(4)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY — VLSI PUBLICATIONS

COMPUTER-AIDED FABRICATION SYSTEM IMPLEMENTATION

Semiannual Technical Report for the period April 1, 1988 to September 30, 1988

Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Principal Investigators:		
	Paul Penfield, Jr.	(617) 253-2506
	Dimitri A. Antoniadis	(617) 253-4693
	Stanley B. Gershwin	(617) 253-2149
	Stephen D. Senturia	(617) 253-6869
	Emanuel M. Sachs	(617) 253-5831
	Donald E. Troxel	(617) 253-2570

This research was sponsored by Defense Advanced Research Projects Agency (DoD), through the Office of Naval Research under Contract No. N00014-85-K-0213.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138

TABLE OF CONTENTS

*G. H. Prueger, Equipment Model for the Low Pressure Chemical Vapor Deposition of Polysilicon, M.S. Thesis, Department of Mechanical Engineering, MIT, March 1988. Also, MIT VLSI Memo No. 88-485, November 1988.

*T.-L. Tung, Boundary Element Techniques for Modeling Thermal Oxidation of Silicon, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, September 1988.

*R. Jayavant, An Intelligent Process Flow Language Editor, M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, September 1988. Also, MIT VLSI Memo No. 88-475, September 1988.

J. Huang, T. A. Lober, M. A. Schmidt, and S. D. Senturia, "The Maximum Free-Standing Length of Polycrystalline Silicon Microbeams," to appear in Proceedings, International Conference on Material and Process Characterization, Shanghai, China, October 1988.

*S. B. Gershwin, "Hierarchical Flow Control: A Framework for Scheduling and Planning Discrete Events in Manufacturing Systems," to appear in Proceedings, IEEE Special Issue on Discrete Event Systems. Also, MIT VLSI Memo No. 88-482, October 1988, earlier version appeared as MIT VLSI Memo No. 88-406, July 1987.

M. L. Heytens and R. S. Nikhil, "GESTALT: An Expressive Database Programming System," submitted to the ACM SIGMOD. Also, MIT VLSI Memo No. 88-484, October 1988.

---

* Abstract only. Complete version available from Microsystems Research Center, Room 39-321, MIT, Cambridge, MA 02139; telephone (617) 253-8138.

RESEARCH OVERVIEW

The objective of this contract is to do research supporting the development of a computer-aided fabrication system that will serve the needs of the American semiconductor industry and the VLSI design community. The semiconductor industry requires computer tools to enhance yields and reduce cost. VLSI designers can benefit from the ability to specify application-specific processes to fabricate their application-specific integrated circuits.

In support of these goals, specific projects are in place to design, develop, and implement a hardware/software CAF system architecture; to develop aids to specifying process flows in a modular way; to develop models for semiconductor fabrication equipment; to develop technology ~~GAD~~ (TCAD) tools for the mechanical properties of integrated structures; and to develop and implement algorithms and programs to assist in scheduling of fabrication operations.

Progress in each of these areas is described below.

CAF SYSTEM STRUCTURE

We made substantial progress in the development of programs relating to our data model and schema. Our Gestalt system architecture provides a uniform query interface to data residing in multiple autonomous, heterogeneous data bases. The Gestalt data base interface routines were expanded to include Lisp interface routines in addition to C interface routines by Ken Ishii.

Mike Ruf has completed his S.M. thesis proposal, "Management of IC Manufacturing Data," August 18, 1988. Mike worked on our CAFE project this past spring and is now at TI on his VI-A internship company assignment. While his work is actually being done at TI, I am his MIT S.M. thesis supervisor. This thesis is an example of how our CAFE system data base philosophy is influencing related work at TI. A comprehensive set of data base tools, DBTOOLS, were generated by Ruf. These include:

• dbinspect - which is a "data base walker" program which enables application programmers (and others) to find their way around the existing data base. It displays an existing entity and allows the user to explore related entities. For example, one can display a facility and see that it has a list of machines. From there one can display a particular machine and see its attributes, etc.
• dbcreate - which enables the creation of entities without writing a special application program.
• dbquery - which can be used to implement data base queries.
• dbmutate - to enable changes to be made to existing entities.
• dbchoose - to select entities from the data base.

An S.B. thesis, "Schema Viewer: A Graphical Representation to Portray the Database Schema of the MIT CAFE System," was completed by Nazhin S. Zarghamee in May, 1988. This software provides a schema display so as to allow lab managers and users to provide more meaningful feedback as to the appropriateness and utility of our schema.

A number of application programs pertaining to status and log reports were written or modified by D. E. Troxel. These included *change-status* which is used for entering data relating to status changes, *describe-machine* to display current status, *new-machine* to enter a new machine into the data base, *format-equipment* to report the status of all equipment in a facility, *up* which is a generalized status and log report generator, and *dbt* which was used to transfer file based data to the data base. The *up* program can produce graphs of uptime for a selected machine for a selected period of time or, alternatively, a summary of the relevant log entries relating to machine status changes. The graphs can be output on a terminal or laser printer via *giraphe3* which was written by Duane Boning and Bob Harris.

A new program, *operate-machine*, was written by Mike McIlrath. This is initially used to make log entries when a machine is operated. It also provides a base for the creation of machine specific data entry and will be expanded substantially in the near future. In addition to being available from the CAFE menu, this program or procedure will be called by the fabrication interpreter in order to actually effect the machine operations. The *operate-machine* program is being expanded to create wip entities when the program is exited without the finish time being specified. When this is accomplished several programs (*up*, *change-status*, and *describe-machine*) will have to be modified to include reports on active wips.

The next stage in the development of operate-machine is to make it specific to the actual machine being operated. This requires generation of opdesc entities which embody the description of the parameters or machine settings and the measurements or data which are to be collected.

A new program which interfaces with the Nanospec film thickness instrument has been developed. In addition to operating the Nanospec via the computer, it now initiates measurements and places the resulting thickness measurement in the appropriate field automatically, thereby reducing the operator interaction required to capture the measurement data for data base storage.

A program written by Peter Monta provides for an alternative direct interface between the Gyrex mask maker and a computer as opposed to requiring users to write data to magnetic tape and transport these tapes to the Gyrex.

We have developed and demonstrated a "hands off terminal." We chose a commercially available TI speech recognition card which plugs into an IBM PC/XT. Software has been developed by Peter Monta to interface the TI PC software to control a fabform interface. As this speech recognition module is speaker dependent, the software automatically loads the data base appropriate to the login name. Several of us have "trained" the recognition software and the results are quite interesting. This "hands off terminal" is now ready to be installed as the terminal next to the Nanospec in the fabrication laboratory.

We have continued the development of a process flow language (PFL). The creation of a PFL and associated interpreters is the key to our approach for generating actual fabrication instructions and for collecting the data resulting from actual fabrication steps. The interpreters provide the actual meaning of the process flows expressed in the flow language.

Our previous PFL development was based on only the machine setting view in order to get something working as soon as possible. We now have a version of our PFL which is based on the two stage process step model which relates the goal of a change in wafer state first to the physical treatment parameters and finally to the actual machine settings used to process the wafers. We have recoded the CMOS baseline process in this new version and, in addition, have encoded a furnace monitor process which is routinely used every week.

We have made substantial progress on an expert PFL editor. An expert PFL editor has been completed by Rajeev Jayavant as his S.M. Thesis, "An Intelligent Process Flow Language Editor". This editor uses fabform as the user interface. Ideally one starts with an existing process flow, encoded in out lisp like syntax, which is somewhat similar to the desired process flow. The editor then displays this existing process flow with a forms based presentation and allows the user to modify the flow. The editor then produces the new flow encoded in our lisp like PFL without the user even being aware of the lisp nature of the PFL. The editor supports the three views required by the two stage generic process model and, in addition, allows any number of hierarchical levels of process flow definition.

Our standard user interface, fabform, has been improved and extended so that it can now be called as a procedure from either C or Lisp.

We have made substantial progress on the development of a simulation interpreter. Duane Boning has completed his Ph.D. proposal, "Custom Fabrication Process Design: Tools and Methodologies," March 8, 1988. We have also realized that we must provide for operation of partial flows. One impediment to the use of our PFL is that users change their minds about the process specification as they progress with the actual fabrication. By concatenating the processing history of fabrication with a number of partial flows we at least will have a trace which accurately reflects what happened. A prototype version of the Suprem-III Simulation Interpreter has been completed. The interpreter generates Suprem-III fragments for multiple one-dimensional cross sections, produces a Makefile to minimize shared simulations, and provides analysis (plotting, sheet resistance, and threshold voltage) capabilities. Duane Boning is now working to get an accurate description of the CMOS Baseline process, so that meaningful comparisons of fabrication and simulation of the MIT standard defect array is possible, and so that realistic work with the flow language can progress.

We completed a substantial reorganization and cleanup of our CAFE software. We now have it all existing under a single directory, /usr/cafe, in preparation for distribution to our fabrication laboratory and perhaps elsewhere.

We have acquired another computer and several workstations. We installed a VAX 750, garcon, for use as a file server. We have installed three monochrome and two color VS2000s, two Symbolics Lisp machines, and relocated the TI Explorer. We have installed a terminal concentrator in building 13. All of our computers are now on the same subnet with the result that file transfer and NFS services are now robust.

This summer we made a major new release of our CAFE software. We installed INGRES 5.1, an updated schema, *operate-machine*, and a series of applications programs related to status and log reports. These programs were described in previous progress reports. In addition, Mike Heytens generated a number of new database access routines which provide for more efficient filtering and sorting by implementing these operations at the underlying data base level instead of at the applications programming level. In spite of these optimizations we found that the system response time had deteriorated substantially both due to increased program size and an increase in the number of active users.

We temporarily took the computer caf1.mit.edu used for CAFE development out of service in order to provide better response time on caf2.mit.edu, which is the computer used by fabrication laboratory users. This doubled the memory available on caf2 from 8 to 16 megabytes. Later, we were able to borrow some memory and we restored caf1 to service and now have 16 megabytes of main memory for caf2. We conducted timing tests on caf2 with 16, 24, and 32 megabytes of memory. Seven identical CAFE operations were started in quick succession. One sample operation implemented in Lisp took 160 seconds with 16 megabytes and 8 seconds with 24 or 32 megabytes. A different operation implemented in C took 125 seconds with 16 megabytes and 85 to 90 seconds with 24 or 32 megabytes. The primary reason that the Lisp program is slower than the C program is that it is much larger and the paging time is thus longer. Clearly we need to acquire more memory.

A fast method of logging in and out of the laboratory has been implemented by Rajeev Jayavant. The actual computations involved is not any faster but the user no longer must wait until they finish. This has been tested but not yet installed on caf2.

An extension to the reservation program has been made and tested, though not yet installed on caf2. A new program to make periodic reservations was written by Joseph Kaliszewski and he also modified the existing reservations program. Periodic reservation enables the convenient entry of lab hours and weekly scheduled preventive maintenance. Reservations indicated by the periodic reservation data structure are merged into blank areas of the reservation forms so as to provide this information to lab users. However, if a lab user wants to make a reservation which overrides these, he or she can do so.

We plan to re-implement the GESTALT object-like interface to provide an in-memory storage option for those sites and applications who do not want or need a persistent database, and do want and need fast, in-memory, object-like data structuring.

We have begun serious consideration of the integration of scheduling programs with CAFE. Xiewei Bai has started to integrate fabform into his factory simulation programs. We have discussed schema additions to represent the data required for scheduling programs but have not yet settled on their final form. The present approach is to write a single machine scheduler first with later extension to a multi-machine scheduler.

The programs relating to lot and wafer tracking have been rewritten and tested but not yet installed. *Create_lot* and *create_processname* were rewritten by Rajeev Jayavant. *Showlot* and *tracklot* were rewritten by D. E. Troxel. The latter two programs include optional reports of the history of opinsts and existing wips.

A project to automate the operation of the HP wafer prober has been initiated by Merit Hung and Joseph Kaliszewski under the direction of Professor Antoniadis. This project is proceeding in three phases, the first being development of a stand alone program on the computer supplied with the wafer prober. The second phase will be to enable the operation of this machine from another computer (caf and/or caf2). A third phase will be to integrate the operation of this machine with the *operate-machine* program running under CAFE.

MODULAR PROCESS

Development of the Profile Interchange Format (PIF) has proceeded in two areas: implementation of a PIF program interface, and development of individual PIF utilities. Taking the published proposal for the "intersite" (or ASCII) PIF as a starting point, we have been developing a set of routines through which CAD and CIM programs may access PIF objects. This "intertool" format is based on the GESTALT database interface to a geometric and attribute schema definition tailored to the PIF, and provides a uniform or "standard" functional program interface. Tools implemented with this interface should, like the data itself, thus be portable. We have so far developed a small set of such utilities, including format conversion programs from SUPREM-III to the PIF ("sup2pif" and "pif2sup"), between the ASCII and database versions ("pif2db" and "pifdump"), as well as a simple profile plotting utility ("pifplot"). Implementation of the interface and specific PIF utilities is continuing.

The Technology (encompassing both Process and Device) CAD Environment is based solidly on the wafer representation (PIF) and the process representation (PFL or Process Flow Language). During the last six months, we have focussed on development of two tools for this environment. The first of these is a prototype Simulation Manager, which has as a key component a translator from the PFL representation of the process to the input required by SUPREM-III. The specification and handling of multiple one-dimensional cross sections for simulation has been a major addition to the Manager. Utilities allow minimal simulation of the process (the MIT CMOS Baseline process has been the vehicle for testing the manager), evaluation of where the process diverges for multiple cross sections, as well as interactive examination of simulation results. The second area of research we have begun is to investigate simple process synthesis utilities. We have written initial versions of an "Oxidation Advisor," an "Implantation Advisor," and a "Diffusion Advisor" to provide physically-based initial guesses for process parameters during process design. Work is underway to further build "Correction Advisors", as well as to experiment with physically-based optimization methods.

EQUIPMENT MODELING

During this time period, progress was made on three projects, and a fourth project was initiated. The first ongoing project is an equipment model for the LPCVD of doped polysilicon. The second ongoing project concerns the use of dimensional analysis in the design of experiments. The third ongoing effort concerns the use of sequential design of experiments for process optimization and control. The new project concerns the development of a general framework for process control.

A common element in our equipment modeling efforts is the fusion of statistical design of experiments with physically based mechanistic modeling. Our first completed work is an equipment model for LPCVD of polysilicon in a horizontal tube furnace. In this work, George Prueger developed a finite difference model for the equipment and process and calibrated that model using data derived from designed experiments. Our current projects on modeling of doped poly and the use of sequential design of experiments seek to further the effort at the combination of experimental design and physical knowledge.

The project on modeling of the LPCVD doped poly is being carried out by Master's student Parmeet Chaddha with experimental support coming from the BTU applications lab, in Billerica, MA. The goal of this work is to develop an equipment model which aids in the design of the cage in the doped poly process. The function of the cage is to serve as a deposition site for reactants formed in the annular layer between the cage and the tube walls, thereby resulting in uniform thickness deposition on the wafers. The challenge in cage design is to specify the size and distribution of the holes so that the deposit is uniform on the wafers, and the deposition rate is as high as possible. Our approach is to develop a highly simplified analytical model which predicts deposition uniformity and rate as a function of cage geometry, and to calibrate this model using data derived from designed experiments. Roughly one half of the experiments have been completed during this reporting period, with the work being performed on 4 inch wafers at BTU, Inc.

Master's student William Wehrle has been developing methods of using dimensional analysis in the design of experiments. Dimensional analysis is a relatively easy means of deriving relationships between variables based on physical arguments. A very general theorem called the Pi Theorem provides a set of rules which can be followed to create the dimensionless groupings which characterize a problem. The theorem, however, results in a tremendous choice of sets of these groupings. Mr. Wehrle has developed two rules which narrow the choice down when the Pi theorem is applied to the design of experiments. He will test the theory in application to the LPCVD of LTO.

Master's student Michele Storm is working on the use of sequential design of experiments for process optimization and control, in collaboration with Ultramax Corp. of Cincinnati, OH. Sequential optimization is a technique particularly well suited to manufacturing, where processes are inherently sequential. The basis of the method is to perform a local regression to the measured data, thereby calibrating a quadratic response surface centered near the current point of operation. This response surface may be considered to be a Taylor series representation of the process. The polynomial representation is then used to design the next experiment in a continuous cycle of "learning and advising". Our goal is to use dimensional analysis to create grouped variables for use in the models, as a replacement for the primitive variable currently used. Ms. Storm has written a sequential optimization program and will shortly begin experimental work on wire bonding. She will compare the effectiveness of grouped (dimensionless) variables with the effectiveness of primitive variables for optimization of wire bonding.

Doctoral student Ruey-Shan Guo was hired toward the end of the summer (August 1988). Mr. Guo has spent the intervening months acquiring a background in statistics and experimental design.

MECHANICAL-PROPERTY TCAD

The overall goal of this project is the development of a design capability that includes the mechanical properties of microelectronic materials. This is a new project, which began in October 1987. The specific goal during this past year has been to develop experimental methodologies with which to determine the residual stress in microelectronic thin films. The material selected for the first set of experiments was polycrystalline silicon, because it is well known that its residual stress is very dependent on process conditions. Several experiments were carried out on polysilicon as a function of its deposition and doping. Using techniques of surface micromachining, suspended polysilicon beams and cantilevers were fabricated in various thickness, doped variously, and subjected to different sequences of thermal annealing. The residual stress in the polysilicon and the stress uniformity were then determined by examining vertical deflection of cantilevers with optical interferometry. In addition, the maximum free-standing non-buckled beam length was determined, which can also be related to residual stress. The detailed descriptions and conclusions are contained in two papers which have resulted from this work, which are attached to this report.

SCHEDULING

Research on three activities continued during this period:

- Systems level model of the integrated-circuit fabrication process.
- Mathematical model of an integrated-circuit fabrication facility.
- Simulation and scheduling software.

An important phenomenon was added to the set of models that have been part of our study, namely the fact that many semiconductor fabrication operations are performed on a batch of wafers simultaneously. Examples include oxidation, deposition, ion implantation, and others. While this is seen in other kinds of manufacturing, it is pervasive in this industry.

This feature is important because in order to realize the full capacity of a system, the machine chambers must be as full as possible. There are two reasons for this: (1) it takes as much time to do an operation on one wafer as 100, but 99% of the capacity is wasted if only one wafer is in a chamber that can hold 100, and (2) maintenance must be performed on machines that do deposition operations when the total amount deposited since the last maintenance reaches a given level, independent of the number of wafers that were in the chamber when the depositions took place.

One way to keep the chambers full is to have large lots. However, this is not desirable in a system that has low volume or that has a diversity of products. Another approach is to group together distinct wafers that require the same operations. This leads to more complex modeling and scheduling issues, which we are currently studying and simulating. A prototype scheduler has been built for a single machine.

PUBLICATIONS LIST

O. Z. Maimon and S. B. Gershwin, "Dynamic Scheduling and Routing For Flexible Manufacturing Systems that have Unreliable Machines," *Operations Research*, Volume 36, Number 2, March-April, 1988, pp. 279-293. Also, MIT VLSI Memo No. 87-370, March 1987.

G. H. Prueger, Equipment Model for the Low Pressure Chemical Vapor Deposition of Polysilicon, M.S. Thesis, Department of Mechanical Engineering, MIT, March 1988. Also, MIT VLSI Memo No. 88-485, November 1988.

E. M. Sachs, "Process Model Construction and Optimization using Statistical Experimental Design," Proceedings, Manufacturing International, Atlanta, GA, April 18-19, 1988. Also, MIT VLSI Memo No. 88-442, March 1988.

N. S. Zarghamee, Schema Viewer: A Graphical Representation to Portray the Database Schema of the MIT CAFE System, S.B. Thesis, Department of Electrical Engineering and Computer Science, MIT, May 1988.

T. A. Lober, J. Huang, M. A. Schmidt, and S. D. Senturia, "Characterization of the Mechanisms Producing Bending Moments in Polysilicon Micro-Cantilever Beams by Interferometric Deflection Measurements," Proceedings, IEEE Workshop on Solid-State Sensors and Actuators, Hilton Head, SC, June 6,1988.

R. Jayavant, An Intelligent Process Flow Language Editor, M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, September 1988. Also, MIT VLSI Memo No. 88-475, September 1988.

T.-L. Tung, Boundary Element Techniques for Modeling Thermal Oxidation of Silicon, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, September 1988.

J. Huang, T. A. Lober, M. A. Schmidt, and S. D. Senturia, "The Maximum Free-Standing Length of Polycrystalline Silicon Microbeams," to appear in Proceedings, International Conference on Material and Process Characterization, Shanghai, China, October 1988.

S. B. Gershwin, "Hierarchical Flow Control: A Framework for Scheduling and Planning Discrete Events in Manufacturing Systems," to appear in Proceedings, IEEE Special Issue on Discrete Event Systems. Also, MIT VLSI Memo No. 88-482, October 1988, earlier version appeared as MIT VLSI Memo No. 88-406, July 1987.

INTERNAL MEMORANDA

X. Bai and S. B. Gershwin, "A Manufacturing Scheduler's Perspective on Semiconductor Fabrication," in preparation.

M. L. Heytens and R. S. Nikhil, "GESTALT: An Expressive Database Programming System," submitted to the ACM SIGMOD. Also, MIT VLSI Memo No. 88-484, October 1988.

TALKS WITHOUT PROCEEDINGS

S. D. Senturia, "Microsensors: The Promise and the Problems," 3M MicroTech Symposium, St. Paul, MN, March 30, 1988.

S. D. Senturia, "Critical Issues in Microsensor Design," Toyota Central Research Laboratories, Nagoya, Japan, May 27, 1988.

S. B. Gershwin, "A Hierarchical Framework for Manufacturing Systems," INRIA-LORRAINE, France, June 28, 1988.

S. D. Senturia, "Critical Issues in Microsensor Design," Hitachi Sawa Works, Mito City, Japan, June 1, 1988.

S. B. Gershwin, "A Hierarchical Scheduling and Planning Framework for Manufacturing Systems," EURO Workshop on Production Planning and Scheduling, Paris, France, July 5, 1988.

X. Bai, "A Hierarchical Scheduling and Planning Framework for Manufacturing Systems," CIM-IC Workshop, Stanford University, Palo Alto, CA, August 5, 1988.

S. B. Gershwin, "A Hierarchical Scheduling and Planning Framework for Manufacturing Systems," Symposium on Automated Manufacturing Systems, Rensselaer Polytechnic Institute, Troy, NY, August 29, 1988.

P. Penfield, Jr. and D. S. Boning, "Is the Two-Stage Process-Step Model Valid?," CIM-IC Workshop, Stanford University, Palo Alto, CA, August 4-5, 1988.

E. M. Sachs, "Equipment Modeling Using Smart Response Surfaces," Digital Equipment Corporation, Hudson, MA, August 3, 1988

E. M. Sachs, "Equipment Modeling Using Smart Response Surfaces," CIM-IC Workshop, Stanford University, Palo Alto, CA, August 5, 1988

P. Penfield, Jr., "Future Direction of University Research: Where will we be in 5 years?" CIM-IC Workshop, Stanford University, Palo Alto, CA, August 4-5, 1988.

R. Jayavant, "An Intelligent Process-flow Language Editor," CIM-IC Workshop, Stanford University, Palo Alto, CA, August 4-5, 1988.

D. E. Troxel, "CAFE Application Programs," CIM-IC Workshop, Stanford University, Palo Alto, CA, August 4-5, 1988.

M. Ruf, "DRIFS: Data Retrieval Interface for Fabrication Systems," CIM-IC Workshop, Stanford University, Palo Alto, CA, August 4-5, 1988.

E. M. Sachs, "Equipment Modeling," SEMATECH Center of Excellence first meeting, Westboro, MA, September 29, 1988

EQUIPMENT MODEL FOR THE LOW PRESSURE CHEMICAL VAPOR DEPOSITION OF POLYSILICON

by

George Henry Prueger

Submitted to the Department of Mechanical Engineering
on March 28, 1988 in partial fulfillment of the
requirements for the Degree of Master of Science in
Mechanical Engineering

ABSTRACT

An equipment model has been developed for the low pressure chemical vapor deposition (LPCVD) of polycrystalline silicon in a horizontal tube furnace. The model predicts the wafer-to-wafer deposition rate down the length of the tube. Inputs to the model include: silane flow rates from three injectors, injector locations, locations of and temperatures at three thermocouples, operating pressure, the number of wafers, wafer diameter, the location of the wafer load, and other physical dimensions of the furnace such as tube length, and inner diameter. The model is intended to aid the process engineer in the operation of equipment, including the selection of optimum process parameters and process control based on measured deposition thicknesses. The model is also flexible enough to aid in the design of new equipment.

The one dimensional finite difference model encompasses the convective and diffusive fluxes of silane and hydrogen in the annular space between the wafer load and tube walls. The reaction of silane is modeled, with full account taken of the generation and transport of hydrogen. Kinetic and injection parameters in the model were calibrated using a series of nine statistically designed experiments which varied four parameters over three levels. The model accurately predicts the axial deposition profile over the full range of experimentation and demonstrates good extrapolation beyond the range of experimental calibration. The model was used to predict a set of process parameters that would result in the least variation of deposition rate down the tube. The predicted parameters agree well with experimentally determined optimum conditions.

Thesis Supervisor: Dr. Emanuel Sachs

Title: Assistant Professor of Mechanical Engineering

BOUNDARY ELEMENT TECHNIQUES FOR MODELING THERMAL OXIDATION OF SILICON

by

THYE–LAI TUNG

Submitted to the Department of Electrical Engineering
and Computer Science on June 17, 1988 in partial fulfillment
of the requirements for the Degree of Doctor of Philosophy
in Electrical Engineering and Computer Science

ABSTRACT

This thesis advances boundary element techniques to model thermal oxidation of silicon in two dimensions. At temperatures encountered in thermal oxidation, silicon dioxide flows viscoelastically. A reduced-dimension, generalized boundary element method for modeling such a problem has been developed. With a Laplace transform technique, a viscoelastic kernel function is derived from Kelvin's solution, which is the fundamental solution to linear elasticity. Constant-velocity loading is chosen to operate with a wide range of stress relaxation times. This scheme is capable of replacing boundary element methods developed for slow viscous flow and elastic deformation. The oxidant diffusion problem is solved using a standard potential method for Laplace problems. Generated by oxide growth, stress affects both oxidant diffusion and oxide flow. In particular, it changes the diffusivity of oxidants and viscosity of oxide, rendering the diffusion and flow problems nonhomogeneous. Domain solutions are needed for both processes. A unified initial stress/built-in field formulation has been developed to account for the nonlinear effects, using interior cells that are placed where stress is significant. The interior solutions are realized with an interfacial source method, whereby an area integral for a cell is transformed to a line intgral on the perimeter of the cell. It has been found that kernel functions based on Kelvin's solution are deficient in modeling incompresible materials with a "hole". A correction method that uses a source term placed at the center of the hole has been implemented to overcome the numerical problem.

Thesis Supervisor: Dr. Dimitri A. Antoniadis
Title: Professor of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-475
September 1988

AN INTELLIGENT PROCESS FLOW LANGUAGE EDITOR

Rajeev Jayavant

Abstract

A process flow language allows a user to specify the process used to fabricate integrated circuits on a silicon wafer. By using a flow language, a designer can modify his process more easily and processing equipment can be reconfigured for use with different processes.

While the benefits gained from using a process flow language have been discussed frequently, one major drawback of using a flow language has been overlooked: users must code their process flows in the flow language. This may not seem like a disadvantage, but it is rather difficult to convince people to so something they have never done before. Furthermore, the process must be in a language that faintly resembles Lisp, not a very appealing thought for users whose primary interest is in processing wafers, not programming computers. Thus there is a severe need for some tool to facilitate the coding of processes flows in the process flow language.

Various types of programming aids have been used in the past to facilitate software development: syntax checkers, semantic checkers, preprocessors, and intelligent editors. The process flow editor combines attributes from all of these. The primary difference between the process flow editor and conventional editors is that the flow editor presents the flow to the user in a format that is very different from the format seen by applications accessing the flow.

By using a different format in presenting the flow to the user, most people will not have to learn the Lisp-like syntax of the flow language and can concentrate on what they really want to do -- specify a process flow. The current implementation of the flow editor uses a forms-based interface to present the flow as a collection of nested operations. A forms-based interface is appealing because it facilitates the design of the editor while providing an interface that lab users will recognize from several CAFE applications. The use of forms also allows the flow editor to highlight the decomposition of the process flow into parameterized operations, thereby providing a more informative view of the flow to the user.

Syntax and semantic checking is performed as the user enters the flow. The time required to code a process flow is reduced since many common errors are caught as they are made rather than being discovered at a later time by an interpreter.

ICMPC, Shanghai, October, 1988

The Maximum Free-Standing Length of Polycrystalline Silicon Microbeams

Jiahua Huang[1], Theresa A. Lober, Martin A. Schmidt, and Stephen D. Senturia

Microsystems Technology Laboratories
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

Doubly supported polysilicon microbeams are fabricated by removing a sacrificial oxide layer from under an LPCVD polysilicon film using hydrofluoric acid (HF) as an etchant. Compressive stress in the beams causes buckling at a critical length, which depends on the polysilicon and oxide process history. This paper reports a matrix investigation of the maximum free-standing length, L_m, of polysilicon microbeams as a function of beam width and thickness, variations in doping and annealing conditions, and different sacrificial oxides. The results indicate that the L_m of polysilicon microbeams has no significant dependence on beam width, and increases with beam thickness. Undoped beams which are annealed are more rigid than undoped, unannealed beams. L_m of boron doped beams is smaller than for phosphorus doped beams, and while an annealing cycle increases L_m for boron doped beams, it shows little effect for phosphorus doped beams. A larger L_m is obtained when PSG is used as the sacrificial oxide layer instead of an undoped oxide.

Introduction

Thin films of LPCVD polysilicon have been widely used in integrated circuits. Recently, with the development of surface-micromachining techniques, polysilicon is also becoming an important mechanical material for microsensor and microactuator applications. Along with electrical characteristics, attention is now given to the the effect of the mechanical stress of polysilicon films. Many investigators have developed methods for characterizing the mechanical properties of films. Guckel demonstrated the direct measurement of the film's strain level with experimental determination of the onset of buckling for known geometries [1]. Choi reported the stress effects in boron-implanted polysilicon films using comparative warpage measurements [2]. Howe used relaxed silicon overhangs to determine the compressive stress in polysilicon and α-Si thin films by measuring observed edge deflections, which vary sinusoidally [3].

Doubly supported polysilicon microbeams have found increasing microsensor and microactuator applications [4, 5]. Since compressive stress can cause such beams to buckle when they reach a critical length, the design of these microstructures is constrained by the maximum free-standing length, L_m, for which the microbeam is stable and maintains little deflection. This paper reports the fabrication of doubly supported polysilicon microbeams of various sizes and with varying doping and annealing conditions using conventional IC process techniques, and investigates the dependence of the maximum free-standing length on microbeam thickness and process conditions.

Experimental

Using surface-micromachining techniques, polysilicon microbeams are fabricated on two groups of 4-inch, <100> oriented, lightly doped wafers. On the first wafer set, a 1μm-thick sacrificial thermal oxide is grown at 950°C. The second group of wafers receives a 2 hour, 925°C diffusion cycle with a $POCl_3$ liquid

[1]A visiting scientist from Shanghai Component No.5 Factory of China

diffusion source before the thermal oxide is grown. This produces a phosphorus content of 1.3 wt% in the thermal oxide. The oxide layer on both groups of wafers is patterned and dry etched using CF_4 and CHF_3 to define trenches for creating pedestals for the beams. Pure silane is reacted at 250 mT and 625°C to deposit blanket LPCVD polysilicon films ranging in thickness from 0.25μm to 1μm. Some samples receive a 60 minute, 925°C doping cycle with either a $POCl_3$ or BBr_3 source. The PSG or BSG is removed from the polysilicon surface using 7:1 buffered HF. Then two sample groups containing both doped and undoped films are annealed in an N_2 ambient either at 950°C for 30 min (low T anneal), or at 1100°C for 20 min (high T anneal). After anisotropically dry etching the polysilicon films with CCl_4 to pattern the microbeams and remove the polysilicon from the backside of the wafers, the sacrificial oxide film is undercut using 1:1 HF:H_2O to release the suspended structures. Once the polysilion bridges are freed they are delicate, requiring careful handling during rinsing and drying. The samples are gently rinsed first with DI water and then methanol, since it has a lower surface tension coefficient than water. Up to the removal of the sacrificial oxide, the fabrication process is compatible with standard IC process equipment and techniques. Each wafer contains 60 dies, and each die contains 81 microbeams of 5μm, 10μm and 20μm widths. The beams range in length from 10μm to 50μm in 5μm increments, from 60μ to 200μm in 10μm increments, and from 200μm to 300μm in 20μm increments. The samples are inspected for the maximum free-standing length, L_m, using a Leitz Model SMLUX inspection microscope and Hitach Model S-806 low voltage scanning electron microscope. L_m is defined as the longest observed beam length before the length which buckles. The vertical deflection magnitude and profile of the unbuckled microbeams is determined by interferometric deflection measurement techniques [6].

Results and Discussion

All of the 60 duplicated patterns on each 4 inch wafer are inspected. At lengths less than the buckling length, L_c, the phosphorus doped beams, whether unannealed or annealed, are undeflected. While undoped, unannealed beams less than L_c deflect slightly upward, boron doped, unannealed beams exhibit large, upward deflections at lengths less than L_c. This may be caused by a doping dependent stress nonuniformity through the beam thickness. L_m is seen to be the same for 5μm, 10μm and 20μm wide beams across a wafer, and from wafer to wafer, suggesting that there is little dependence of L_m on beam width for this range of widths. Table 1 summarizes the dependence of L_m on beam thickness for undoped and phosphorus doped beams. Figure 1 and 2 illustrate trends for the dependence of the average beam buckling length, L_c, on beam thickness. L_m and L_c are larger for undoped, annealed beams than for undoped, unannealed beams, with the higher annealing temperature producing the longest unbuckled, undoped beams. L_m and L_c are constant for phosphorus doped beams, whether unannealed or annealed, suggesting that the beams are in a stable equilibrium state after the doping cycle.

Table 1. L_m in Various Beam Thickness t and Process Conditions

process conditions	L_m			
	t = 0.25 μm	t = 0.52 μm	t =0.8 μm	t = 1.07 μm
undoped, unanneal	20-25 μm	30-40 μm	45-50 μm	50 -70 μm
undoped, low T anneal	20-25 μm	35-45 μm	50-70 μm	70-80 μm
undoped, high T anneal	25-30 μm	45-50 μm	70-80 μm	80-90 μm

process	L_m			
	t = 0.2 μm	t = 0.46 μm	t = 0.75 μm	t = 1 μm
P doped, unanneal	15-20 μm	35-40 μm	50-70 μm	70-100 μm
P doped, low T anneal	15-20 μm	35-45 μm	50-70 μm	70-100 μm
P doped, high T anneal	15-20 μm	35-40 μm	50-70 μm	70-100 μm

The increase in buckling length for thin beams beyond that predicted by the thicker beams may be explained by a simple model that accouOBnts for compliance of the step-up beam supports. This model will be reported separately. If compliance of the beam support is ignored, a good approximation for the residual strain level, ϵ_c, of the 1μm thick beams at the critical buckling length is estimated by Euler buckling theory [7] for each process condition as

$$\epsilon_c = \frac{\pi^2 t^2}{3Lc^2}, \tag{1}$$

and

$$Lc = \frac{\pi t}{\sqrt{3\varepsilon_e}}, \tag{2}$$

where t is the polysilicon film thickness. Based on Eq. (2), Fig. 1, and Fig. 2, ε_e and L_e for each of the polysilicon process conditions are shown in Table 2. These values are in good agreement with those previously reported by Guckel [1].

Table 2. Buckling Length Lc and Strain ε_e

process conditions	undoped			P doped
	unanneal	low T anneal	high T anneal	
strain ε_e	1.07×10^{-3}	0.67×10^{-3}	0.48×10^{-3}	0.41×10^{-3}
buckling length Lc (μm)	$\pi t/\sqrt{3.2 \times 10^{-3}}$	$\pi t/\sqrt{2.0 \times 10^{-3}}$	$\pi t/\sqrt{1.4 \times 10^{-3}}$	$\pi t/\sqrt{1.2 \times 10^{-3}}$

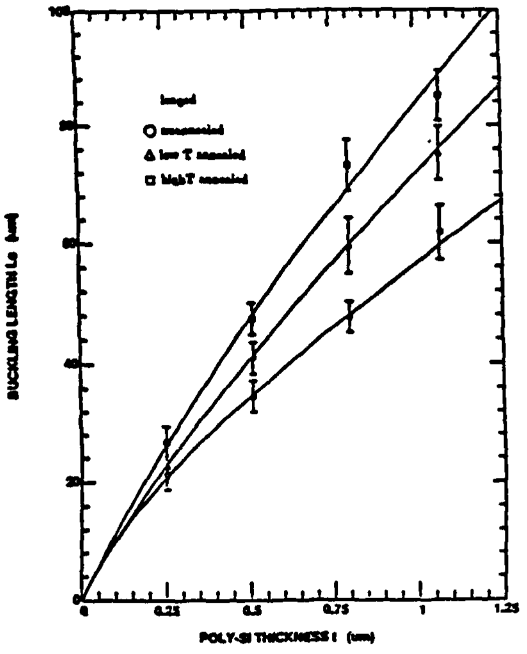

Figure. 1. Buckling length as a function of film thickness for polysilicon doubly-supported undoped beams with or without annealing

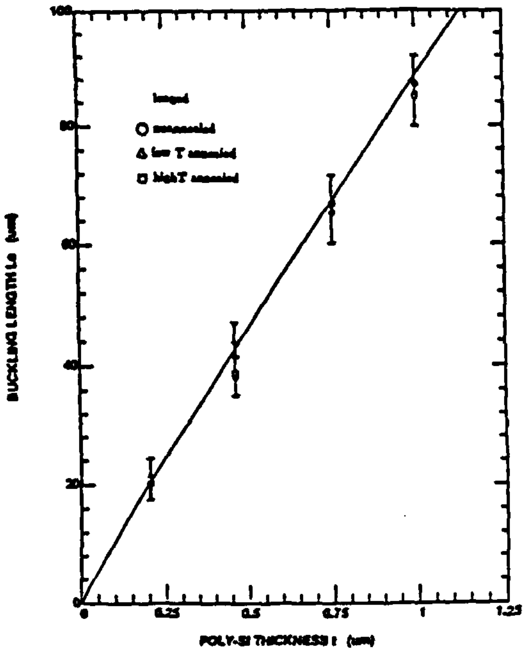

Figure 2. Buckling length as a function of flim thickness for polysilicon doubly-supported phosphorus doped beams with or without annealing

Table 3 compares L_m for boron and phosphorus doped microbeams, 0.46 or 1μm-thick, unannealed and annealed. Although both doping cycles are completed at the same temperature and for the same time, the unannealed, boron doped samples exhibit a shorter L_m than the unannealed, phosphorus doped samples. After annealing, both boron and phosphorus doped samples display similiar free-standing lengths, which are the same for phosphorus doped, unannealed beams. This suggests that phosphorus doping has a stronger effect on grain growth than boron doping. Mei mentions variable grain growth enhancement with polysilicon phosphorus doping, and little growth with boron doping [8]. Mandurah reports that phosphorus has a strong tendency to segregate at grain boundaries while boron does not [9]. The phosphorus impurities segregated at the boundaries during the doping cycle may hinder boundary migration, thereby inhibiting further polysilicon grain growth during the following anneal cycle.

Also compared in Table 3 are the free-standing lengths for microbeams fabricated on undoped or phosphorus doped oxide. L_m is dramatically longer for beams fabricated on the phosphorus doped oxide than for beams on undoped oxide. This length increase implies that the phosphorus in the underlying oxide effects both the grain size and doping profile in polysilicon, therby reducing the level of stress in the film. The high temperature annealing cycle is more effective at increasing the free-standing length of thicker beams than thinner beams, indicating that more redistribution of dopant occurs in thicker beams during the annealing cycle. Alternatively, the hinge compliance of the thinner beams' supports may mask the effects of the annealing cycle on the free-standing length.

Table 3. Lm in Different Doping conditions and Sacrificial Oxide Layers

sacri. oxide	poly-si	anneal	Lm	
			$t = 0.46\ \mu m$	$t = 1\ \mu m$
undoped	B doped	unanneal	25-30 μm	50-60 μm
undoped	B doped	high T anneal	35-40 μm	70-120 μm
undoped	P doped	unanneal	35-40 μm	70-100 μm
undoped	P doped	high T anneal	35-40 μm	70-100 μm
P doped	P doped	unanneal	60-70 μm	100-160 μm
P doped	P doped	high T anneal	50-60 μm	120-180 μm

Conclusion

Doubly supported polysilicon microbeams are fabricated with conventional IC process tecniques. The maximum free-standing length of the beams is seen to not be a function of width, but is dependent on beam thickness. Before buckling, undoped and boron doped beams which are not annealed deflect slightly upward, while phosphorus doped beams, and undoped and boron doped beams which are annealed are undeflected. Accordingly, the annealing cycle significantly increases L_m for undoped and boron doped beams, but does not significantly alter L_m for phosphorus doped beams. For the phophorus doped beams, Lm $< \pi t/\sqrt{1.2 \times 10^{-3}}$; undoped and unannealed beams, Lm $< \pi t/\sqrt{3.2 \times 10^{-3}}$; undoped beams with a low temperature anneal, Lm $< \pi t/\sqrt{2.0 \times 10^{-3}}$; and undoped beams with a *high temperature anneal*, Lm $< \pi t/\sqrt{1.4 \times 10^{-3}}$. Use of PSG as the sacrifical oxide layer significantly increases L_m, suggesting that more rigid microbeams may be fabricated using this as the micromachining spacer layer.

Acknowledgments

We acknowledge many useful discussions and help on this subject with R. T. Howe, H. Guckel, P. A. Maciel, and M. T. Schroth. This work was sponsored in part by DARPA , under contract 87-SP-080; additional support of NSF (TAL), under contract ECS-8614328 and of the 3M sensor fellowship (MAS) is gratefully acknowledged.

REFERENCES

1. H.Guckel, T.Randazzo, and D.W.Burns, J.Appl.Phys. 57(5), 1 March, 1671(1985).
2. M.S.Choi and E.W.Hearn, J.Electrochem.Soc., vol.131, No.10, 2443(1984).
3. R.T.Howe and R.S.Muller, J.Appl.Phys. 54(8), August, 4674(1983). 1603(1980).
4. R.T.Howe and R.S.Muller, IEEE Transactions on Electron Devices, Vol.ED-33, No.4, 499(1986).
5. Y.Tai and R.S.Muller, The 4th International Conference on Solid-State Sensors and Actuators, 360(1987).
6. T.A.Lober, J.Huang, M.A.Schmidt, and S.D.Senturia, IEEE Workshop on Solid-State Sensors, Hilton Head, June 1988.
7. Gere and Timoshenko, Mechanics of Materials, Wadsworth Inc., Belmont California, 1984.
8. L.Mei, M.Rivier,Y.Kwark and R.W.Dutton, J.Electrochem.Soc., vol.129, No.8, 1791(1982).
9. M.M.Mandurah, K.C.Sarawat, C.R.Helms and T.I.Kamins, J.Appl.Phys., 51(11), 5755(1980).

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-482
October 1988

HIERARCHICAL FLOW CONTROL: A FRAMEWORK FOR SCHEDULING AND PLANNING DISCRETE EVENTS IN MANUFACTURING SYSTEMS

Stanley B. Gershwin

Abstract

This paper discusses the synthesis of operating policies for manufacturing systems. These are feedback laws that respond to potentially disruptive events. We develop laws that are based on realistic dynamic programming models which account for the discrete nature of manufacturing and which are computationally tractable.

These scheduling and planning policies have a hierarchical structure which is systematically based on the production process. The levels of the hierarchy correspond to classes of events that occur with distinct frequencies. At each level, feedback laws select (1) times for the controllable events whose frequency class is treated at that level, and (2) frequency targets for much higher frequency controllable events.

In this hierarchy,

(1) Most calculations deal with expected rates of high frequency activities, conditioned on current states of low frequency activities. There is an important relationship between conditional expected rates at different levels.

(2) Rates are constrained because only one activity (e.g., production operation) can take place at one resource (e.g., machine) at any time.

(3) Rates are found at each level according to a dynamic programming problem for which there exists good approximate solutions. Times for controllable events are chosen to agree with those rates.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138

GESTALT: An Expressive Database Programming System*

Michael L. Heytens[†]
Rishiyur S. Nikhil[‡]
Massachusetts Institute of Technology

June 23, 1988

Abstract

Many new database applications require computational and data modelling power simply not present in conventional database management systems. Developers are forced to design complex encodings of complex data into a limited set of database types, and to embed DML commands into a host programming language, a notoriously tricky and error-prone enterprise.

In this paper, we describe the design and implementation of GESTALT, a system and methodology for organizing and interfacing to multiple heterogeneous, existing database systems. Application programs are written in a supported programming language (currently C and Lisp) using high-level data and control abstractions native to the language. The system is flexible in that the underlying database systems can easily be replaced/upgraded/augmented without affecting existing application programs.

We also describe our experience with the system: GESTALT has been in daily operational use at MIT for over a year, supporting an information system for CAF, a research facility for the automation of semiconductor fabrication.

1 Introduction

Database management systems have traditionally been used in administrative applications to provide efficient access to large data sets, preserve data integrity and consistency, and to control access. Recently, however, database systems have been applied to diverse new domains such as VLSI computer-aided design and voice and image processing.

A straightforward application of conventional database technology (*e.g.*, current relational systems) to these nontraditional areas has met with limited success. A key problem is the difficulty inherent in expressing complex objects and operations in terms of relations and relational operators. Moreover, conventional systems provide no general mechanism for abstracting over a given set of operations or data, thus all representation and manipulation must be encoded directly in terms of primitive constructs. In many instances, such a relational representation is cumbersome, leading to abstruse application logic.

*This research was sponsored in part by the Defense Advanced Research Projects Agency (DoD) through the Office of Naval Research under Contract Number N00014-85-K-0213.

[†]Room 36-667, MIT, 77 Massachusetts Avenue, Cambridge MA 02139, USA; Phone: (617)-253-7811; Arpanet: heytens@caf.mit.edu

[‡]MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA; Phone: (617)-253-0237; Arpanet: nikhil@xx.lcs.mit.edu

To obtain adequate expressivity and abstraction, application developers typically embed query language commands into a host programming language. An embedded interface, however, is notoriously tricky and error-prone. Programmers must contend with both database and programming language concepts, *e.g.*, two mismatched type systems, two error/exception mechanisms, two types of control structures, *etc.* These inherent difficulties make it accessible only to database experts.

A number of research efforts are currently underway aimed at developing more expressive database systems [1,4,7,11,16]. Still, conventional systems are likely to remain the commercially-available state of the art for a number of years. Thus, a natural question is: How can we more effectively organize and interface to today's database systems.

This paper presents one possible organization, developed as part of the integrated circuit computer integrated manufacturing (IC-CIM) effort at the Massachusetts Institute of Technology. Our system, called GESTALT, is currently employed by CAFE (Computer-Aided Fabrication Environment) a sophisticated information system supporting the many facets of IC manufacturing. From a database perspective, CAFE is an especially challenging domain due to the diversity of information it manages, and the many complex programs it supports for the acquisition and manipulation of semiconductor manufacturing data. (For a discussion of IC-CIM data, see [8].)

One of the main motivations behind GESTALT was the need for an application development paradigm that did *not* require database and/or programming language expertise. The implied requirement for simplicity, however, had to be tempered with enough expressive power to permit a conceptually natural representation of the complex structures found in the domain of semiconductor manufacturing.

Another aim of GESTALT was *database independence*, *i.e.*, the model visible to applications was to be independent of the actual underlying database management system or systems. This capability provides a great deal of implementation flexibility as well as an elegant way to support "cross system" queries. The latter (similar to [3,15]) is especially useful in a manufacturing setting, *e.g.*, it can provide applications with a unified view of design, manufacturing, test, and product sales data, even though this information may physically reside at multiple heterogeneous database systems. Without this kind of support, application programmers must query component databases directly (after first mastering all the necessary interfaces!) combining partial results to form the desired answer. This process is complicated, tedious, and error-prone.

Finally, in a design and manufacturing environment, an effective data management system *must* seamlessly encompass a variety of CAD/CAM tools; it is simply not feasible to rewrite data and operations supported by these tools into a single monolithic DBMS. Thus, the GESTALT data model had to support the databases and procedural primitives of such tools.

We begin by presenting an overview of GESTALT in the next section. Section 3 describes the application interface, illustrating the programming model with sample applications. In Section 4 we describe the internal organization of GESTALT, and finally, we conclude with a discussion of the experience gained from using GESTALT in CAFE— a large, complicated system.

2 Overview of GESTALT

In essence, GESTALT is a layer of abstraction which shields application programs from underlying database systems. The application programming paradigm consists of a supported language (currently C and Common Lisp) and a set of abstractions for accessing and manipulating persistent data. An important point is that these abstractions are constructed using mechanisms native to the host language, thus application developers do not have to contend with the programming language–database dichotomy typically found in conventional approaches.

The two supported languages have their strengths and weaknesses. Many new database applications have an artificial intelligence component, for which Lisp is the preferred language. Also, the interactive nature of Lisp is convenient for posing *ad hoc* interpreted queries (analogous to query interpreters in conventional systems). The C interface is available for lower-level tasks or for applications requiring the compactness and speed of compiled C code.

The basic architecture of GESTALT consists of a software system running atop existing heterogeneous databases. GESTALT is not itself a full-fledged DBMS, rather it is a mechanism for logically integrating disparate data managers. This integration enables applications to view data in terms of a unified "global" schema. While GESTALT supports both read-only and update queries, it does not allow dynamic schema modifications. Such updates require changes to the system data dictionary, which must be "hand-coded"[1] by the database administrator (DBA).

Figure 1 illustrates both a conventional application architecture (a) and the organization of applications written in GESTALT (b). In conventional systems, applications are written using an embedded query language interface (*e.g.*, QUEL in C); the resulting program is then transliterated by a pre-processor and finally compiled. Conceptual entities and operations must be encoded and manipulated explicitly in terms of primitives supported by the data model, leading to much complexity in coding applications.

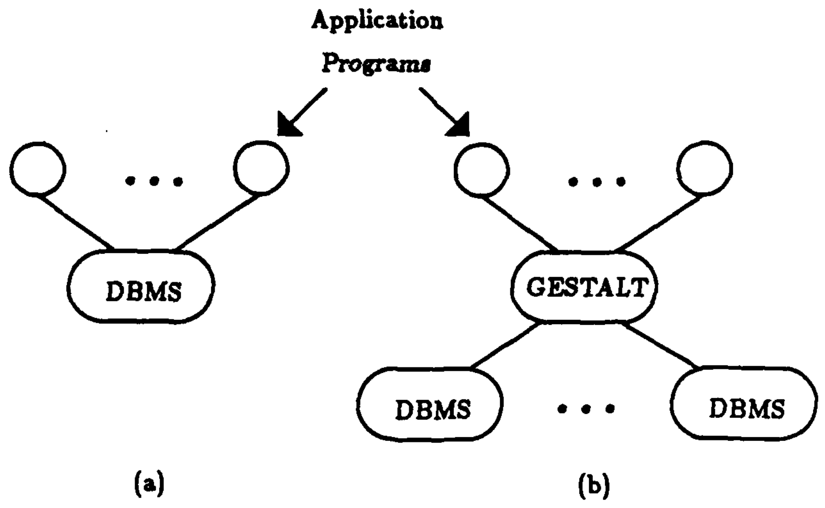

Figure 1: Structure of (a) conventional systems and (b) GESTALT.

In GESTALT, applications are also coded in a host programming language (C or Common Lisp.) Application programmers determine how to query and update persistent data by examining a specification detailing the interface to a collection of *procedural, data,* and *iteration abstractions.* (For a good discussion of the techniques of abstraction and specification, see [10].) The abstractions

[1]A structured schema-interface tool has been implemented which makes updating the data dictionary also straightforward.

which describe persistent data are object-oriented in nature[2], enabling applications to view domain data at a level commensurate with the actual real-world structure.

GESTALT provides a rich set of primitive data types and explicit support for null objects. Without system support of this kind, programmers often resort to nonstandard representations of null values and awkward encodings of, say, temporal and engineering data. In a large project where communication between developers is difficult, this can lead to inconsistencies and incompatibilities between applications.

Many of the procedural abstractions supported by the system are higher-order (*e.g.*, maps and filters). Operations of this sort encourage a clear and elegant style of programming, allowing concise expression of a wider range of computational processes [12,18].

3 The Application Interface

The interface visible to application programs is similar, in many respects, to the functional data model [12,14]. A GESTALT schema consists of a specification of entity or object types, and a collection of operations defined on instances of those types. This information is available to application programmers in two different forms. The first is a textual specification (maintained by the system) which provides an overview of each object type and a description of the effects, modifications, and requirements of the associated operators. The second is the data dictionary; all query facilities supported by GESTALT are available for accessing schema information stored in the system catalogs. The interpreted Lisp interface provides a convenient mechanism for browsing through meta-data of this form.

At the end of this section we present an example application as a concrete illustration of the programming model. We first, however, examine various aspects of the interface in more detail.

3.1 Object Types

All data in a GESTALT system are viewed as abstract types, *i.e.*, each abstract type specifies an object type, a list of named typed slots (which record attributes and relationships) and the set of procedures available for manipulating objects of that type. This is a very powerful tool, enabling the DBA to add new kinds of data objects cleanly, effectively extending the application programmer's virtual machine.

Operations in data abstractions can be divided into the following categories:

selectors — procedures to select an object component
mutators — procedures to update or delete an object
constructors — procedures to create objects
generators — procedures to generate null objects and iterators

A GESTALT system contains both *pre-defined* and *domain-specific* abstract types. The latter are abstractions of objects present in the application domain, while the former encompass all the built-in types, including an extended set of primitive entity types, abstract types for the data dictionary, and list- and tuple-structured types.

[2]In the following sense: Real-world entities are modelled as "objects" of various abstract types, objects can "contain" other objects, and objects can share other contained objects. We do not model inheritance.

It is important to realize that even though a particular type may be built-in, its behavior is still specified only in terms of the operators defined on instances of that type, so that the user sees no semantic difference between built-in and user-defined types. Hiding the representation of built-in types enables the system to perform run-time type checking. It also eliminates the possibility that applications depend on particular encodings of values.

3.1.1 Pre-defined Types

The set of primitive types supported was designed specifically to facilitate modelling the temporal and engineering data commonly found in manufacturing and design domains. The *atomic* members of this set are:

TEXT	variable length array of bytes
INTEGER	fixed point number
FLOAT	floating point number
BOOLEAN	"true" or "false"
DATE	month, day, and year
TIMEOFDAY	hours, minutes, and seconds
TIMEDURATION	elapsed time in hours, minutes, and seconds.

Also pre-defined are several *inexact* and *interval* types. The inexact types record a *value* and an *uncertainty;* interval types consist of an *upper* and *lower* bound. These provide a convenient means of encoding engineering and scientific data, *e.g.*, the thickness of a layer of epitaxial silicon (say 1250 $\pm$ 30 Å) can be naturally expressed as an INTEGER_INEXACT with the appropriate value and uncertainty.

To provide applications with a convenient mechanism for grouping related objects, GESTALT provides the structured types LIST and NTUPLE (*n-tuples*). The former is a homogeneous sequence of objects of unspecified length, while the latter is a heterogeneous collection of a fixed number of objects.

GESTALT provides operators to map between pre-defined types and host programming language values. These routines are needed because the representations of even atomic GESTALT types are hidden from application programs. For example, the following Lisp procedure prints the id, crystal orientation, and sheet resistivity of a specified wafer by applying dbl-coerce— an operator which coerces atomic built-in data objects to corresponding Lisp values— to the desired wafer attributes.

```
(defun wafer-print (w output-stream)
  (let ((wd (wafer-waferdesc w)))
       (format output-stream "wafer ~A: <~A>, ~A ohm/square ~%"
               (dbl-coerce (wafer-id w))
               (dbl-coerce (waferdesc-orientation wd))
               (dbl-coerce (waferdesc-sheet-resistivity wd)))))
```

The data-dictionary is represented as a pre-defined set of abstract types, complete with selectors and generators, so that the same computational model can be used to peruse it. These include:

TYPE	built-in and domain-specific types
ATTRIBUTE	object components or slots

OPERATOR - structure for describing operators
OPERATORARG - formal operator arguments

3.1.2 Domain-Specific Types

It is the responsibility of the DBA to augment the data dictionary accordingly with the domain-specific abstract types for a given domain. We have implemented many tools to facilitate this activity.

An example specification is shown in Figure 2. Such specifications are automatically generated by the system from information stored in the data dictionary and are used by application developers. The first three operations (machine_name, machine_available, and machine_labuser) are selectors; they take objects of type MACHINE and return an object corresponding to the desired property or attribute. The next two (modify_machine_available and modify_machine_labuser) provide a mechanism for mutating or modifying objects. Entities of type MACHINE are created by constructor create_machine; this operator takes a name, labuser, available triple and returns a newly-created machine. Finally, generators null_MACHINE and machine_iterator produce a null machine and machine iterator, respectively.

Another example from the CAFE system is the type WAFER, with components: a TEXT id, a wafer description (WAFERDESC, which records dopant, crystal orientation, sheet resistivity, *etc.*), and a BOOLEAN flag indicating whether or not a layer of epitaxial silicon is present.

3.2 Procedural Abstractions

GESTALT's procedural abstractions include generic operators for LISTs and NTUPLEs, as well as common computational processes (such as maps and filters) packaged as higher-order procedures. For example, the following Lisp code prints all wafers from a named lot (a logical set of wafers) with resistivity values below a certain threshold:

```
(map (lambda (w) (wafer-print w t))
     (filter (lot-wafers (lot-with-name name))
             'low-resistivity?))
```

where name identifies the desired lot. Note that map and filter take arbitrary procedures as arguments. Applications can also make use of the procedure mechanism of the host language to create new procedural abstractions.

In addition, a variety of others procedures are supported, *e.g.*, to copy and (extensionally) compare data objects, a simple mark-and-release memory management scheme, *etc.*

3.3 Iterators

GESTALT iterators offer a convenient and space-efficient way to examine instances of a particular type, and are inspired by CLU [10]. Iterators serve the function of "retrieve loops" or "record cursors" found in conventional embedded query languages. For example, the following fragment of C code illustrates how an application might operate on all machines:

Overview

A MACHINE is a piece of equipment used in the fabrication of integrated circuits. Attributes name, labuser, and available are maintained for each machine, recording the machine's name, a list of labusers waiting to use the machine, and the current availability, respectively. Only the latter two are mutable.

Operations

```
TEXT machine_name(m)
MACHINE m;
   effects Returns machine name.

BOOLEAN machine_available(m)
MACHINE m;
   effects Returns machine availability.

LIST machine_labuser(m)
MACHINE m;
   effects Returns list of labusers waiting to use machine.

MACHINE modify_machine_available(m, available)
MACHINE m; BOOLEAN available;
   modifies m.
   effects Sets availability of m to available.

MACHINE modify_machine_labuser(m, labuser)
MACHINE m; LIST labuser;
   modifies m.
   effects Sets labuser list of m to labuser.

MACHINE create_machine(name, available, labuser)
TEXT name; BOOLEAN available; LIST labuser;
   effects Returns newly created machine.
   requires Name attribute must be unique.

MACHINE null_MACHINE()
   effects Generate null machine.

ITERATOR machine_iterator()
   effects Returns a machine iterator.
```

Figure 2: MACHINE data abstraction.

```
...
while ( BOOLEANtoi( iter_more( machines ))) {
    current_machine = iter_current( machines );
    ... computation involving current_machine ...
}
```

where machines is an iterator, and iter_more and iter_current test for an exhausted iterator and return the head of an iterator, respectively.

3.4 Type and Exception Checking

All GESTALT operators perform dynamic type checking. While static type checking could (in principle) be performed, an unobtrusive implementation would require modifications of the C compiler and Lisp interpreter, which we wished to avoid.

Certain operators also perform null checks at runtime. Generally speaking, operators that examine object components are strict with respect to null objects, whereas operators which do not examine components are nonstrict. For example, a null wafer can be freely included in a list or tuple; a null exception is detected only when an attempt is made to, say, select the id component.

Routines that detect type or null errors generate a run-time warning message and return a null object consistent with their range type. If an attempt is made to coerce a null object to a host language type, then a pre-defined value is returned, after printing a suitable warning message.

In the case of domain-specific abstractions, the DBA is free to augment the run-time checking with additional integrity constraints (or *invariants* in programming language parlance.)

3.5 Object Persistence

In GESTALT, all objects of a user-defined type persist, whereas objects of atomic, list and tuple types are ephemeral (unless they are part of a persistent type). For example, the constructor create_machine creates a persistent machine object, whereas itoINTEGER – which coerces a C integer value to a GESTALT INTEGER– returns a heap-based object whose lifetime is bounded by the duration of the enclosing program.

Associated with each user-defined type is a single persistent extent. The system automatically updates these extents in response to creation and deletion of objects of the appropriate type. Applications can easily examine the contents of an extent *via* an iterator.

3.6 A Sample Application

As an illustration, we present a simple C application. The program, which utilizes the data abstraction of Figure 2, informs the next laboratory user waiting to use a particular piece of processing equipment when he or she is free to do so. The code is shown in Figure 3; for brevity, it assumes (1) there are always waiting users for a machine, and (2) a procedure send_mail_msg, which does the obvious thing.

The application constructs a machine iterator (machines) and then uses it to examine the availability of each machine. If a machine is available, a message is sent to the next labuser, and the availability and waiting list are updated. Finally, the iterator is deallocated.

```
#include "specification.h"

main()
{
    ITERATOR machines = machine_iterator();
    MACHINE current_machine;

    while ( BOOLEANtoi( iter_more( machines )))
    {
        current_machine = iter_current( machines );
            /* grab machine at head of iterator */

        if ( BOOLEANtoi( machine_available( current_machine )))
        {
            LIST labusers = machine_labuser( current_machine );

            send_mail_msg( machine_name( current_machine ), head( labusers ));
                /* inform next user */

            modify_machine_available( current_machine, itoBOOLEAN( 0 ));
            modify_machine_labuser( current_machine, tail( labusers ));
                /* modify availability and labuser list */
        }
    }
    iter_free( machines );
}
```

Figure 3: Sample C application program.

Notice that nowhere is the programmer forced to deal with representational issues; one need only concentrate on the logic of the application at hand.

Note also that one does not have to annotate or flag the database commands in a program (*e.g.*, the ## of embedded QUEL and embedded SQL's **EXEC** [6].) The programming language and database have been integrated into a single framework, so that the actual encoding of a task closely matches the corresponding abstract computational process, resulting in programs that are perspicuous and easily modifiable.

Finally, since programs have no way of determining where data actually reside, the DBA is free to change the underlying database systems without compromising the correctness of application programs. (They will have to be relinked, however.)

4 The Implementation of GESTALT

Our implementation relies heavily on the proven software engineering principles of abstraction and modularization. First, such techniques constitute a programming methodology which is effective at controlling the complexities inherent in any large programming effort. Second, since the DBA was expected to modify the implementation (*e.g.*, to add a new component database) a clear, well-partitioned internal structure was deemed essential.

In this section we present the internal structure of GESTALT, including a discussion of the steps required to modify the system. We conclude by describing the system configuration currently in use by CAFE.

4.1 System Architecture

The organization of GESTALT (Figure 4) is similar to that of Multibase [15]. The schema architecture consists of a GESTALT global schema, and a GESTALT local schema-local DBMS schema pair for each component system. As in Multibase, the latter insulates the global system from local database details, allowing it to be structured in a clean and extensible manner. All component-specific details are confined to translator modules (one per underlying database) which are responsible for mapping operations on GESTALT local schemas to local database operations.

GESTALT, however, is responsible for translating global requests into operations (in terms of GESTALT local schemas) on one or more of the underlying databases. This translation is performed by the GESTALT evaluator, which is coded in a manner independent of the number and nature of the underlying database systems. All such dependencies are recorded in a dispatch table, enabling the DBA to extend the system in a straightforward way. This table-driven approach is possible because the evaluator assumes a standard interface to each of the underlying systems.

The system is made available to applications in the form of a library of procedures. This library is either linked into compiled application programs (*e.g.*, in C), or made available to an interpreter (*e.g.*, in Lisp).

4.2 Modifying the System

Due to the modular implementation, modifying the system is relatively straightforward. Modifications are required when the underlying database configuration changes or when abstractions are added, deleted, or modified.

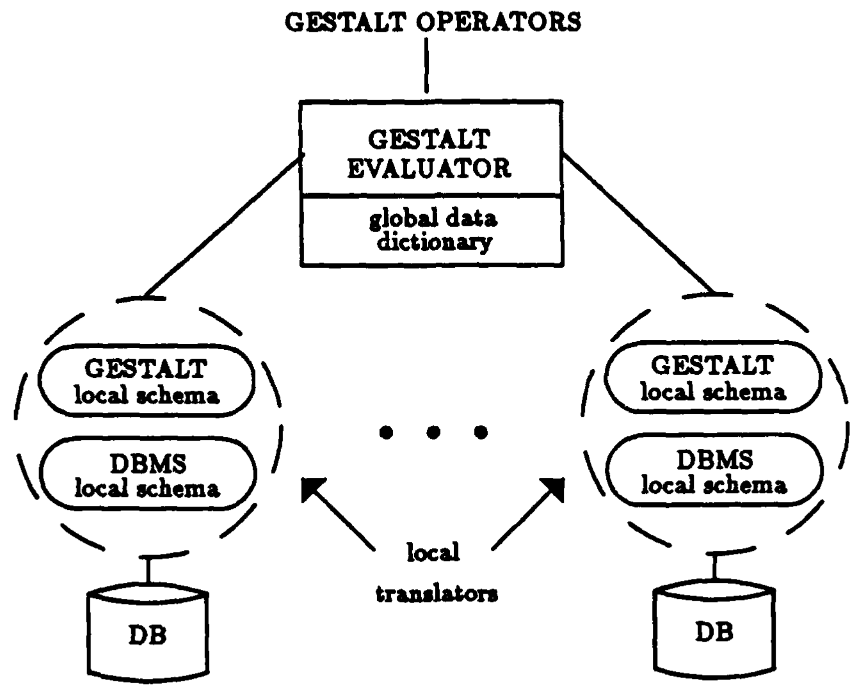

Figure 4: GESTALT system structure.

Adding a database to the system requires updating the dispatch table used by the evaluator, recording the new database in the data dictionary, and incorporating the corresponding implementation module into the system.

Altering the set of procedural abstractions supported by the system simply entails adding, deleting, or modifying the appropriate library routine. When updating data abstractions, however, things are a bit more involved. In the simplest case, modifications of existing data abstractions only involve source-level changes to routines in the corresponding abstraction module (*e.g.*, performing additional integrity constraint checks prior to invoking the evaluator). Modifications that are tantamount to changing the GESTALT schema require adding or deleting abstraction operators, entering new or updated information into the data dictionary and (possibly) changing the schema at one or more of the underlying database systems.

As an illustration, consider the steps required to introduce type LABUSER, which captures the notion of a certified laboratory user. The first step is to add the approriate type, attribute, and operator information to the data dictionary. Secondly, suitable logical structures must be created at one or more of the underlying databases. Finally, a LABUSER specification and implementation module are generated (based on information in the data dictionary) and incorporated into the system.

4.3 Sample Configuration, and Experience

GESTALT has been the database manager of the CAFE system since January, 1987. Since then, approximately 50 programs totalling some 30,000 lines of GESTALT code have been written by several applications programmers. An example is an electronic machine reservation system (EMR) which provides laboratory users with a convenient mechanism for reserving time on processing machines. EMR eliminates paper sign-up sheets from the clean-room, a potential source of contaminating particles. The system is regularly used in the microtechnology laboratories at MIT.

An early database configuration used by CAFE had GESTALT running atop three component databases: University INGRES [17], PRELUDE [13], and a home-brewed system. Each of these databases made an important contribution to the overall system. INGRES provided a reliable, fully-functional data manager. However, due to its lengthy startup time, it was unable to support applications requiring fast, simple data access. To remedy this, PRELUDE— a lightweight ASCII-based DBMS— was incorporated. Finally, the home-brewed system was employed to store the large, variable-length data objects commonly found in the IC manufacturing domain. Storing such objects in either of the other systems would have been difficult or impossible.

The current configuration utilizes only two databases: commercial INGRES from Relational Technology, Inc.[9] and WiSS, the Wisconsin Storage System [5]. Most CAFE data are stored in INGRES; WiSS, because of its support for variable-length data items, handles those objects not easily captured by the relational data model. For example, WiSS is used to store voluminous execution "traces" of program-like process flow descriptions.

Despite the wholesale database changes to move to the current configuration, application programs remained unchanged— they only needed relinking. Thus systems using GESTALT are not locked in to a particular database system; they are free to incorporate newer, more powerful data managers as they become available.

CAFE developers may still write applications using INGRES or WiSS directly, *e.g.*, to take advantage of INGRES' report-writer tools.

5 Conclusions and Future Work

We have described GESTALT, a system which offers an expressive application programming paradigm in which the database and programming language have been integrated into a single framework. The system provides uniform access to existing heterogeneous databases.

While GESTALT offers a number of benefits, there are some difficulties and limitations inherent in its approach. One obvious limitation is the lack of a dynamic data definition capability. Because GESTALT does not have control over the entire programming environment, this is simply not feasible in the system. In general, this problem is very difficult in a multiple heterogeneous database environment (it is analogous to view updating in conventional systems).

Another limitation is the lack of global concurrency control and recovery. Although all interactions with component databases are atomic, the system contains no general mechanism for grouping several GESTALT operations together into a transaction.

GESTALT does no query optimization of its own, thus certain kinds of queries execute inefficiently. For example, tasks implementing the equivalent of a multiway join are not supported well, unless there is a data abstraction that executes it entirely within a component database system.

Our experience to date with the system has been encouraging: CAFE application developers have responded very positively. Despite the wide range of programmer experience (from novices to veteran software engineers) all have commented on how easy the model was to understand, and how quickly they were able to produce sophisticated, working applications.

An immediate need that we plan to address soon is a coherent access control strategy. Currently applications using GESTALT must rely on component database and/or operating system protection mechanisms.

Our longer-term plans involve extending GESTALT so that it is a computationally complete, stand-alone system, incorporating its own persistent objects. We are currently exloring a functional database programming language with an *immutable* or *functional* database [11]. We feel that this combination offers an expressive system for high level applications programming, admits much parallelism (for high performance), and facilitates the management of historical data.

Acknowledgements

We would like to acknowledge the contributions made by Michael McIlrath and Rajeev Jayavant to both the design and implementation of GESTALT. Duane Boning, Paul Penfield, and Donald Troxel also made many helpful suggestions.

References

[1] Albano, A., Cardelli, L., and Orsini, R. *Galileo: A Strongly Typed Interactive Conceptual Language.* Technical Report 83-11271-2, Bell Laboratories, 1983.

[2] Atkinson, M.P., Chisholm, K.J., and Cockshott, W.P. Ps-Algol: An Algol with a Persistent Heap. *SIGPLAN Notices* 17(7):24-31, July, 1981.

[3] Brodie, M.L., Blaustein, B., Dayal, U., Manola, F., Rosenthal, A., CAD/CAM Database Management. *Database Engineering,* Vol. 7, No. 2, IEEE, June 1984.

[4] Carey, M.J., and DeWitt, D.J. Extensible Database Systems. In *Proceedings of the Islamorada Workshop,* February 1985.

[5] Chou, H.T., DeWitt, D.J., Katz, R.H., and Klug, A.C. Design and Implementation of the Wisconsin Storage System, *Software - Practice and Experience,* Vol. 15(10), 943-962, IEEE, October 1985.

[6] Date, C.J. *An Introduction to Database Systems.* Addison Wesley, Reading, Mass., 1986.

[7] Dayal, U., and Smith, J.M. PROBE: A Knowledge-oriented Database Management System. In *Proceedings of the Islamorada Workshop,* February 1985.

[8] Hodges, D.A., and Rowe, L.A. Information management for CIM. In *Proceedings of Advanced Research in VLSI.* Palo Alto, CA, March 1987.

[9] *INGRES Reference Manual,* Version 3.0, VAX/VMS, Relational Technology, Inc., Berkeley, CA, May 1984.

[10] Liskov, B.H. and Guttag, J.V. *Abstraction and Specification in Program Development,* The MIT Press, Cambridge, MA, 1986.

[11] Nikhil, R.S. Functional Databases, Functional Languages. In *Proceedings 1985 Persistence and Data Types Workshop, Appin, Scotland,* August 1985.

[12] Nikhil,R.S. *An Incremental, Strongly-Typed Database Query Language.* PhD thesis, Moore School, University of Pennsylvania, Philadelphia, PA, August 1984.

[13] *PRELUDE Reference Manual,* Release 2.0, VenturCom, Inc., Cambridge, MA 1986.

[14] Shipman, D.W. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems* 6(1):140-173, March 1981.

[15] Smith, J.M., et al., MULTIBASE–Integrating Heterogeneous Distributed Database Systems. In *Proceedings National Computer Conference,* Chicago, May 1981.

[16] Stonebraker, M. and Rowe, L.A. The Design of POSTGRES. In *Proceedings 1986 SIGMOD Conference,* Washington, DC, pp. 340-355, May 1986.

[17] Stonebraker, M., Wong, G., Kreps, P., and Held, G. The Design and Implementation of INGRES, *ACM Transactions on Database Systems* 1(3):189-222 1976.

[18] Turner, D.A. The Semantic Elegance of Applicative Languages. In *Proceedings ACM Conference on Functional Programming Languages and Computer Architecture, Portsmouth, NH,* pp. 85-92, ACM, October 1981.