.11, No.3 (Special issue on databases) (May 1979).
4. Encarnacao, J. (ed.): File Structures and Data Bases for CAD, Proc. IFIP WG5.2 Conf. (Sept. 1981).
5. Proc. of Engineering Design Applications in Database Week Conf., IEEE Computer Society Press (May 1983).
6. Chen, P.: The Entity-Relationship Model - Toward a Unified View of Data, ACM TODS, Vol.1, No.1 (March 1976).
7. Nakamura, F.: Engineering Databases (in Japanese), Joho-shori, Vol.25, No.4 (April 1984).

Call For Papers
INTERNATIONAL CONFERENCE ON
FOUNDATIONS OF DATA ORGANIZATION
Kyoto University, May 22 - 24, 1985

The International Conference on Foundations of Data Organization will be held at Kyoto University, Kyoto, Japan on May 22-24, 1985. Papers presenting original research on theoretical aspects of data organization are being sought.

Suggested Topics: typical, but not exclusive, topics include:

Mathematical file organization	Impact of VLSI on file organization
Consecutive retrieval property and applications	File organization for relational databases
Geometrical techniques for data organization	Models of data organization
Data organization for high-level databases (historical, inferential, statistical, CAD databases, etc.)	

Submission of Papers: Authors are invited to submit three copies of a full paper before October 15, 1984 to the Vice-Chairperson of the Program Committee.

Program Committee Chairperson

Witold Lipski
 Université de Paris-Sud
 Centre d'Orsay
 Laboratoire de Recherche en Informatique
 Bât. 490
 91405 ORSAY Cédex, France

Program Committee Vice-Chairperson

Katsumi Tanaka
 College of Liberal Arts
 Kobe University
 Nada, Kobe 657, Japan

Authors will be notified of acceptance/rejection by January 25, 1985. Final papers will be due by March 10, 1985.

Honorary Conference Chairperson

Sumiyasu Yamamoto
 Department of Applied Mathematics
 Science University of Tokyo
 Shinjuku, Tokyo 162, Japan

Conference Chairperson

Yahiko Kambayashi
 Department of Computer Science & Com. Eng.
 Kyushu University
 Hakozaki, Fukuoka 812, Japan

International Organization Committee

Chairperson

Sakti P. Ghosh
 IBM Research K54/282
 5600 Cottle Road
 San Jose, California 95193, U.S.A.

European Coordinator

Fabrizio Luccio (Università di Pisa)
 Middle East/Africa Coordinator
 Sabah S. Al-Fedaghi (Kuwait University)

Program Committee Members:

François Bancilhon (Université de Paris-Sud, France)
 Walter A. Burkhard (University of California, San Diego, U.S.A.)
 Merrick Furst (Carnegie-Mellon University, U.S.A.)
 Hideki Imai (Yokohama National University, Japan)
 Won Kim (IBM Research, U.S.A.)
 Yoshifumi Masunaga (University of Library and Information Science, Japan)
 J. Ian Munro (University of Waterloo, Canada)
 Tetsuo Mizoguchi (Mitsubishi, Japan)
 Peter Scheuermann (Northwestern University, U.S.A.)
 Michel Scholl (INRIA, France)
 Shinsei Tazawa (Kinki University, Japan)
 Mirosław Truszczyński (Technical University of Warsaw, Poland)

An attempt will be made to publish selected outstanding papers of the conference in the book form, similar to the previous conference, which was held in Warsaw in August 1981 (Academic Press, titled: Data Base File Organization, 1983).

Kyoto used to be the capital of Japan for over one thousand years. There will be Tsukuba Science EXPO'85 at Tsukuba. Details to attend the conference are available from the Conference Publicity Chairperson.

International Organizing Committee Members

Canada:	Tim H. Merrett (McGill University)	Yutaka Matsushita (Oki)
China:	Shu Gang Shi (Wuhan University)	Takeo Nakano (St. Paul's University)
Czechoslovakia:	Braislav Rován (Komenský University)	Hirotsuka Sakai (Hitachi Software)
France:	Claude Delobel (University of Grenoble)	Kenji Suzuki, Hirofumi Katsuno (NTT)
India:	Jogobrata Roy (Indian Statistical Institute)	Kazuhiko Ohnishi (Hitachi)
Italy:	Barbara Pernici (Politecnico di Milano)	Setauo Ohsuga (University of Tokyo)
Japan:	Yoshihiro Akiyama (IBM Japan)	Syunsuke Uemura (Electrotechnical Lab.)
	Setuo Arikawa (Kyushu University)	Shkko Lee (Seoul National University)
	Ryosuke Hotaka (Tsukuba University)	Poland: Wiktor Marek (Warsaw University)
	Hideto Ikeda (Hiroshima University)	Singapore: Robert A. Cook (Nat. Univ. of Singapore)
	Yoshiaki Ishii (Software A.C. Far East)	Switzerland: Jürg Nievergelt (ETH)
	Isamu Kobayashi (Sanno University)	U.S.A.: Paul Dietz
	Akifumi Makinouchi (Fujitsu)	(University of Southern California)
	Masao Managaki (NEC)	Chung Le Viet (HDR Systems)
		W. Germany: H.-D. Ehrlich (Tech. Universität Braunschweig)

Local Arrangement:

Chairperson: Kazuo Iwama (Kyoto Sangyo University)
Members: Shirou Iwasawa (IBM Japan), Shojiro Muro (Kyoto University), Osamu Konishi (Nagoya University), Yuzuru Hiraga (University of Library and Information Science)

Publicity and Treasurer:

Chairperson: Masatoshi Yoshikawa (Kyoto University)
Members: Hiroto Yasuura (Kyoto University), Tetsuya Furukawa (Kyoto University)

In cooperation with:

IEEE Computer Society, ACM SIGMOD, Kyoto University, Kyushu University, IBM Research, IBM Japan

Call for Papers and Participation

FIRST INTERNATIONAL WORKSHOP ON EXPERT DATABASE SYSTEMS

October 25-27, 1984, Kiawah Island, South Carolina



Sponsored by:

The Institute of Information Management, Technology and Policy,
College of Business Administration,
University of South Carolina

In cooperation with:

Association for Computing Machinery – SIGMOD and SIGART



IEEE Technical Committee on Data Base Engineering

This workshop will address the theoretical and practical issues involved in making databases more knowledgeable and supportive of AI applications. The tools and techniques of database management are being used to represent and manage more complex types of data and applications environments.

The rapid growth of online systems containing text, bibliographic, and videotex databases with their specialized knowledge, and the development of expert systems for scientific, engineering and business applications indicate the need for intelligent database interfaces and new database system architectures.

The workshop will bring together researchers and practitioners from academia and industry to discuss these issues in Plenary Sessions and specialized Working Groups. The Program Committee will invite 40 to 80 people, based on submitted research and application papers (5000 words) and issue-oriented position papers (2000-3000 words). Topics of interest include (but are not limited to):

Knowledge Base Systems

environments
architectures
languages
hardware

Knowledge Engineering

acquisition
representation
design
learning

Expert Database Systems

natural language access
domain experts
database design tools
knowledge gateways
industrial applications

Database Specification Methodologies

object-oriented models
temporal logic
enterprise models
transactional databases

Constraint and Rule Management

metadata management
data dictionaries
constraint specification
verification, and enforcement

Reasoning on Large Databases

fuzzy reasoning
deductive databases
semantic query optimization

Please send five (5) copies of full papers or position papers by **June 1, 1984** to:

Larry Kerschberg, Program Chairperson

College of Business Administration
University of South Carolina
Columbia, SC 29208
(803) 777-7159/messages 777-5766

Submissions will be considered by the Program Committee and authors will be notified of acceptance or rejection by July 16, 1984. Preprints of accepted papers will be available at the workshop. Workshop presentations, discussions, and working group reports will be published in book form.

Announcing a major new IEEE Computer Society membership benefit . . .



Computer communications for today's computer professionals

Using COMPMAIL+ you can . . .

1. Communicate via electronic mail with your colleagues — one-on-one, or in electronic mail conferences. Either way, there's no more postal system delays, no more "telephone tag."
 2. Access Computer Society listings of upcoming conferences, publications, and technical activities.
 3. Scan the complete, up-to-the-minute list of Computer Society Press publications.
 4. Speed up your Computer Society publications orders and conference registrations by ordering on-line and charging to your credit card. On-line book orders are shipped in 48 hours; conference registrations are processed in 24 hours.
 5. Locate your colleagues in the system by accessing a complete on-line directory of all COMPMAIL+ users.
 6. Post your own messages to — and scan — a society-wide electronic bulletin board.
 7. Set up your own executive calendar system: schedule meetings, display your own open time slots, and scan colleagues' schedules for open time slots.
 8. Access state, national, and international newswires. Scan the headlines, or do keyword inquiries.
 9. Obtain current stock, bond, and commodity quotes.
 10. Use the on-line Official Airline Guide to obtain current flight schedules and fares.
 11. Utilize a wide range of programs in the system library (over 200), ranging from finance and statistical routines to games.
 12. Create your own programs and data files using compilers and the database management system resident on COMPMAIL+.
- And remember: this is just a partial list of the tools and facilities available through COMPMAIL+ right now. Even more will be available in the future.



THE INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.

EASY TO USE

To assure maximum access for all society members, COMPMAIL+ is available via Telenet, Tymnet, or Uninet. All you need is a telephone, a modem, and a terminal or a microcomputer capable of communicating via ASCII protocols over telephone lines. When you log on you'll see a complete menu of available services, together with extensive on-line help commands and functions. There's even a special HELP mailbox in case you run into a problem, and a SUGGESTION mailbox in case you think of enhancements that will make the system even more useful to Computer Society members.

LOW COST

For most COMPMAIL+ services, the basic rate is an hourly connect charge that varies by time of day. In addition, there is a communications charge that varies by time of day and by the particular network you select (Telenet, Tymnet, or Uninet). The following sample rates cover basic electronic mail and communications assuming you access COMPMAIL+ via Telenet:

Prime Time	(8:00 a.m.* to 6:00 p.m., Monday through Friday)	\$16/hour
Off-Prime	(6:01 p.m. to 9:00 p.m., Monday through Friday, and 8:00 a.m. to 9:00 p.m., Saturdays, Sundays, and holidays)	\$7/hour
Nighttime	(9:01 p.m. to 7:59 a.m. daily)	\$6/hour

*Times shown are based on Eastern Time

If you use the filing capability of the system there is a small storage charge (40¢ per 2048-byte storage unit per month). There are surcharges for use of some of the special services such as the OAG and news or stock quotation systems. Finally, if you use the system for programming or database applications there are CPU time-and-storage charges. A complete schedule of rates for all services will be sent to all new subscribers, so you'll know exactly what each service costs before you use the system.

BEST OF ALL . . .

- There's no start-up or enrollment charge.
- As a special introductory offer, you'll receive a **\$30 CREDIT** toward your use of the basic COMPMAIL+ services (items 1 through 7 opposite)

SUBSCRIBE NOW!

Complete and mail the coupon below. That's all there is to it. By return mail you'll receive an ID and password; a complete schedule of rates, terms, and conditions; and basic documentation

COMPMAIL+ APPLICATION

ITT Dialcom, Inc. is hereby authorized to register me as a user in the IEEE Computer Society's COMPMAIL+ system. I understand that as an introduction to the new system, I will receive a \$30.00 credit toward my use of electronic mail and communications (Tymnet and Telenet) services. The credit is not applicable to the use of surcharge services such as OAG, Unistox, Info-X, and UPI. Further, the credit is only applicable to charges incurred through the last day of the month following the month in which I am registered. I understand that COMPMAIL+ services will be made available upon ITT Dialcom's standard terms and conditions for COMPMAIL+ services at the rates specified in the COMPMAIL+ schedule of prices, which will be furnished to me with my access ID number and password. I agree that system use will be upon said terms and conditions and at said rates and agree to be bound thereby. I further understand that: if I do not use the system, no charge will be incurred, this authorization constitutes no other obligation to me, charges for my use of COMPMAIL+ services may be invoiced to my credit card account as indicated below, and, if I elect not to use my credit card and opt for direct invoice, a \$25.00 minimum monthly usage charge will apply.

Check one: VISA MasterCard Direct Invoicing

PLEASE PRINT OR TYPE

Credit Card Account No _____

Credit Card Exp Date _____

My billing address is

Name _____

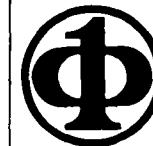
Membership No (mandatory) _____

Address _____

City/State/Zip _____

Signature _____

Date _____ THIS FORM MAY BE DUPLICATED



Return to:
Compmail +
IEEE Computer Society
P.O. Box 3489
Silver Spring, MD 20901

TCDBE

