SPECIFYING INTEGRATED CIRCUIT PHOTOLITHOGRAPHY PROCESSES USING HEURISTIC AND ALGORITHMIC TECHNIQUES

by

Michael Frank Klein

Memorandum No. UCB/ERL M85/73

10 September 1985

SPECIFYING INTEGRATED CIRCUIT PHOTOLITHOGRAPHY PROCESSES USING HEURISTIC AND ALGORITHMIC TECHNIQUES

by

Michael Frank Klein

Memorandum No. UCB/ERL M85/73

10 September 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Specifying Integrated Circuit Photolithography Processes Using Heuristic and Algorithmic Techniques

Michael Frank Klein

Specifying Integrated Circuit Photolithography Processes Using Heuristic and Algorithmic Techniques

Michael Frank Klein

ABSTRACT

In synthesizing the fabrication process for an integrated circuit (IC), process engineers spend a great deal of time manually "integrating" existing CAD tools, algorithms, technical literature, and advice from local experts. Because of its difficulty and scope, this "integration" is rarely performed completely, and is usually only performed for a few alternative IC processes.

An expert computer-aided design system called Cameo has been developed that interactively leads the user through the synthesis steps for IC photolithography. At each step, Cameo offers the user a unique "mini-expert" from its knowledge base which reflects the method an expert would use to synthesize the process step. Thus the different kinds of knowledge used to synthesize an IC fabrication process may be integrated into a single system. The user may also provide his own solution, and may ask for further information and references on the step. Cameo is exceptionally easy to use, and allows the user to synthesize and evaluate photolithography alternatives with the SAMPLE process simulator in a matter of minutes.

Cameo is a specific application of a general framework which partitions a knowledge base consisting of heuristics, algorithms, graphs, and/or formulas

into individual mini-experts, each of which can be assigned to a single design decision. The framework allows new mini-experts to be added or the system's decision structure to be modified quickly and easily. Its knowledge base is highly modular, leading to easy modification and maintenance. Its data base structure reflects the tentative, hierarchical nature of engineering design and allows the user to incorporate previously-developed photolithography steps from libraries. Cameo is implemented in Hewlett-Packard's HPRL I and the University of Utah's Portable Standard Lisp, and presently runs on an H-P 9836 desktop computer.

David A. Hodges

Chairman

Contents

List of Figures

List of Tables

List of Code Lists

Acknowledgments

The first votes of thanks must go to the IC processing experts at UC Berkeley's Electronics Research Laboratory who volunteered much time and effort during all phases of this research. They are Gino Addiego, Prof. Ping K. Ko, Dr. Ping Wai Li, Prof. Andrew Neureuther, Prof. William G. Oldham, and Dr. Yosi Shacham. It is their collective expertise that forms the backbone of this research.

I also thank the people in Hewlett-Packard's Expert Systems Project, led by Dr. Steven Rosenberg, for helping me learn the basics of HPRL and the development system running on the H-P 9836 computer, and for arranging for the donation of the system I used. I would also like to thank Drs. Jeff Y.-C. Pan, Harry G. Barrow, and Jay M. Tenenbaum, and the others at Schlumberger's Laboratory for Artificial Intelligence Research for the stimulating exchanges of ideas we had in the early phases of this research.

Special thanks go to Professor David A. Hodges, my research advisor, for his support and advice, and for the just-enough guidance he gave to this

research's direction. Professors Randy Katz and Frederick E. Balderston, the other members of my thesis committee, made many substantive and stylistic suggestions on drafts of this dissertation that greatly helped its focus.

Cliff Lob, whose RUBICC circuit critic was an inspiration that AI-based CAD systems can be successful, deserves thanks for his help in getting me off the ground. It is impossible to thank everyone else who helped during my years at UC Berkeley individually, but I must make special mention of Ron Gyurcsik, Dr. Mark Hoffman, Karti Mayaram, Peter Moore, Tom Quarles, Rick Spickelmier, and Christopher Williams.

And, of course, to June, my wife, whose continuous support throughout this research allowed me to concentrate on doing the best I could.

This research was supported in part by a grant from the Semiconductor Research Corporation, and in part by a grant from Xerox Corporation with a matching grant from the University of California's MICRO program.

CHAPTER 1

Introduction

Many computer aided design (CAD) tools are available to the integrated circuit (IC) fabrication process designer. Other resources, such as local experts and technical literature, are also often easily available. But in most cases, these design resources are isolated from each other and therefore can not be used to their fullest potential.

Some areas of engineering design have been well addressed by CAD research and now have integrated CAD systems available for designers. Others, including IC process design, have not yet had the same research activity aimed at them. The main intent of the research described in this dissertation is to tie together these many IC process design resources using a wide selection of computer-based techniques.

1.1 Motivation for This Research

Research in IC processing is a major thrust at the Electronics Research Laboratory (ERL) of the University of California at Berkeley. Over 50 students and 10 professors are engaged in IC process research. A wealth of valuable resources exists at UC Berkeley's ERL for IC process design, such as computer-based tools like circuit, device, and process simulators, technical literature, and local experts in all phases of IC processing.

Because these resources are not tied together, however, they cannot be used to their fullest potential. The resources are difficult to find and may be difficult to apply to particular design problems. These limitations often apply to all users, even experienced IC process researchers. A CAD system that serves to tie together *existing resources* would be an important contribution to the UC Berkeley Electronic Research Laboratory's IC process design community.

When this research project began in 1983, its goal was to develop a CAD system for IC process design, with emphasis on investigating the usefulness of artificial intelligence (AI) techniques for this specific problem. AI techniques, particularly those of expert systems, were considered to be useful for this application because of the great deal of heuristic [1] knowledge that appeared to separate the expert designer from the beginner.

[1] *Heuristic* describes solution methods that are learned by observing the solution methods to other problems [Poly57], or those learned by experience. In the context of AI, *heuristic* specifically refers to the "rules of thumb" knowledge that experts in an area build from experience [Haye83].

Developing a complete CAD system for IC process design is far too ambitious a task for one person to undertake. Instead, this research has concentrated on the photolithography steps of the overall IC process. Three reasons make this a good choice for this system. First, the photolithography steps are repeated once for every mask layer in the fabrication process, and thus improving this one step can provide great leverage. The second reason is that this is the area of IC processing in which UC Berkeley's ERL has the most local expertise. Lastly, photolithography is a relatively bounded area that can be considered independently of many other facets of IC processing without oversimplification.

Closely related to this research is a parallel effort to provide a computer aided manufacturing (CAM) system for the Berkeley integrated circuit fabrication laboratory. While well beyond the scope of this research project, the eventual integration of these CAD and CAM systems is a primary design goal for both research projects. The eventual system should provide integrated services spanning design, fabrication, fabrication equipment control and monitoring, yield and other engineering data analysis, cost analysis, and inventory.

1.2 Research Contributions

The contributions of the research described in this dissertation are three-fold. First, an investigation was undertaken to learn what kind of CAD system would be most useful for the particular environment at UC Berkeley's IC

laboratory. The initial work was along the lines of a typical expert system where the system consists of a large knowledge base and an inference engine (these terms will be explained in Section 3.2.1 on Page 25). The system would ask the user some questions and would eventually generate its "best" solution. As work progressed, goals were revised in response to user reactions, and the system's structure evolved into a highly interactive, step-by-step design advisor under the user's control.

Second, it is shown that the overall photolithography design process can be decomposed into subproblems, each of which can have a "best" solution method assigned to it. The decomposition relies on assembling an IC process from libraries of previously-used process steps, then refining the steps to customize them to the specific requirements the user has entered. This decomposition, while admittedly limiting the set of solutions that may be found with it, helps the user find a starting point for his IC process design, a valuable contribution in itself. In the case of IC process design, the design process itself is not well structured and can benefit from a system that helps designers structure the problem as an expert might.

Third, a system called Cameo has been implemented that reflects these observations and provides a useful service to IC process designers. Cameo is really a specific application built on an underlying framework developed as a result of the initial work described just above. The framework allows a large and complex synthesis problem to be broken into small, manageable, relatively-independent subproblems and assigns a specialized "mini-expert"

to each one. The "mini-expert" may represent the knowledge for solving each subproblem in whatever way is most appropriate. It was found that an equally important contribution was the set of "reference files," short, descriptive files explaining the *how* and *why* of each subproblem. A reference file can be seen by the user at any time with a single keystroke.

1.3 A Running Example

After Cameo's framework was designed and implemented, an application was chosen to evaluate it. More about the choice is discussed in Chapter 10. In short, Cameo was made to be an expert on linewidth control for the metal and polysilicon layers, for a linewidth of 2 μm.

This example will be referred to or expanded upon in many of the remaining chapters of this dissertation to help ground the discussion to a real problem. Chapter 10 is devoted to a detailed discussion of the application.

1.4 Outline of Dissertation

Chapters 2 through 5 are largely descriptive chapters introducing many concepts and current work in the fields of artificial intelligence, computer-aided design, and IC processing. Chapters 6 through 9 describe the specific structure of Cameo and its major contributions.

Readers interested in a short summary of the research work and results should probably read first this chapter and the concluding chapter

(Chapter 11). For an introduction to Cameo's structure with some but not overwhelming detail, Chapter 6 ("Overview of Cameo's Structure") is most useful. The remaining chapters will be of varying interest to readers depending on their backgrounds and interests. A short description of each chapter follows.

Chapter 2 presents a short survey of current approaches to CAD systems, with the aim of finding a direction that a new AI-based CAD system should take. Chapter 3 introduces terminology used throughout the dissertation and gives general background on the application of those terms in this dissertation. Chapter 4 examines an abstract view of the design process and how an ideal (but realistic) CAD system might look. Chapter 5 gives a very short overview of IC processing and describes a typical design process for IC photolithography. Chapter 6 introduces the structure of the system developed for this project, and describes the tools and development system used. Chapters 7 and 8 describe the knowledge base and data base of the system, respectively, in detail. Chapter 9 explains the importance and describes the structure of the user interface. Chapter 10 describes the specific application addressed by this research in detail. Chapter 11 summarizes conclusions about the research done for this project and points to areas for further work.

CHAPTER 2

Survey of Previous Approaches

Most previous contributions to IC processing CAD have been simulation programs. Outside of IC processing, previous CAD systems have ranged from simulation programs to a CAD "toolbox" whose tools are integrated by a standardized data format or a data base and a consistent, often graphic, user interface. These approaches to CAD tools and systems can be called *algorithmic*, since they are based on either numeric or algorithmic computations or algorithms provided for special data structures.

A few CAD systems based on artificial intelligence (AI) techniques are beginning to appear. These may be divided into two categories: those that use solely or predominantly AI techniques, and those that freely mix AI techniques and conventional tools. However, the dividing line between AI and non-AI techniques is often hazy.

2.1 Simulators

Simulation programs (simulators) model physical processes. Using simulators allows many alternative processes to be investigated and data from any point in the physical process to be studied. This is especially valuable when the physical process is difficult to carry out or data is difficult to collect. An example is simulating the stress points of a bridge under many different loading conditions. The physical process itself is extremely costly to carry out, and measuring all the important stress points is essentially impossible.

For IC process design, simulation programs are as valuable as for bridge design. It is impossible to measure all the changes that a semiconductor wafer undergoes during a processing step. It is also expensive and time-consuming to route wafers through a sufficiently complete set of alternative processing steps to characterize an IC process. A simulator can provide access to a "virtual" IC processing line that has ideal measuring instruments on every wafer at every point on the processing line.

It is important to remember that simulation programs have limitations. Assuming accurate physical models, simulators may accurately simulate the world described by the user's input, but they cannot indicate whether the input describes a "reasonable" world, or whether it comes close to satisfying the user's design requirements. Thus simulators are most valuable when used as verification tools for designs arrived at by other means. One problem noted during this research is that many IC process designers tend to use simulation programs too early and too often in the design cycle.

Some popular and important simulation programs for IC processing are described in the following sections. All of these programs have contributed greatly to the state of IC processing and form a rich base of resources to build upon.

2.1.1 SPICE

Originally developed as an assemblage of class projects in a computer simulation class at UC Berkeley, SPICE [Nage75] has evolved to become perhaps the best-known circuit simulation program, currently in use at over 2000 locations. Its main virtues are its popularity and its robustness [Gyur85]. Input to SPICE is a sequence of text lines describing the circuit interconnections, models for devices, and analysis commands. Output is line-oriented ASCII, either as a table or as rough plots of node voltages or currents at each time step.

2.1.2 SAMPLE

SAMPLE [Oldh79,Oldh80] is a two-dimensional IC process simulator also developed at UC Berkeley to study the IC fabrication processes that shape the topography on a wafer [Nand84]. SAMPLE's development philosophy was to take advantage of recent advances in the availability of computing resources to allow the simulator to be "substituted" for the actual physical processes, ideally making operation of the simulator transparent to the user. SAMPLE simulates the lithographic, deposition, and etching processing steps.

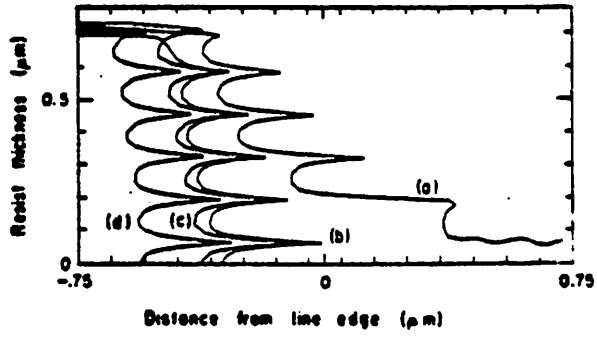

Figure 2.1: SAMPLE output example. This example shows the simulated resist line-edge profiles for two resist thicknesses and development times. This figure is Figure 9 of [Oldh79].

Designed for interactive use, SAMPLE's input format is more flexible than that of SPICE. Output can be in the forms of SPICE or as standard graphics commands suitable for interpretation by a graphics display device such as a Hewlett-Packard 2648A graphics terminal. A typical output graph from SAMPLE (see Figure 2.1) would be a cross section of the photoresist layer on a wafer after exposure and developing for a range of developing times, showing the shape and extent of the developed region for each develop time.

2.1.3 SUPREM-III

SUPREM-III, developed at Stanford University, is also an IC process simulator [Ho83,Anto79]. While it is strictly one-dimensional, being able to report only on thicknesses of layers and the distribution of impurities in those layers

as a function of depth, SUPREM-III can simulate most steps of a complete IC process. Input to SUPREM-III is similar to that of SAMPLE, in that IC process parameter values are entered in a flexible command language. Output is either line-oriented ASCII or graphic.

2.1.4 PISCES II

PISCES II, recently developed at Stanford University, simulates the behavior of individual devices (transistors, for the most part) as opposed to circuits like SPICE does or physical processes like SAMPLE and SUPREM-III do [Pint84]. It does so by simultaneously solving the fundamental equations for charge and electric current for two-dimensional finite elements throughout the semiconducting material described in the input. Each element is assumed to have constant material properties inside it. The elements can be of different shapes and sizes so that smaller elements can represent parts of the material where its properties vary more widely over small regions.

Some initial decisions in PISCES II are based on heuristics (rules of thumb) which help PISCES II converge quickly to an initial starting point for further simulation. This knowledge is built into the code and is not separate as in an expert-system knowledge base.[1]

Input to PISCES II can be in the form of line-oriented ASCII input, or a preprocessor may be used to enter the same information in an interactive

[1] One of the major distinctions between expert systems and conventional programs is the explicit separation of the knowledge needed to solve a problem from the mechanism that solves it. More about this is presented in Chapter 3.

graphic form. Output is in tabular ASCII form, but plotting postprocessors are available for graphic output.

2.1.5 FABRICS II

FABRICS II is different from all other simulators discussed so far, in that it simulates the variances in outcomes of an IC process due to disturbances or random perturbations in the process [Nass83]. Output includes probability distributions of outcomes such as yield as a function of sheet resistance [Stro84]. It uses simplified analytic models of the steps of the IC process derived from solving boundary conditions of the complete analytic models to keep computation time to a minimum.

2.2 Integrated CAD Systems

Many recent CAD systems have taken the approach of providing an easy-to-use "front end" to a CAD toolbox. These systems vary a great deal in how open their architecture is. An open architecture allows others to add capabilities to the system, while a "turnkey" closed architecture is available only as a monolithic system with continued product and customer support a major part of the system's cost.

Commercial integrated CAD systems abound, of which General Electric's Calma system is the most widely used. The Calma system, like other commercially available systems, has a closed architecture. In contrast, most research CAD systems have open architectures. This difference is mainly

due to strategic reasons. The vendor of a monolithic turnkey system knows that customers will incur high "switching costs" to switch from one system to another and are therefore reluctant to do so. Most customers of the commercial systems would prefer a turnkey monolithic system to avoid potential support problems, while the research community usually prefers the flexibility an open architecture provides and is not as concerned about compatibility and support, and is concerned about being locked in to any particular commercial product.

As part of a major research program at the Massachusetts Institute of Technology in computer aided manufacturing, a program called MASTIF [Boni85] has been developed. MASTIF uses graphics and a window-oriented user interface to tie together existing simulation, synthesis, and analysis tools in a uniform fashion. Quoting from the abstract to [Boni85],

> MASTIF provides a common, interactive user interface to different process simulators (both one- and two-dimensional)... MASTIF provides process design tools including incremental Process Description development and analysis, as well as version management of Process Descriptions and physical cross-sections.

MASTIF helps solve one of the major problems in IC process CAD, the inaccessibility and incompatibility of many of the available IC processing CAD tools.

A widely used open-architecture IC layout system called Kic [Kell82a], developed at UC Berkeley, is a good example of an integrated CAD system. It provides a central CAD data base (Squid [Kell82b]) and a graphics-oriented user interface. Special programs that perform specific tasks like simulation,

layout rule checking, layout compaction, PLA generation, and so on, are easily interfaced with the data base and user interface. Another important toolbox-based CAD system, also developed at UC Berkeley, is the Magic layout editor [Oust84].

An optimization program that serves as a front end for other specialized programs that is in use at UC Berkeley is DELIGHT [Nye83]. DELIGHT consists of optimization algorithms and an interface language that links other programs such as simulators with the optimization algorithms in well-defined ways. DELIGHT is used less interactively than the previous examples.

2.3 AI-Based CAD

Whereas an integrated toolbox CAD system provides the tools needed to perform a design task, most recent AI-based CAD systems help the user find a design strategy. The design strategy can then be carried out using a CAD toolbox system. Some AI-based CAD systems perform only strategic decisions, while others (detailed in Section 2.4) mix AI techniques with conventional CAD tools.

The VLSI Design Automation Assistant (DAA) being developed at Carnegie-Mellon University uses heuristic knowledge to generate a technology-independent layout description of a VLSI system given an algorithmic description [Kowa83]. DAA first decides on the system's "global allocations," such as the number of global registers and ports, and establishes constraints and timing information if not all of these are supplied. It

then iteratively makes decisions about dividing the system into modules and routing the data path. When this is done, global improvements are made such as removing or combining redundant registers and logic. DAA is implemented almost exclusively in OPS5 [Forg81]. Its designers recognized the need for algorithmic routines such as counting, cost estimation for evaluating choices, and checking set membership of elements, and rewrote rules for these operations as procedures in the C language [Kowa85].

A research group at the Massachusetts Institute of Technology has been applying AI techniques to VLSI circuit and systems design and layout since 1979 [Suss79]. This group has emphasized the development of integrated systems, decomposing complex design problems into hierarchical or other structured abstractions in order to simplify the problems. The current direction of their work is to develop "junior assistant" systems where the system works interactively with the user, using its knowledge about the design problem to be able to offer timely and relevant help to the user. This approach is modeled after work done to develop a "programmer's apprentice," also at MIT [Rich79].

An interesting application of expert systems to circuit design is RUBICC [Lob84]. RUBICC uses knowledge about NMOS digital circuit design to act as a circuit critic, pointing out possible errors in a design such as floating nodes, too many threshold drops, and other typical design problems. Work on a more general, technology-independent version of RUBICC is described in [Spic85].

Another AI approach to CAD focuses on developing "consultation systems" for a specific problem, where the user answers a series of questions and the system generates one or more plausible solutions. The usage of this type of system is less interactive than with the junior assistant system. The consultation system relies on an extensive knowledge base about its application field, so may be difficult to maintain in a fast-changing field.

One system using AI techniques for IC processing was developed at Hewlett Packard Laboratories for diagnosing problems with photolithography while the wafers are undergoing processing [Clin85]. This is a diagnosis system as opposed to a design or synthesis system.

2.4 Mixed Approaches

A mixed approach to a CAD system is one that uses a combination of algorithmic and AI-based approaches smoothly integrated into a single system. The appeal of a mixed approach is that the "best" solution method—AI, algorithmic, or a combination—can be used for each problem in the decomposition of the overall design problem. This is an area of active current research.

A major effort in this area is ARSENIC, the work of Prof. Daniel Gajski at the University of Illinois at Urbana-Champaign [Gajs84]. ARSENIC is a "quasi-expert system" for IC system and layout design. Gajski has decomposed the IC design problem into a three-level hierarchical design process.

ARSENIC consists of a library of specialized silicon compilers that are assigned to subproblems of this decomposed IC design process. A combination of heuristics and algorithms determines the assignment of these silicon compilers to the subproblems. The system is highly interactive. As will be seen, there are many similarities between ARSENIC's and Cameo's design philosophies.

A major research program in VLSI chip design is proceeding at Schlumberger's Laboratory for Artificial Intelligence Research [Kram85]. This research combines ideas from a number of areas being researched at both Schlumberger and Stanford University, including developing representations for designs, using these design representations to reason about the design, and incorporating the reasoning mechanisms with other synthesis and analysis tools. Current and previous work relevant to this research includes the Palladio system, a circuit design environment [Brow83]; MARS, a hierarchical rule-based circuit simulator [Sing83]; Corona, a design description language [Sing84]; and DIRT, a program that derives heuristic rules describing a device's behavior from a functional description of the device [Kram84].

Another research program at CMU is developing an expert-system-based VLSI design environment [Bush85]. ULYSSES (Unified LaYout Specification and Simulation Environment for Silicon) accepts IC descriptions as logic equations and analog circuit descriptions. With assistance from the designer, it will produce a geometric layout for the design, then verify that the layout meets desired electrical specifications. The emphasis is on controlling existing CAD tools rather than writing new CAD tools. Current

and previous research work relevant to this project includes the Palladio system described above [Brow83]; a hardware description language called DIF [Bush83b]; Delilah, a graphic user interface [Bush83a]; OPS5, the knowledge representation language used [Forg81]; the Designer's Workbench, an early design environment developed at Bell Laboratories [ONei79]; Demeter, a design methodology and environment for the highest levels of computer and computer network design [Siew83]; and SRL (Schema Representation Language), used in ULYSSES for archival storage of designs [Wrig83].

The SRC-CMU Center for Computer-Aided Design at Carnegie-Mellon University has been investigating mixed and otherwise integrated approaches to developing a large CAD and Computer Aided Manufacturing (CAM) system for VLSI [Dire81].

Much work in a large integrated CAD system defies easy categorization into AI-based vs. algorithmic approaches, however. A chip routing or placement program certainly incorporates some heuristic knowledge in addition to algorithmic routines. Many layout design rule checkers, to ensure flexibility in the face of constantly changing technology, express knowledge about allowed and disallowed layout combinations as "layout rules" separate from the control and algorithms of the rest of the program.

2.5 Other Approaches

Other approaches that do not fall cleanly into AI-based approaches, algorithmic approaches, or a combination of the two are currently active research

areas. The work at MIT referred to earlier (Section 2.3 on Page 15) mixes a number of approaches, including those detailed below.

2.5.1 Symbolic Algebra

Most programs can perform some aspects of algebra by repetitively plugging values into a given equation to find answers. Others can solve complex systems of linear equations once the equations' coefficients have been entered. Symbolic algebra programs like MACSYMA [Math77], however, work with the actual symbolic representations of the equations. These programs can perform operations on the equations by factoring, algebraic simplification, definite and indefinite integration, etc. Thus a system incorporating symbolic algebra might be able to mimic an expert's approach to a problem by performing the same algebraic operations on equations that the expert might.

2.5.2 Deep Reasoning

Deep reasoning systems use special models to simulate the systems they represent [Brow77,Pan83]. Often these models are qualitative in nature, specifying the behavior of the system in inexact terms.[2]

While this is a similar approach to the algorithmic simulators discussed earlier, deep reasoning systems use some AI techniques in their operation.

[2] An example is describing a resistor by saying that the voltage across it *increases* as the current through it *increases*.

primarily the separation of the application-specific knowledge from the control portion of the program, and the qualitative, often heuristic, models used.

A very good example of a deep reasoning system is described in [Pan83]. This system uses its deep knowledge about the specific application to diagnose multiple dependent failures,[3] something most diagnosis systems cannot do.

2.5.3 Constraint Propagation

Any two or more variables that are not completely independent of each other are somehow constrained in the sets of values they take [deKl80]. Whenever any variable in an equation takes a value, this immediately constrains each of the remaining variables to a smaller set of values. For most useful constraints of N variables, if $N - 1$ variables have taken values, the value of the Nth variable can be immediately determined.

Typically one variable will take part in more than one constraint, analogous to most analytic problems where any one variable appears in more than one equation. Thus a single variable changing can cause a rippling effect throughout all variables related to it. When one variable's value is assigned or changed, the constraint propagation system checks to see if any other variables or parameters can be calculated or updated. Constraint propagation can be used to maintain correct system state under changing conditions.

[3] Multiple dependent failures occur when a primary failure, such as a transistor shorting, causes secondary failures, such as a resistor burning and a fuse blowing, to occur. A simple-minded diagnosis system might simply suggest replacing the fuse or the resistor, or may not even be able to diagnose anything since this particular combination of failures does not match anything it is programmed for.

2.6 Other Important Work

One important factor in Cameo's development was work done by Clancey [Clan84] on Classification Expert Systems. These systems do not attempt to piece together a complete solution to a problem from scratch but instead have coded into them a number of potential solutions expressed in highly general ways; their job is to choose from among these solutions by classifying the problem to fit the known solutions, and to refine the chosen solution(s) until the desired level of specificity is reached.

Clancey's work led to the idea of using step libraries (as explained in Section 6.8.4 on Page 76) of "canned" photolithography steps. After choosing one or more steps from libraries, the system would concentrate on refining the choices.

2.7 Chapter Summary

Simulation programs are very valuable tools but their limitations must be kept in mind. They are best used as verification tools.

Open-architecture integrated CAD systems appear to have great promise in improving designer productivity. The main reasons are that they provide a standardized data structure and consistent user interface to a host of special tools, and their open architecture allows new users to add special-purpose tools of their own. This latter capability is essential for most CAD systems since the technology they address is constantly changing.

Recent integrated CAD system efforts emphasize integrating existing tools rather than developing yet more new tools. Integrated CAD systems built with a "toolbox" approach assume that the design process itself has been decomposed into phases that can each be addressed by a special-purpose tool.

Most CAD systems surveyed cannot answer a user's question of why a certain step should be performed a certain way. Most assume a quite competent level of user expertise in the design field. Often the novice designer must find his own way through the trees to see the forest.

AI techniques hold great promise in addressing certain problems where heuristic knowledge, rather than numeric or algorithmic routines, is the predominant method of solving problems. A few systems have recently appeared that mix AI techniques and conventional CAD tools. Like recent integrated non-AI CAD systems, these emphasize the integration of existing CAD tools. The AI techniques are usually used to help the user make initial design decisions and choose appropriate tools for further designing.

A successful design philosophy for this CAD system may be one that combines all these observations. It should offer a standardized data structure and a user interface that tie various special-purpose tools together. Its architecture should be open, allowing users and future system developers to add tools easily. It should be able to mix conventional and AI-based tools, depending on the specific problem the tool addresses. The user should be able to ask "Why?" of the system.

CHAPTER 3

Terminology and General Background

Since this research combines technical concepts and terminology from the two diverse areas of IC processing and computer science, and readers will probably be more familiar with one area than the other, some explanation of terminology is in order. In the following two sections, some of the most common terms used in this dissertation are described. Other terms used less commonly will be described as they are introduced in context.

The first section of this chapter introduces terms used in IC processing, while the second section concentrates on software terminology. Both sections introduce terms that are used extensively throughout the remainder of this dissertation. While intimate familiarity is certainly not required of either area, some understanding will help greatly in following the discussions in the rest of the dissertation. Thus some terms are defined rather narrowly, reflecting only the context in which they will be used in this dissertation.

3.1 IC Processing Terminology

An *integrated circuit (IC) process* is the set of fabrication steps that a semiconductor wafer goes through to become a finished product. A complete IC process normally consists of about 8 to 16 major *process steps*, each of which consists of a photolithography step and a processing step.

Photolithography is the step that transfers an image from a mask to the wafer. The photolithography step defines regions on the wafer corresponding to patterns on the mask, and is usually followed by the actual *processing step* that affects those regions.

IC process design is the design activity that results in a set of process steps, often called the *process flow*. In most cases, IC process design is separate from circuit or system design and physical layout design of the IC. It is strictly involved with the fabrication process, and assumes that the person performing the processing already has a set of masks generated by other CAD resources like layout tools.

However, all three of these IC design phases are interrelated, although the extent to which they interrelate varies: One extreme is the case where an IC design pushes the state of the art in all three areas. In this case, design of the circuitry, layout, and processing steps cannot be considered separately at all and iterative design of all three phases must be done. This is also the matter of course when the final product is a high volume item and sales volume will justify the cost of iterative development. Primary examples are dynamic memories and analog circuits.

The other extreme is beginning to appear, where a "silicon foundry" provides a standardized IC process. It publishes the specifications for that process, and anyone using it must adhere to its requirements. This affects both the circuit or system design and the physical layout of the chip, as compromises may have to be made in both these areas to accommodate the given IC process.

For the purposes of this dissertation, the result of the process design activity is a *process plan*, since the result looks very much like a step-by-step description of actions to take. Usually the goal of IC process design is a complete IC process, but sometimes it is just one or a few specific process steps that might form a subset of the overall IC process.

3.2 Software Terminology

3.2.1 Artificial Intelligence

Cameo is an *expert system*, a system that has some amount of knowledge built into it that enables it to perform certain limited tasks as well as or better than a human expert in the same area. Expert systems combine many results of artificial intelligence research into areas such as knowledge representation and the human decision method. Expert systems are specifically intended to emulate human performance in a very narrow area.

A number of properties are generally accepted as differentiating an expert system from conventional software systems, among them:

- Expert systems separate the method of solving problems from the information used by the solution methods. Most computer programs are written with knowledge and method intertwined in the same code. An expert system separates the two into a *knowledge base* and an *inference engine*. The inference engine is intended to be general purpose and ideally makes no assumptions about the content of the knowledge base, while the knowledge base customizes the expert system to its specific application. The price to pay for this flexibility is execution speed.

- Expert systems can deal with complex application fields by using heuristic knowledge to narrow the set of solutions to a problem to a much smaller set that deserve further investigation.

- Most of the knowledge entered into an expert system is *surface knowledge*, or superficial knowledge. This knowledge is often in the form of indirect cause-and-effect relationships gained by experience, and is not based on rigorous application of theory. Most conventional programs use *deep knowledge*, like circuit simulators that use precise models of devices to simulate their behavior.

- Most expert systems are *goal-directed*, in that the user specifies a goal for the system to solve, and the system uses its store of knowledge in whatever order is needed to attempt to "prove" that goal. In contrast, most conventional programs work through a sequence of steps determined by the input data to arrive at an answer. In other words, most

conventional programs are deterministic, whereas expert systems are not.

- Most expert systems rely heavily on pattern matching instead of numeric computation to identify the bits of knowledge in their knowledge base that might aid in proving a goal.

Most expert systems are goal-directed, using a process called *backward chaining*. A goal is given to the system to attempt to prove. All parts of Cameo's knowledge base that employ AI techniques are based on the goal-directed model of reasoning.

In backward chaining, the expert system follows the following steps until it has succeeded or failed to prove the goal:

1. If the goal matches an item that is already in its data base, then the goal is already proven and backward chaining is finished.

2. If the goal does not yet exist in the expert system's data base, its knowledge base is consulted to find any bits of knowledge that might prove the goal. These bits of knowledge are most often expressed as *production rules*, or "*IF... THEN...*" rules. If the "*THEN*" part of the rule matches the goal, then the "*IF*" part of the rule is adopted as another goal and the system runs through these same steps with it.

A short example will illustrate. Suppose it is known that Bill is going to London and he is trying to decide whether to take a raincoat. Bill would

ask his expert system whether a raincoat is necessary in London. Further suppose that the expert system has the following rules already entered into it:

IF the weather is raining,
THEN you will need a raincoat.

IF you are in London,
AND it is winter,
THEN the weather is raining.

The expert system finds that the successful conclusion of the first production rule would prove the goal, so it uses it. The premise of this rule (IF the weather is raining) is then checked against the data base to see if this fact already exists. It does not, so the expert system adopts it as a goal and attempts to prove it.

The second rule's conclusion could prove the new goal, so it is now selected. Bill has already told the system that he will be in London, so the first part of the premise exists in the data base. The data for the second part is not anywhere to be found (assuming the system is not tied into a weather information service), so the system asks Bill, "Is it winter?" If Bill replies "Yes," then the second rule's premise is satisfied, so it concludes successfully. This now satisfies the premise of the first rule, so it concludes successfully in turn, informing Bill that he will need a raincoat.

Some expert systems are *data-directed* and use what is known as *forward chaining.* Whenever a new fact is entered into a data-directed expert system, all rules whose premises contain that data are checked to see if their entire premises are satisfied. If so, the rules conclude successfully and the data specified in their conclusions are added to the system's data base. This may in turn cause other rules to become satisfied, and so on.

There are many ways to implement expert systems. One way is by using *frames* as data structures to hold data in the data base [Robe77a,Robe77b]. A frame data base is the major part of Cameo's internal structure.

A frame is much like a record in Pascal or a structure in C with two important exceptions:

- A frame's members (*slots*) can appear and disappear dynamically.
- Frame systems use the properties of *hierarchical inheritance.*

A data base of frames most closely resembles a *hierarchical data base* [Date81], but frame systems allow referring to frame members or to other frames by their names instead of by pointers or addresses. This allows the dynamic nature of such a system to be more easily implemented since address and size information is not kept with the link itself.

Whereas Pascal's records and C's structures are fixed data structures with explicitly defined members, a frame can both vary in size after its initial definition and can appear to have slots that were defined in another frame. The inheritance properties of frames are well suited to expressing data in

a manner more natural to people. When a frame is defined as a *child* of another frame, the new frame "inherits" all the structure and data of the other frame, and may also add members and data of its own. People like to classify things and then make statements that are applicable to all members of a classification. We know, for instance, that all mammals have hair, so we define a classification called "mammal" all of whose "children" have hair. We store the information about hair with the definition of the mammal class, not with the definition of every single mammal.

Frames with inheritance allow expressing data in the same fashion. One might define a frame called "mammal" (which in turn might be a child of the "animal" frame) where one of the slots is called "hair" and the data stored in that slot is "yes." Any frame which inherits from the "mammal" frame will automatically appear to have its own slot called "hair" with the data "yes." The elephant frame might also define additional slots like "trunk."

One of the most important results of expressing data with inheritance frames is that rules can be written about whole groups of things at once. It might have been better to express the fact about hair as a rule that said

IF the thing is a mammal,
THEN the thing has hair.

Either forward- or backward-chaining would be able to apply this rule to any child of the "mammal" frame and immediately conclude that the mammal in question has hair.

3.2.2 Object-Oriented Programming

Another programming technique used heavily in Cameo is object-oriented programming. This type of programming involves defining *objects*, which are independent packages of data and code. The major difference between object-oriented programming and conventional programming is that in object-oriented programming, each object type may define its own set of operations that it can perform. The real advantage occurs when many different types of objects define operations that are called by the same name.

Again, an example will illustrate. In most programming languages, a system call is provided that can print a few types of data, such as numbers and strings. One cannot usually print the contents of a graphics screen by using the same system call. In an object-oriented programming system, each data type would define its own "print" *method*. If the *message* "print" is sent to that object, it would run its "print" *method*, as defined for that object.

Now imagine that all objects in the system that might ever be printed have all had "print" methods defined. Whenever the programmer would like an object printed, he simply sends it the "print" message. Two very important advantages result from using object-oriented programming:

- A high degree of consistency.
- Since each object has its own implementation of its methods, a high degree of modularity is preserved. One object's method might be altered but its effect is strictly local.

The major disadvantage of object-oriented programming is a decrease in potential performance as measured by speed of execution of the finished system. Unless the system has been intelligently compiled, locating the actual code for each method requires two lookups instead of one: first to find the object's type, then to find the code in the object that executes the method.

Most object-oriented programming systems also implement a form of inheritance, like frames. The idea is almost identical, in that an object defined as a child of another one inherits all its data (local variables) and also its methods (code). The child object may also add data and methods of its own, or may override those that it inherited. For instance, an object called "number" might be defined that defines local variables such as "value," and a method for adding two numbers. Most children of the "number" object, like integers and reals, will probably not change anything inherited from the "number" object, but the object "complex" will add local variables like "real-part" and "imaginary part," and will override the method for addition.

CHAPTER 4

Characteristics of an Ideal CAD System

4.1 Part of an Integrated System

An ideal solution to the problems faced by IC process designers is an easily used, easily accessible CAD system that is part of a comprehensive, integrated design and manufacturing system. It must be equally useful for beginning fabricators as well as experts, must be easy to modify incrementally, and must interface with arbitrary other resources. The ideal CAD system provides a complete *design environment.*

The key to a fully integrated design and manufacturing system is the organization of data [Beeb83]. The different *applications* or *services* all work from a central data base, converting the data in it to forms which they can use. Communication between applications is largely through the data base. An overview of data base requirements for engineering design is contained in [East81].

The "central data base" may be implemented exactly as such, on a large mainframe computer with high-speed networks linking it to the computers running applications, or the entire system may be implemented on a large, high-performance computer. An alternative, currently being considered for UC Berkeley's integrated CAM system, is to distribute the data base among many smaller computers running the applications, linked by a high-speed local area network, and provide each application with data interface routines that convert requested data from one application's format to the requesting application's format. Another computer on the network would perform conventional data base functions in addition. Figure 4.1 illustrates this approach to a CAD/CAM system.

A distinct advantage of this approach is its inherent modularity. Each individual application contains its own data base in whatever form is optimal for it; all that is required for integration with the rest of the system is the ability to convert its data from its own format to the format required by the requesting application. The potential drawbacks are those afflicting all distributed data-base systems, such as problems with concurrent access of the data base, stale data, duplicated data, etc. The planned implementation uses concepts from object-oriented operating systems. It is hoped that many of these drawbacks can be minimized or eliminated, since in a pure object-oriented operating system, data can only be received by requesting it from another object in the system; all objects have complete control over their own data.[1]

[1] This is a new research area and no published information is available. Faculty members in charge are Professors D.A. Hodges and L. Rowe.

An example of a large, successful CAD and CAM system is one in place at the Boeing Commercial Airplane Company and is described in [Beeb83]. The title of the paper describing this system, "The Heart of Integration: A Sound Data Base," leaves no doubt about the most important component of such a system.

One application in this ideal integrated system would be the design function—the system described in this dissertation. Another would be a verification function using a modeler or simulator. Others might include a critique function analogous to the critiquing performed by RUBICC [Lob84,Spic85] on digital circuits, an execution function that controls fabrication equipment, a statistical analysis function using a program such as FABRICS [Nass83], a scheduling function to help the user schedule critical equipment use, and inventory and other bookkeeping functions.

The user's interaction with the design application of the system must be intuitively straightforward and not impede the user's natural design process. It should be modeled after the structure that an expert designer assigns to the design problem, supporting the same abstractions, decompositions, and fundamental design activities the expert designer performs. Hopefully good hardware and software performance allows the user interaction to be modeled after what is best for the user's efficiency, not the machine's.

An ideal CAD system allows the user to explore potential solutions to his design problem comfortably. The following principle was referred to when making decisions that affect the user interface of the system:

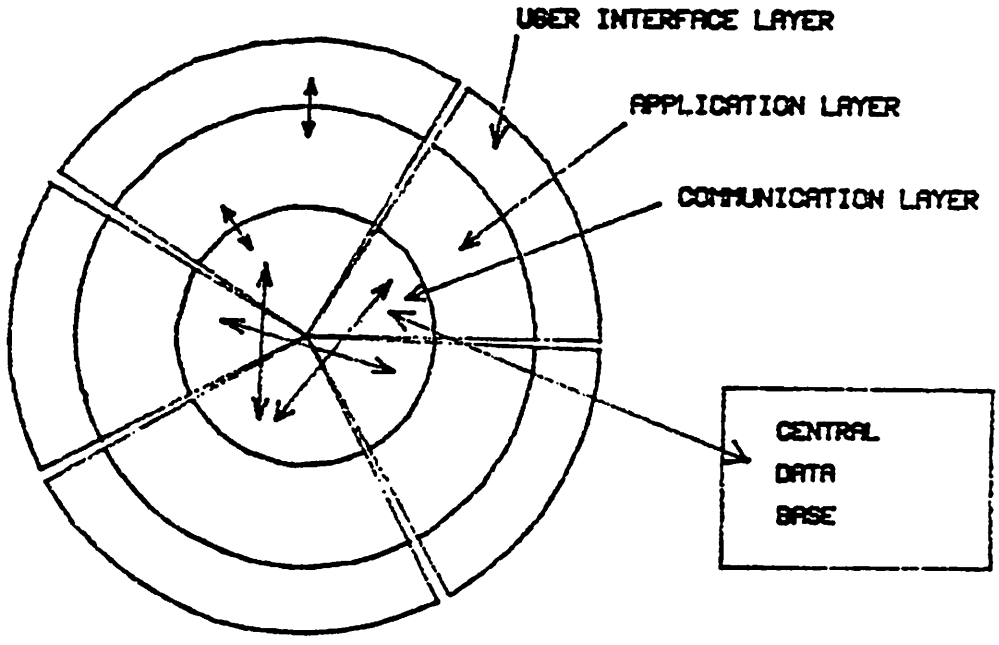

Figure 4.1: Data base design for ideal CAD/CAM system based on a distributed approach. Communication is between applications by converting data formats.

- No matter what sort of data structuring is used internally by the system, it must present the data to the user in a manner intuitively understood by the user and provide operations that manipulate this presentation of the data consistent with the expectation of the user.

Much recent research has been directed at producing more concrete guidelines for user interfaces for interactive computer systems. An excellent

overview of this subject, with many useful references, is contained in [Shne80]. A recent paper that should prove to be of great use in identifying the theoretical underpinnings of user interfaces for interactive technical applications is [Fole84]. Although this paper deals specifically with identifying fundamental properties of interactive graphics interfaces, the methods used in the research described by the authors are applicable to many other interactive computer applications.

This aspect of the ideal CAD system assumes very low cost of computing resources and high information bandwidth to and from the system's display. While this is a valid assumption for this research, it may not be so universally. Appropriate tradeoffs must be recognized in the design of the user interface. For instance, users for whom computing resources are very expensive will probably prefer to forego this principle in exchange for investing considerably more of their own effort in using the system.

4.2 Specific Characteristics for IC CAD

Two kinds of wishes about the capabilities of the system were expressed by people consulted during this research. One I will call the "professor's view," which was that beginning students tended to spend too much time doing detailed investigation of IC process designs that were not always close to a good solution, and not enough time on exploring general directions of process designs. This view held that a CAD system that would help students make the beginning planning decisions of process design would be highly desirable.

The other view, the "student's view," held that the most valuable contribution of an IC process CAD system would be to help the IC fabricator while he was in the midst of a fabrication process (at UC Berkeley, students perform most of the steps of the IC fabrication process themselves). The students envisioned a system that would allow them to enter the steps they had already performed and ask, "What should I do next?" This question often comes up when something in their process has not worked quite the way it should have, and they would like to find out if it is possible to compensate now or in a future step.

In an industrial environment, these two conflicting requirements are equally evident but take different forms. A typical company will only have a few different IC processes available for fabricating ICs due to the enormous difficulty of fine-tuning a process for acceptable yield and performance. Because of the difficulty of designing new processes, most industrial processes are essentially perturbations or relatively slight modifications on earlier successful processes. If an advanced CAD system along the lines of the professor's view were available, new processes optimized for a certain set of requirements could be quickly investigated for feasibility. These processes could be entirely unlike previous ones.

On the other hand, there are many times when a very useful capability would be investigating the slight perturbations on an existing process. This is the kind of work most often performed by industrial process engineers. A CAD system modeled after the student's view would be most helpful here.

These are two very different requirements for a CAD system, and not many (if any) current systems address them equally well. Many systems assume the user to be an expert in the field, and provide specialized tools relieving the user of some design details. Primary examples of systems like this are most VLSI layout editors, which present the design on a graphics screen as it would look physically, and provide cell abstraction features, automatic replication of cells, data base functions, and sometimes layout rule checking. The user must still begin with an architecture and chip topology or floor plan before a VLSI layout editor can be used.

Another type of system is the expert system. An expert system is supposed to mimic a human expert's performance in a very narrow field, or domain. The design philosophy behind some expert systems tends to be opposite that of conventional CAD systems, in that the expert systems assume the user to be a novice, and are designed to make high-level or initial decisions. Often these systems first ask the user some questions and then give him its solutions. Other expert systems perform tasks that are helpful to more expert users. This type of expert system keeps track of design context and is able to perform appropriate actions when requested to do so by the user. Most expert systems are limited to using heuristic knowledge, and are not easily able to integrate conventional CAD capabilities.

Each of these system types is geared toward a certain user sophistication, but neither one alone can serve all kinds of users. An ideal CAD system would be able to provide services both at the initial, high-level planning stage and

at later stages where most of the work is algorithmically oriented. Expert users must be able to bypass the initial stages since these users will often have their own process design already developed. Novice users must be able to find the system useful for initial planning.

4.3 The System Must be Flexible

Building a CAD system for a fast-changing field like IC process design presents some sharp contrasts. The ideal CAD system must be useful for both novices and experts. These groups of users have very different CAD needs. Another sharp contrast is that the overall design procedure for IC processes changes slowly over time, but the specific steps of a design, like which exposure machine to use, change quickly in response to constant technology changes. The design of the system must allow for these contrasts.

The CAD system should be structured around the way that an expert approaches the design problem, but allow its individual elements to be easily changed. In other words, the system's framework can be designed assuming a certain method of use, but must allow almost arbitrary modules to be connected into the framework.

A good analogy would be a technical book. While books are all largely made in the same way, their contents (which are easily changed by the printer) define their subject. An expert and a novice can use the same book; the novice probably reads it by leafing through the table of contents and reading introductory and concluding sections, while the expert probably leafs through the index to find the specific section he needs.

4.4 Interface to Other Resources

The ideal CAD system interfaces to the many other elements of the integrated design and manufacturing environment. This might include entering data from measurement equipment to analyze intermediate results of the process while it is in progress. In keeping with the model of the system shown in Section 4.1 with applications clustered around a central data base, integration of this sort is accomplished through the data base interface routines of each separate application.

A researcher may be considering the next step in his process after the application of photoresist. To derive specific values for the next steps (exposure intensity and time, developing time), he will need to know the precise thickness of the photoresist coating. He may also need to know the variation of the photoresist thickness over the wafer. An ideal CAD system would be able to interface directly with the film thickness measurement instrument, and use the results of the measurement in future steps.

An ideal integrated system, once a process step is designed, can also interface directly to control the fabrication equipment for that step. The results of the fabrication step are recorded and noted, and can be saved, to be used to help design future processes. This particular goal is the most far reaching, for two reasons:

- Few manufacturers of IC fabrication equipment provide consistent computer interfaces to their equipment.[2]

[2] A communications standard, SECS II, has been developed specifically for IC fabrication

- Attainment of this goal assumes that an intelligent system capable of a limited form of learning can be built and its performance relied upon, which has not yet been shown.

A more limited but still useful goal is to provide instructions for the operator of the fabrication equipment. The operator would actually operate the equipment and note the results.

4.5 Chapter Summary

An ideal CAD system, which at this point in the development of CAD technology is probably not possible to build, is a part of an integrated design and manufacturing system. The heart of the integrated system is a universally accessible data base, either physically centralized or distributed among the applications. Each application (the CAD function is but one) ideally interfaces only with this data base. Other applications would include a verification or simulation application, a critiquing application, a control or process execution application, a scheduling application, and inventory and other bookkeeping applications.

---

equipment. At present, however, few IC equipment manufacturers are incorporating the SECS II standard into their equipment. Another equipment interface standard has evolved from automobile manufacturing. The MAP equipment interface jointly developed by General Motors and many of its computing and manufacturing equipment suppliers. This appears to be becoming a *de facto* standard among manufacturers of certain types of equipment due to GM's purchasing power. It remains to be seen whether this standard can be carried over into the IC fabrication equipment arena. See [Leop84,Gene84] for further information.

Two apparently conflicting desires for an ideal IC process CAD system are often expressed. The first, the experts' view, implies that the system be able to make initial decisions about the direction of the resulting design. The other, the novices' view, asks that the system be able to understand a fairly complete process design and advise the user on the current step of the process. The ideal CAD system can address both these requirements equally well.

CHAPTER 5

The Design Process for IC Photolithography

This chapter begins with a short description of the integrated circuit (IC) fabrication process and how IC photolithography fits into it. Then a typical design process for the photolithography steps is described for a particular set of goals, and more details are added to the example design problem introduced in Section 1.3 on Page 5.

5.1 The IC Process

The IC fabrication process consists of from roughly 8 to 16 major separate sequential fabrication steps. Each step consists of transferring an image to the wafer, and performing an actual processing step on the wafer that affects the regions of the wafer that were defined by the image-transfer operation.

The accurate transfer of this image is crucial to the final function of the IC being fabricated.

The steps that transfer the image to the wafer are *lithography*. When these steps are performed using masks, waves of visible light, and photosensitive emulsions, the process is known as *optical lithography*, or *photolithography*. Most current lithography is done optically because it is fast (high wafer throughput), relatively well understood, can be done with commonly available equipment, and is relatively inexpensive. Photolithography is, however, fundamentally limited in its capability to reproduce features on the wafer that are on the order of a wavelength of the exposing light. (Currently the best experimental photolithographic processes have almost reached this limit—about 0.5 to 0.7 μm.) Other lithographic methods developed to advance this resolution capability include "direct writing" on an emulsion on the wafer with a focused electron beam scanned over the wafer, and a method similar to photolithography but using x-ray radiation which has a much shorter wavelength than visible light.

The design of a complete IC process is an immensely complex task that can only be successfully carried out by experts. The difficulty stems from many causes, among them:

- Interdependencies among virtually all steps of the process.
- High degree of divergence between theory and practice—actual fabrication equipment does not behave like the ideal conditions under which theoretical principles are derived.

- Little generally accepted structure to help simplify the conceptual understanding of the entire fabrication process.
- Great economic and competitive pressures to find *optimal* processes instead of more easily designed and reproduced *adequate* ones.
- Literature and other resources that are scattered and largely relevant only to highly specific IC processes.
- Rigidity, inaccessibility, difficulty of use, or irrelevance of computer based design and analysis aids.

In these regards, the state of IC process design is similar to the state of IC *circuit and system* design ten to fifteen years ago. We can observe the rapid progress of IC circuit and system design over the last decade and apply some of those observations almost directly to IC process design. Here are some relevant observations taken from this reflection:

- IC process designers and their managers must realize that creating *optimal* IC process designs is approaching or has already passed the point of reasonable economic return. The complexities involved greatly favor accepting *less than optimal* IC process in return for being able to create those processes more quickly, more reproducibly, more reliably, and with a greater understanding of causes of problems. Recent work in UC Berkeley's Graduate Business School [Byer83] shows the attractiveness of being able to choose from among an envelope of "near-optimal" solutions to a problem, leaving the choice of the most appropriate solution

in light of the tradeoffs involved to the person best able to make these qualitative judgments.

- IC processes should be decomposed into more independent steps with simple and well-understood interfaces between the steps. The price is settling for less-than-optimal IC processes.

- IC processes need to be more portable than currently, so a process can be reliably transferred from one facility to another, whether in the same company, between cooperating companies, or from a research organization to an industrial concern. This favors the decomposition of the IC process and understanding the interfaces between components.

- IC processes need to be amenable to continuous improvement by incrementally upgrading single process steps. This again favors decomposition and understanding the interfaces between steps.

- There will always be niches for the highest-speed IC processes, but the great majority of finished IC products will be those that are fabricated under the above assumptions.

The research undertaken for this project assumes that the future direction of IC process design is the same general direction as IC circuit and system design has taken. This allows the simplification of the overall IC process and enables investigating the photolithographic steps more independently from the rest of the IC process than is usually done in competitive industrial settings today.

5.2 IC Photolithography Design

The initial design decisions made by an experienced IC photolithography designer revolve around the concept of transferring the image from the mask to the wafer. Image transfer is conceptually divided into two major steps, based on the physical processes involved. The first major step involves transferring the image from the mask to the photo-sensitive emulsion ("photoresist," or often simply "resist") coating the surface of the wafer. This involves choosing the photoresist material and the piece of equipment that will do the actual exposing. Since IC processing is a multi-step process, an important part of this is how well alignment among layers is performed.

The second major step involves transferring the image now on the photoresist to the actual wafer underneath. This is usually performed by developing the photoresist (removing the photoresist that was exposed[1]), hardening the remaining resist, and finally actually performing the processing step which affects only the parts of the wafer not covered by the remaining resist.

5.3 Initial Planning Decisions

Design decisions span over a range from purely numeric calculations, like calculating the required exposure energy from a given graph, to purely heuristic

[1] Developing *positive* resist removes exposed resist; developing *negative* resist removes unexposed resist. Most resists used in modern IC processes are positive although most resists in current use (including older technologies) are negative. All known negative resists are limited at a resolution of about 3 μm [Elli82] and so are unsuitable for modern IC processes.

decisions. This section examines the major initial planning decisions made when an expert IC process designer works with a set of specifications or requirements to plan the general direction that the IC process will take. It is seen that these decisions are often based on experience and "rules of thumb," and that they are best represented in a CAD system by heuristics.

5.3.1 Choosing the Aligner

The first major design decision is usually the choice of the aligner, otherwise known as the stepper or the wafer stepper. This is because any given fabrication facility usually offers few choices due to the very high cost of these machines—often about $1 million to purchase and up to $200,000 per year to maintain and operate for the most advanced aligners. In addition, certain IC process requirements, the most important of which is resolution, often narrow the choice of the aligner to very few candidates. Two major factors influence the choice of an aligner: its optics determine the resolution capability, while its alignment method determines the accuracy with which an exposure is aligned, or registered, over the exposures of previous processing steps. Often physical factors, such as wafer size, material, thickness, and shape, preclude the use of some aligners. In most settings outside the research laboratory, the aligner's throughput (wafers per hour) is also a major consideration.

While specifications are published for the resolution and alignment capabilities of any aligner, experience with these machines tends to dictate the

realistic capabilities used to form decisions. The resolution capability of the GCA Corporation's 4800 Wafer Stepper is published as being 0.8 μm. This is under ideal conditions, with no vertical features on the wafer and no significant reflections of the exposing light by any boundaries under the resist coating. Experience shows, however, that 1.2 μm resolution is the best that can possibly obtained under typical conditions encountered at UC Berkeley's Electronics Research Laboratory (ERL). The best resolution will vary somewhat from this number due to vertical features on the wafer and reflections from boundaries underneath the resist. This is a classic example of heuristic knowledge.

5.3.2 Choosing the Etch Method

The second major decision is usually the etching method (for those processing steps where this is the next step after exposing and developing the resist). The etchant removes the material uncovered by the resist development step. On some IC process steps, especially in analog IC processes where component matching is crucial, this is a critical operation, as it often defines the sizes of devices like transistors and capacitors.

The major decision concerning etching is wet vs. dry. Wet etching refers to the conventional approach where the wafer is immersed in a bath of concentrated acid which attacks the wafer surface where resist has been removed. Wet etching is completely *isotropic*, meaning it has no sense of etch direction. Dry etching is performed by subjecting the wafer to a stream or cloud

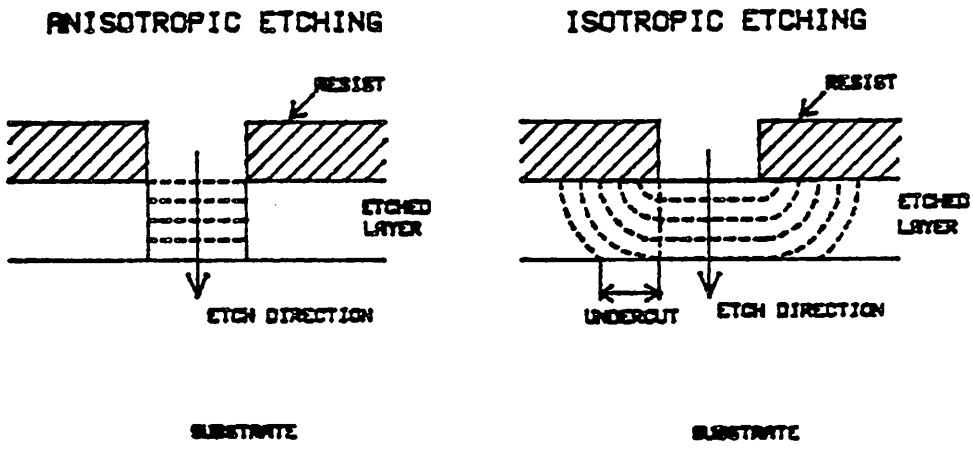

Figure 5.1: Isotropic and anisotropic etching. Note the high sensitivity of undercutting to etch extent for isotropic etching.

of ionized gases which reactively etch the wafer surface. The ionized gases may also be directed by an electric field, thereby being "aimed" at the wafer. A strong electric field will overcome random motion of the ions and direct them toward the wafer. This is called *ion milling*, in reference to the very straight sides cut into the wafer by the ions. Ion milling is highly *anisotropic*, having a highly directed etch direction. Weaker electric fields allow a higher fraction of the ions' energy to be in random directions and result in some *undercutting* of the wafer surface underneath the resist. Figure 5.1 shows the spectrum of isotropy and the resulting undercutting.

The problem with undercutting is that it is very difficult to control its

extent. As seen in Figure 5.1, undercutting is highly sensitive to etch extent for isotropic etch methods. Some undercutting is unavoidable with isotropic etching because the etched layer should be slightly overetched to guarantee full etching across the whole layer. Most IC fabrication processes have a known "etch bias" which is the increase in linewidth due to undercutting. Layout designers must take this etch bias into account when deciding on line widths. Due to thickness differences of the etched layer and fluctuations in etch uniformity across a wafer or even across a single IC, undercut variation can result in poor capacitor value matching, linewidth variation, and transistor size variation.

Another good example of heuristic knowledge is evidenced by the expert IC process designer when making a decision about etching. Professor Ping K. Ko of UC Berkeley's Electronics Research Laboratory, when asked about what type of etching should be used in the example video-speed A/D converter, immediately replied "Dry!" after just hearing that it was to be an analog circuit. The degree of isotropy is still to be determined, however; the general rule in this case is to use the highest degree of anisotropy that has been shown to work for someone else.

5.3.3 Choosing the Resist Scheme

The resist scheme choice, the third and last major initial planning decision, is fairly straightforward but is again based almost entirely on heuristic knowledge. The general rule here is more or less to ignore published specifications

for resists, instead using what has been shown to work under comparable conditions for previous fabricators.

5.4 Refining the Initial Plan

Once the initial planning decisions have been made, the remainder of the design process is an iterative refining and verification process. The design decisions made during the remainder of the design process tend to be less and less slanted toward heuristics and more toward numeric computations. A complete listing of the actual design decisions would be too large for inclusion here, so I will list a few exemplary decisions.

After choosing the resist scheme, for instance, the IC process designer might first want to derive the required resist film thickness to use. Once the type of resist and the current state of the wafer is known, finding the resist thickness is mostly a matter of a few table lookups. Given a resist and desired resist thickness, it is a straightforward curve interpolation exercise to find the spin speed of the resist spinner[2] to obtain the desired thickness. Once this set of design decisions has been completed, the IC process designer could actually perform the resist application operation.

Another design decision that would be analogous is finding the ion energy to use if the IC process calls for some degree of anisotropic dry etching.

---

[2]Resist is usually applied to a wafer by dripping a small quantity of resist onto the center of the wafer and spinning the wafer at high speeds

The first decision to make is more heuristic in nature, which would be estimating the maximum allowable undercut. The second decision would be finding the actual ion energy, which would best be implemented by some table lookups, using tables specific to each type of dry etcher.

5.5 The Example

In reference to the example introduced in Section 1.3 on Page 5, the set of design decisions are first broken into the three fundamental steps, the choices for the aligner, resist, and etch method. Practical and heuristic knowledge form the basis for these decisions. About 95% of the alignment and exposure work in ERL's IC laboratory is done with the GCA 4800. so that is usually the first choice for this step. Its resolution capability is sufficient for most process requirements.

Once the aligner has been chosen (or assumed, as is usually done), the next steps are to make some rough calculations to find how closely the process requirements push the aligner to its resolution limits. These calculations will usually take into account worst-case projected image degradation due to focusing errors and reflections from layer boundaries. The image degradation forms the basis for estimating the potential variation of the projected linewidth. Based on these calculations. the process designer may decide to choose another aligner, or more typically, to perform a special step on this layer that will enhance the projected image and therefore improve the resolution.

The next fundamental step to be chosen is the resist scheme. Again, heuristics usually make this decision. The current favorite resist used at UC Berkeley's Electronics Research Laboratory for high-definition work is the KTI 820 positive resist. A few decisions will be made that will form the rest of the resist scheme, such as whether a special coating will be needed (an anti-reflection coating may be needed if reflections from the wafer surface will seriously degrade the projected image). After the resist scheme itself has been refined, a few calculations need to be performed to find the preferred resist dilution, spin speed for application, and the exposure dose.

The last fundamental step to be chosen and refined is the etch step. For good linewidth control, the etch method needs to be at least partially anisotropic. The only way to perform anisotropic etching at UC Berkeley's ERL is by dry etching. Once this has been decided, the next decision is the choice of gas used in the etcher. Some calculations follow this choice to find what ion energy should be used, the etch rate expected, and the etch time.

It can be seen that the model of selecting the three fundamental IC photolithography steps from libraries and then refining them one step at a time is highly applicable to the real IC process design procedure. It is also clear that different forms of knowledge are used for each decision. Initial decisions tend to be guided by heuristic knowledge, while later decisions to refine the initial ones are largely numeric in nature. Seen in another way, this synthesis procedure begins with a set of strategic decisions that point the solution in a general direction; these are followed by tactical decisions that carry the strategy out.

CHAPTER 6

Overview of Cameo's Structure

Cameo's structure attempts to balance the most important requirements of its application domain using a number of unique techniques. The combination of these techniques into a single integrated and easy to use and understand system appears to be unique and successful.

6.1 Features Differentiating Cameo

Numerous features differentiate Cameo from other CAD systems. Some stem directly from the requirements of its application domain, others are potentially unique features applicable to many CAD systems, while others are adaptations of known features to the special case of IC process CAD. The most important features differentiating Cameo from other CAD systems are:

- Cameo works with a decomposition of the IC photolithography design process into a set of smaller, more bounded design decisions.

- Cameo's large body of knowledge is structured into small bits, each applied to a specific design decision. Each bit of knowledge is implemented in whatever fashion is best for the specific design decision.

- The assigning of different types of knowledge to design decisions is entirely transparent to the user and the rest of the system. Thus new types of knowledge can be added to the system with a minimal amount of work.

- Recognizing that most IC process designs are modifications of earlier designs, an important part of Cameo's data base is a set of libraries of complete, documented lithography steps.

- Few CAD systems allow users to ask why or how certain decisions are made. Novice users need this feature to gain confidence in the system's actions. Every possible design decision that appears on Cameo's screen has an associated "references file" that can be viewed with a single keystroke. This references file lists a summary of notes and other information in a short, easily readable form, and includes a list of further references for users interested in more detail. These references often include local experts.

- Cameo's data base design was driven by the tentative nature of any design process, in which the designer often wishes to explore a number of alternative designs while keeping "back ups" of earlier designs to which he can always retreat. This is done with a minimal penalty in

storage space by taking advantage of the hierarchical inheritance of the underlying frame-based data representation system.

- Cameo keeps as much information about its own structure as possible in frames so that rules can be written that reason about the system itself. One particularly useful application of this structuring is that one can write meta knowledge, or "rules about rules," which can choose the best solution method for a design decision.

The explicit separation of Cameo's knowledge from the mechanisms that use the knowledge is also different from most CAD systems but is a property of expert systems in general. Cameo also has the potential (not yet exploited) of explaining its design decision solutions, also different from most CAD systems but a property of expert systems.

6.2 Overall Description of Cameo

6.2.1 Implementation

Cameo is written in the Heuristic Programming and Representation Language (HPRL), Hewlett-Packard's frame-based expert systems development system [Rose82,Lana83a,Lana83b] and runs on a Hewlett-Packard 9836C computer. HPRL is written in Portable Standard Lisp (PSL)[1], developed at the University of Utah and adapted for the Hewlett-Packard desktop 9836

[1] Portable Standard Lisp (PSL) was developed by the Utah Symbolic Computation Group at the University of Utah and is copyrighted by its authors. It is available as public domain software.

computer. The 9836C computer is a Motorola MC68000-based workstation with capabilities for color graphics, main memory of over 16 megabytes, and interfaces through the Hewlett-Packard Interface Bus (HPIB)[2] to many types of laboratory instruments.

6.2.2 Appearance of Cameo to the User

The version of PSL available for the 9836 computer includes some extensions to PSL's standard editor which take advantage of the high bandwidth to the 9836's screen. One of these extensions is a *browser* facility for many of the selection tasks constantly performed while editing, such as selecting files to edit or buffers to visit, finding documentation, and editing directories. This browser facility is based on the assumption that many tasks are preferably performed interactively as long as the information bandwidth to and from the screen is very high, which is the case with the 9836. Cameo's appearance to the user is based on a special adaptation of the 9836 PSL's browser facility.

This browser facility displays a screen full of lines to the user, where each line represents one specific object on which the user can perform operations. In the case of selecting files to edit, the screen shows a list of files, one per line, and the user can position the cursor over one line and press a key to perform the operation on the corresponding file. Examples of operations are editing or deleting the file, searching for matching file names, or reordering the listing according to size or modification time.

---

[2]Hewlett-Packard's implementation of the IEEE-488 1975 Standard Digital Interface for Programmable Instrumentation.

Cameo works on much the same idea. Each line in the main browser represents one design task of the overall design problem of photolithography. The user may position the cursor over any one visible step and press a key to perform an operation on that step.

Other browsers are used in the system, for instance to display a library of previously used and documented process steps. Again, the user can position the cursor over the line that represents that step and perform an operation like read documentation on the step or choose that step for incorporation into his process.

6.2.3 Organization of the Knowledge and Data Bases

The key element in the overall design of Cameo is to provide a framework for assigning specialized *mini-experts* to specific problems in the overall photolithography design problem. Thus the knowledge base is composed of many small groupings of knowledge. In this system, as opposed to most others, the term "knowledge" does not mean only heuristic knowledge, or knowledge written as expert-system-type rules, but any collection of information or methods that is used to solve problems. The most appropriate type of knowledge is assigned to each specific design task, whether that means plugging values into a formula or calling HPRL's inference engine.

Cameo's data base is composed of five major parts, as follows:

- the *plan skeleton.* This is essentially a template of a complete lithography plan with no data in it. Each slot of this frame (a slot is much

like a field of a record or structure in other programming languages) will represent one specific design task. The plan skeleton reflects the research results of the decomposition of the overall photolithography design task into the separate subproblems. The plan skeleton has very little structure of its own, only providing data storage.

- the evolving set of *working plans* (refer to the short summary of terminology in Section 3.1 on Page 24 for this use of "plan"). These are dynamically changing under the user's control.

- a set of *description frames.* The description frames describe the structure of the plans, both in terms of the time-sequence of the design decisions and the data dependencies among them. Each description frame describes one slot in the plan skeleton, and specifies things like the appearance of that slot in the browser, its dependence on other slots in the plan skeleton, which design decisions to show after this one has been answered, the type of mini-expert to use to find a value for the slot, and many other pieces of information that are needed to tie together the various other parts of the system.

- the *libraries* of previously used and documented steps. The user may browse through these libraries and select a step for incorporation into his lithography steps. This approach was used because many design decisions rely heavily on previously successful designs, and often consist of picking appropriate existing designs and making minor modifications or refinements on them. Thus the idea of a set of step libraries is useful.

- the *reference files*, one of which is assigned to each slot of the plan skeleton. The user may view the reference file for the design decision he is considering at any time. The files are short (less than one screenful if possible) and hold summary information about the design decision plus pointers to further references.

Only the working plans change as the system works on a particular IC process design. All other parts of the data base are static during each use of the system.

6.3 Description of HPRL and PSL

Hewlett-Packard's Heuristic Programming and Representation Language is a frame-based system modeled after FRL [Robe77a,Robe77b]. A *frame* is a data structure that is similar to a structure or record in languages like C or Pascal. Where members of structures or records are called fields, members of frames are called *slots*. Two major differences between frames and structures of other languages are that

- frames can be dynamically changed. They can have slots added and removed after the frame has been defined.

- frame-based systems often support *inheritance*. With inheritance, a frame appears to have slots that were actually defined in a parent. Adding data to a slot in a parent frame makes that data appear to exist in a child as well.

A short example will explain. Say a frame is defined, called "aligner." Slots of this frame may include some named "alignment-accuracy," "alignment-method," and "location." Now define another frame, called "GCA." This is a specific aligner, so the definition of this frame will specify that it shall inherit from the "aligner" frame defined earlier. (In keeping with the terminology of frame-based systems, the "GCA" frame is an *instance* of the "aligner" frame.) The alignment method of a GCA aligner is "Fresnel Zone," so that data will be put in the "GCA" frame. Suppose that in the fabrication line we are describing, all aligners are in the same physical area on the fabrication line and so can be considered to have the same location. To accomplish this, the value "aligner-area" is installed in the "location" slot of the "aligner" frame. Now all frames that inherit data from the "aligner" frame will all have the value "aligner-area" in their "location" slots.

The structuring of data into hierarchical inheritance trees closely resembles how people often think of data. It seems natural to think of things as belonging to classes, and apply general rules to all things in that class. This is how rules can be written in HPRL. A rule might express the fact that all aligners are expensive. Only one rule needs to be written, as long as the data is structured as in the example above. If the program needs to know whether a specific object is expensive, the system might discover that the object is a kind of aligner and that a rule exists that says that aligners are expensive.

Cameo's working plans data base also takes advantage of the hierarchical inheritance provided by HPRL frames. Any new plans that are generated

inherit all the information from the older plans (their parent plans) from which they are generated. The new plans will also add some data of their own. This mimics the exploratory manner in which process designers develop processes, where a single design might be split into several at many points in the design process, in order to try out different alternatives.

The step libraries are good examples of the inheritance technique. Each step in a library is an instance of a blank step frame "template" with no data. Each complete step in the library inherits all of the slots of the step frame template, and provides all its own data.

6.4 Overall Structure

The overall structure of Cameo is shown in Figure 6.1. Major functions are performed by separate parts of the system, leading to higher flexibility and modularity.

The user interacts solely with the *display manager.* A plan is displayed on the computer's screen as a set of lines, one line per problem. The user performs an operation on one of these problems by scrolling a cursor over the plan, leaving it on the line representing the problem, and selecting operations to perform on the plan by typing a single key. The lines representing problems are grouped and indented to reflect the problems' hierarchical structure. Figure 6.2 shows an example of the screen during the system's running.

When the user has selected a problem and has pushed a key to perform an operation, the display manager sends a message to the *program control*

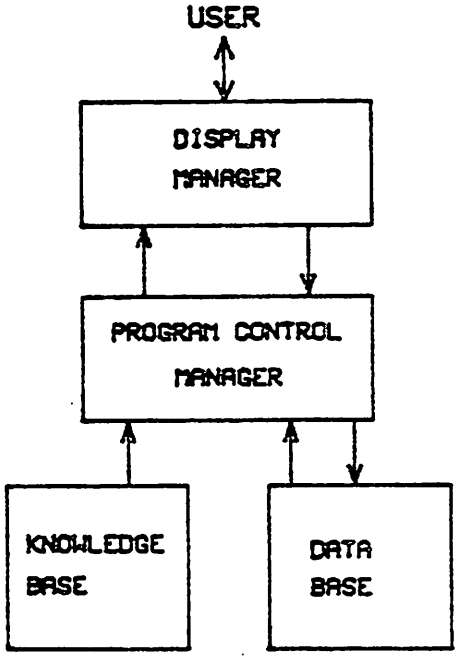

Figure 6.1: Cameo's overall software structure

manager telling it which operation the user selected and which problem to operate on. The program control manager performs the operation, looking up additional information in the *plan skeleton* and the *description frame* for this problem.

For instance, one of the operations the user can perform on a problem is the *derive* operation, which uses the mini-expert for this problem to attempt to solve it. If the user selects the derive operation, the program control manager activates the *mini-expert* found by examining the corresponding

description frame. The mini-expert then attempts to find a solution for the selected problem. When the operation is completed, control returns to the display manager, which updates the screen if necessary and waits for new input from the user.

6.5 The Display Manager

Cameo's display manager is based on the NMODE[3] browser extensions to PSL for the Hewlett Packard 9836 desktop computer. The browser facility is implemented using PSL's object-oriented programming features. An item in a browser is represented by a PSL object which has methods for a few standard messages needed to maintain and update the browser.

An example of the display screen during the running of the system is shown in Figure 6.2. The operations shown in Table 6.1 are currently supported in the most recent version of Cameo. Each operation is performed by positioning the cursor on the desired line and typing the first letter of the operation shown.

6.6 The Program Control Manager

As described in Section 6.5 on Page 66, each item in the browser is represented by a PSL object which responds to a few standard messages related

[3]NMODE is an EMACS-like extension to PSL developed at Hewlett-Packard Laboratories and copyrighted by Hewlett-Packard Company. It is based on an earlier editor, EMODE, developed by William F. Galway at the University of Utah.

Operation	Operation Type*	Description
?	View	Help on using this screen.
Refs	View	Shows a reference file specific to this problem. References files give the user the purpose of the step, special notes on it, references to literature or other people, etc.
Derive	Plan	Have the system use the mini-expert for this problem to attempt to solve it.
Provide	Plan	Allow the user to provide his own solution to this design decision.
Override	Plan	Override (replace) the solution for this design decision. Ideally, this would invalidate other answers that depend on this one. Currently it does only a part of this.
Expand	Display	Shows the next level of detail below the problem.
Collapse	Display	Collapses all levels of detail below the selected problem (in the same plan).
Split	Plan	Copies (splits) the plan of which the selected problem is a part. Results in two identical plans which can be operated on independently.
New	Plan	Builds a new plan that has no problems solved.
Kill	Plan	Kills the entire plan of the selected problem. After user confirmation, it is removed from the data base.

*A *view* operation type temporarily shows another buffer on the screen, a *plan* operation type alters the plan and possibly also the display, and a *display* operation type alters only the display.

Table 6.1: Operations supported by Cameo's Display Manager

to maintaining and updating the browser. Each object represented in the browser has additional methods defined that control the solution of the specific problems it represents. The inheritance feature of PSL objects allows the definition of the common methods in a "generic" object while other objects may inherit all those methods and add their own specific ones. The definitions of these specific objects and their methods form the program control manager.

While each specific type of object may define different methods, all are required to respond to the same messages. For instance, the generic object defines a "solve" method which is invoked when the user chooses to run the "Derive" operation. The "solve" method of the generic object performs some operations common to all solution methods, then sends the "solve-specific" message to itself. Each specific type of object must respond to this "solve-specific" message in whatever way is appropriate for its own type. Defining an entirely new type of object mainly involves defining its own method for the "solve-specific" message. Thus extensions to Cameo are easily accomplished, and the specific type of an object does not need to be known by any part of the rest of the system.

The current version of Cameo defines four specific types of objects, depending on the solution method required. The use of these different solution methods and the transparent manner of assigning them to design problems is unique to Cameo. The four object types defined are:

- *Backward-chain-control*—for problems that require calling HPRL's inference engine.[4]

- *Matrix-control*—for problems that require looking up in a one- or two-dimensional symbolic table. Only exact matches to row and column values are allowed, and no interpolation can be done.

- *Graph-control*—for problems that require looking up in a two- or three-dimensional graph or table. Numeric interpolation is performed.

- *Nothing-control*—for placeholders or outline headings in the browser, where no actual problem is represented but a line of text is desired simply for aiding the displayed structure of the screen.

6.7 The Knowledge Base

Cameo's knowledge base, in contrast to many other systems using heuristics, is a collection of different kinds of knowledge. Each bit of knowledge that is intended to be used for one specific problem is called a *mini-expert.*

Even though most knowledge can be expressed in terms of production rules, often forcing knowledge into a single form is both computationally inefficient and conceptually difficult. The ability to provide different ways of expressing knowledge allows matching the problem type with the most appropriate solution. If the use of different knowledge types is transparent

[4] Backward-chain-control objects are also used for problems that use formulas or call arbitrary procedures. See Section 7.1.4 on Page 94 for a detailed discussion.

to the remainder of the system, the knowledge base can be incrementally changed as needed.

The one type of knowledge most often associated with heuristic-based systems is a collection of heuristic *rules*. In HPRL and many other heuristic programming systems it is possible to group rules into *domains* (sometimes called *rule sets*). For problems which are solved by backward-chaining with a certain rule domain, that rule domain is the mini-expert for that problem. See Code List 6.1 for an example of a rule that is a member of a certain rule domain. As will be described later (Section 6.8.3 on Page 74), some of the knowledge in Cameo's knowledge base is designed to choose from among a number of mini-experts for a specific design decision. This allows meta knowledge, or "rules about rules," to be used, giving Cameo something like strategic knowledge in addition to tactical knowledge.

The heuristic programming system's inference engine can be told which rule domain to use to attempt to solve for a certain goal. Specifying a single rule domain to use limits the number of rules that are used to attempt to satisfy a goal. In addition, it allows the heuristic programming system designers to take advantage of special indexing or hashing schemes to speed rule searching. The most important benefit from the standpoint of Cameo's development is that dividing rules into domains greatly improves the modularity of the knowledge base. This was a critical requirement for Cameo's development, since its design assumes the ability to divide a large body of knowledge into small and manageable bits.

Which Mask Layer to Align to

Current Mask Layer:	Optimizing For: Yield	Linewidth Control
N Well	—	—
Active Area	N Well	N Well
Field Implant	Active Area	Active Area
Gate	Active Area	Active Area
N+ Implant	Active Area	Active Area
P+ Implant	Active Area	Active Area
Contact	Gate	Active Area
Metal	Contact	Contact

Table 6.2: Knowledge in the form of a symbolic table. This table shows the previous mask layer that should be aligned to as a function of which quantity the user is trying to optimize.

Often solutions to a problem can be found without relying on the pattern-matching and goal-solving abilities of artificial intelligence techniques. Instead, an answer can often be found simply by looking in a table. So two other forms of knowledge supported by Cameo are table lookup, one symbolic and the other numeric. The knowledge of a symbolic table is in the form of a table, or *matrix*, as shown in Table 6.2. To find an answer for a problem using a symbolic table, the system must provide exact matches for the row and column. Clearly no interpolation can be done.

The knowledge in a numeric table is similar to the symbolic table but interpolation can be done between adjacent rows and columns. Cameo allows the use of two-dimensional or three-dimensional numeric tables. Three-

dimensional tables are internally expressed as a family of two-dimensional tables.

6.8 The Data Base

Cameo's data base consists of five major components. Four, the *plan skeleton*, the *description frames*, the *step libraries*, and the *reference files*, are static, in that they do not change during a single run of the system. The other major component, the set of evolving *working plans*, is the part of the data base that reflects the current set of IC photolithography designs being considered by the user. Figure 6.3 shows these five elements of the data base.

6.8.1 The Plan Skeleton

Functioning much as a template for the working plans, the plan skeleton provides only the structure of the slots where data will eventually reside in the working plans. Working plans (described later in Section 6.8.2) use the inheritance properties of HPRL frames to inherit the slots from the plan skeleton. Each problem shown in the browser is a single slot in a frame of the plan skeleton.

The structure of the plan skeleton is that of a two-level tree. The top level (root) is an HPRL frame that contains slots for common information about the plan, pointers to major steps of the photolithography process, and pointers to frames that contains requirements, specifications, and information about the current state of the wafer. Nothing is installed in these slots yet, as the plan skeleton functions strictly as a template.

6.8.2 The Working Plans

All *working plans* inherit the slots defined in the plan skeleton. Whenever the user finds or provides an answer to any problem, data is added to a working plan. In addition, when a major step is chosen from a step library (see Section 6.8.4 on Page 76), the name of that step's frame is installed in the current working plan.

The structure of the working plans data base was developed to provide support for the tentative nature of exploratory design. A major requirement of a synthesis system such as Cameo is to allow the user to make tentative decisions, explore their ramifications, and either proceed or retract the decisions. Since a general solution to this problem is a major area of research in itself, only a simple (and limited) capability was built into Cameo.

The user can split a working plan into two identical, independent plans (see Figure 6.4). This is accomplished by building two entirely new working plans, both immediate children of the original working plan the user wants to split. The two new working plans are added to the browser and the original working plan is removed from it. Thus the user cannot change the original working plan (because it is not shown on the browser) but must now work with the new children. The original working plan is not, however, removed from the data base, because it probably contains data that its children inherit.

The user may now work on one plan (splitting it again if he desires) and can always back up to the original state of the plan. Plans may be split as

many times as desired. One may think of splitting a plan as performing a "back up" of it. By using the "Collapse" command of the display manager, the user may collapse the display of any plan on the system's screen down to a single line, leaving the plan accessible but greatly reduces clutter on the screen. Thus the user may make intermediate copies of plans but can continue to concentrate on more up-to-date copies.

Sometimes a mini-expert will return more than one possible answer to a problem. What this really means is that more than one working plan may be a valid solution to the user's design problem, so the logically correct action is to split the current working plan once for each answer. First, though, the user is shown all the multiple solutions derived by the system, and may choose any or all of them for further consideration. This again allows the user more control over Cameo's reasoning process and prevents runaway plan splits.

Each new working plan is a direct child of the original working plan. As in two-way splitting under the user's control, the original working plan is removed from the browser but remains in the data base.

6.8.3 The Description Frames

Description frames are the "glue" of the system. They contain the information that ties together and gives structure to the data in the plan skeleton and working plans, and associates mini-experts in the knowledge base with slots in the plan skeleton. One description frame is required for each problem

that is displayed in the browser. The description frames and their contents are static, in that they do not change as the system runs.

Some of the information given in a description frame includes

- The type of mini-expert to use to find an answer for this problem.
- Any further information needed by the mini-expert. This information will depend on the type of mini-expert.
- How the problem is to appear in the browser.
- A listing of the problems which may appear on the browser once a value for this problem is found.
- A listing of the slots which the answer to this problem depends on.

Since the basic structure, dependencies, and other vital information about Cameo is stored in these dependency frames, rules can be written about these as well. This kind of knowledge, knowledge about its own structure, is one criterion that is sometimes used to distinguish an expert system from other kinds of systems. One useful application of this *meta knowledge* is to enable the system to pick a specific mini-expert from a number of possible ones for a certain problem. One particularly useful application has been to select a certain curve from a family of curves of a three-dimensional graph.

6.8.4 The Step Libraries

The last major component of Cameo's data base is the step libraries. These are sets of frames with many values (answers) already provided. One library of steps exists for each of the three major steps of photolithography, the resist scheme, the alignment and exposure step, and the etch step. The user may browse through each library, find references on each step in the library, and may also read in a personal library from a floppy disk.

These frames can be linked into a working plan. The advantage of using libraries of steps, especially in an area like photolithography, is that a complete, tested, and documented major step can be added to anyone's plan. Information and experience in the form of step libraries can be passed along to future researchers.

6.8.5 The Reference Files

One reference file is provided for every design decision ever shown on Cameo's screen. The reference file is supposed to answer the "Why?" questions about the decision, or supply enough information for the user to find out more. The user can view the reference file for any item on Cameo's screen by positioning the cursor on that item and typing the R key for "References." The reference files are intended to be easily read and understood. They are kept to a single screenful if at all possible.

6.9 Chapter Summary

Numerous features differentiate Cameo from other CAD systems. Many are unique because the combination of its application (IC process design) and implementation as a flexible frame-based expert system is unique. Some features, however, should be applicable to more general CAD problems.

The most important features are those having to do with the decomposition of the very complex IC process design problem into individually manageable design decisions. The large body of knowledge about IC process design is split into bits of knowledge, each of which is assigned to a single design decision. Libraries of previously successful major steps are provided since much IC process design is modification of previous processes. All these features were driven by Cameo's specific application.

Every design decision displayed on Cameo's screen has an associated references file which answers questions like "How?" and "Why?" Cameo's data base design reflects the tentative and exploratory nature of IC process design. Use of Cameo is highly interactive and the user feels in control of the system at all times. These features should be applicable to other CAD systems beyond IC process design.

Together, these features enable Cameo to be used easily by novice IC process designers. It is hoped that extensions to Cameo will enable it to be equally useful to experts.

```
    CAMEO PHOTOLITHOGRAPHY DESIGN

PLAN: Initial Plan
    --PROCESS REQUIREMENTS--
        Alignment Accuracy (3 sigma): 1 micron
        Field Size: 5 mm
        Minimum Feature Size: 2 microns
        Optimizing for: DIMENSION-CONTROL
    --WAFER STATE--   [more...]
    Aligning and Exposure: GCA-1
        Machine: GCA
            Type: PROJECTION
            Field Size: 10 mm
            Resolution: 1.2 microns
            Alignment Error (3-sigma): 0.3 microns
            Reduction Ratio: 10
            Exposure Wavelength: 4360 angstroms
            Numerical Aperture: 0.28
        --Contrast and Linewidth Variation Calculations--   [more...]
        Align To
    Resist Scheme: KTI820-1
        --Preparation--
        Resist: KTI820
            Resist Thickness: 1.04 microns
                Viscosity: 20.0 cst
                    Spin Speed: 4928.47143 RPM
            Refractive Index: 1.62   [more...]
    Etch Method: DRY-1   [more...]


--------------------------------------------------------------------------
? Refs [Derive/Provide/Override] [Expand/Collapse] [Split] [New/Kill-plan]
```

Figure 6.2: Screen example from actual running of Cameo. Note grouping and indentation to reflect hierarchical structure of the design process. Available commands are at the bottom of the screen. Items with [more...] can be further expanded.

```
% Define the rule domain aligning-step-domain
(rule-domain aligning-step-domain backward-chain)

% Define the generic rule aligning-step-rule
(fassert aligning-step-rule
         (ako ($value (backward-chain-rule)))
         (domain ($value (aligning-step-domain))))

% When to use the GCA stepper
(rule use-gca aligning-step-rule
      (premise (and (?process-plan specs ?specs)
                    (?process-plan wafer ?wafer)
                    (?specs minimum-linewidth ?lw
                                              (geq ?lw 1.2))
                    (?wafer field-size ?wfs (leq ?wfs 10))
                    (*usable-equipment* aligner gca)))
      (conclusion (?process-plan aligning gca)))
```

Code List 6.1: example of a rule in the knowledge base. First is the definition of a new rule domain, aligning-step-domain. Next is the definition of a *generic rule*, aligning-step-rule, all of whose children will inherit its domain. Finally, the rule, use-gca, is defined to be type of aligning-step-rule, and so it is a member of the aligning-step-domain domain. This rule says that the GCA is a valid aligner to use if the minimum linewidth is greater than 1.2 microns, the field size less than 10mm, and the GCA can be used by the user.

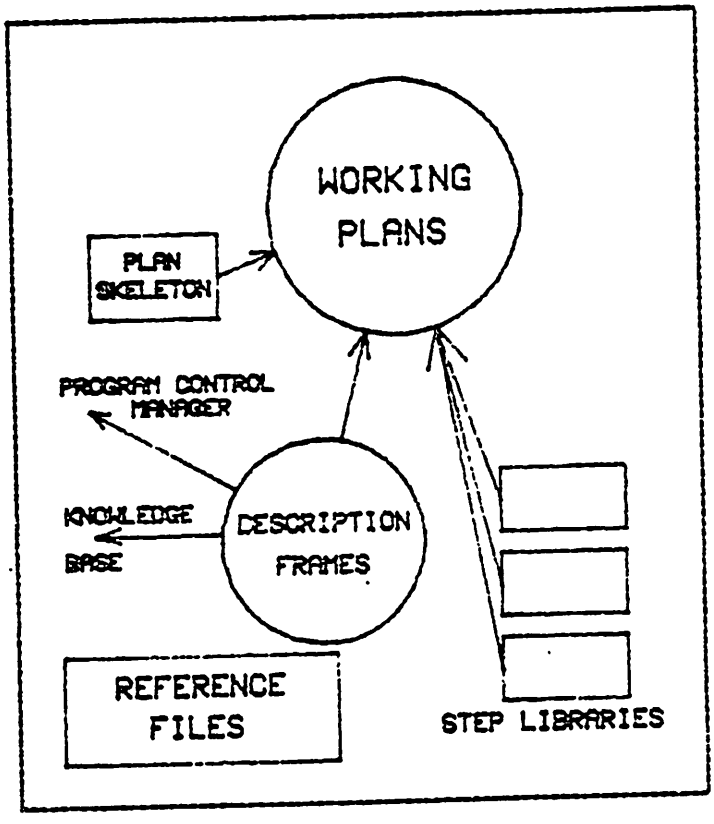

Figure 6.3: The main components of the data base.

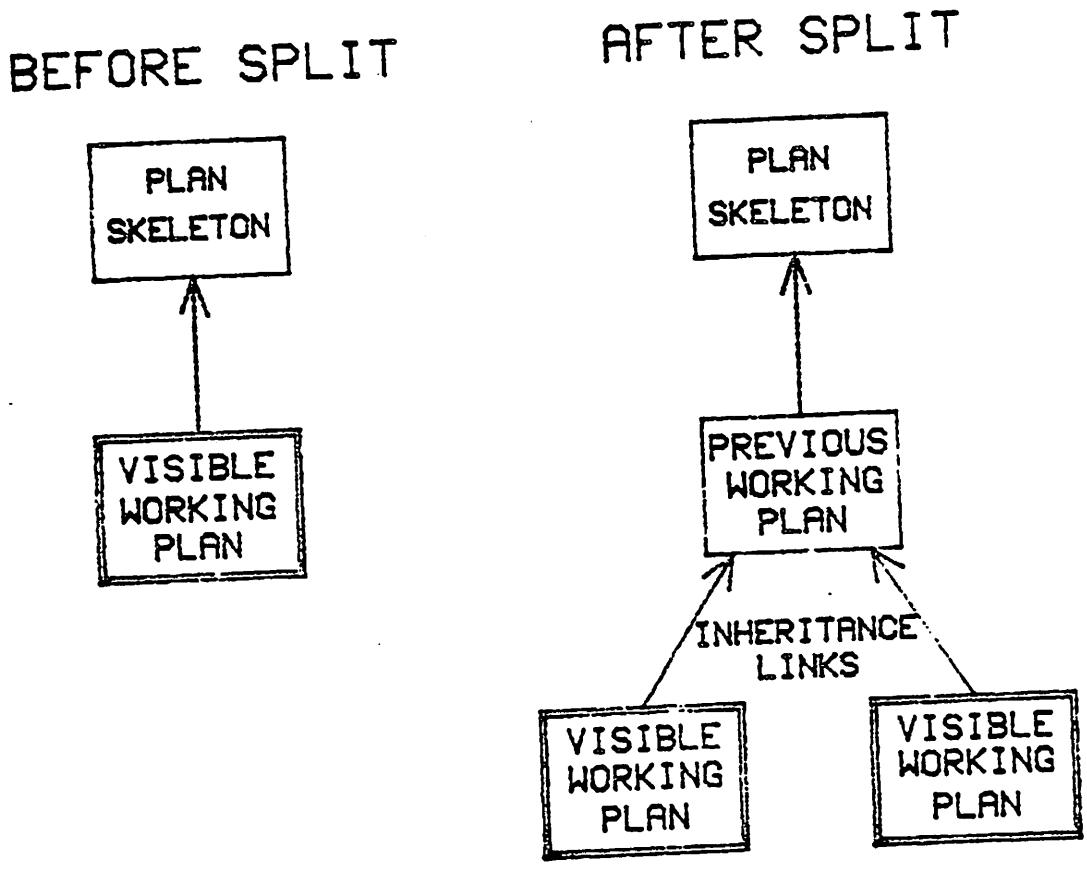

Figure 6.4: Splitting a working plan. Each working plan splits into two children, and the previous working plan is removed from the browser.

CHAPTER 7

Cameo's Knowledge Base

Cameo's knowledge base contains the "knowledge" which makes it an expert in the domain of choosing and refining steps of an IC photolithography plan. Its main features are its modularity and its ability to express different kinds of knowledge in different ways. Each bit, or module, of special knowledge is termed a "mini-expert."

Examples and listings of bits of knowledge are given throughout this chapter. It should be understood that Cameo is a constantly evolving system, and that the specific code fragments or listings shown in this chapter may not be what is contained in the most recent version of the system. They should be considered only as informative illustrations of the principles being discussed.

7.1 Kinds of Knowledge

Knowledge about any domain takes many forms. Most expert systems to date have used only one form of knowledge, heuristics. Most of these express their knowledge in terms of *production rules*, or *productions*, which are *IF ... THEN ...* patterns. While production rules are ideal for expressing the "rules of thumb" that make up a great deal of an expert's knowledge, the best approach in building a knowledge base is to use production rules as yet another programming tool for knowledge and data representation in addition to more conventional tools like tables, graphs, and formulas. An expert system that can truly be claimed to perform as an expert must have all these forms of knowledge equally available to it.

Four major classifications of knowledge were identified as being particularly necessary for an IC process CAD system: heuristics, numeric tables, symbolic tables, and formulas. The kind of knowledge used for each design decision is based on the method an experienced IC process designer uses to solve the same problem. The types of knowledge can be expected to grow as more experience is gained in entering knowledge into the system. One type of knowledge that might be a valuable addition is an interface to an external, laboratory-wide data base system that could provide current data such as equipment status, user permissions on equipment, reservation schedules on equipment, current and projected inventory of critical supplies and chemicals.

Each mini-expert type might be best considered as an abstract data type. As will be explained in detail in this chapter, each type of mini-expert has a certain internal data structure and a few special algorithms that operate on that data structure to provide answers to design decisions.

7.1.1 Heuristic Knowledge

As seen in Section 5.3 on Page 48, many of the initial planning decisions tend to be made on the basis of largely heuristic knowledge. Thus the inclusion of a way to represent heuristic knowledge is a critical factor in the success of this system.

In HPRL, heuristic knowledge is expressed as a Lisp form. Code List 7.1 shows an example of a rule domain and rule definition in HPRL. (This is the same code list as Code List 6.1 on Page 79.) The first line of the rule specifies that the rule is named `use-gca`, and that it is a member of the `aligning-step-rule` rule domain.

The premise is satisfied if all of the conditions listed are satisfied: the specifications given to the system show the minimum required linewidth is greater than or equal to 1.2 μm, the field size (maximum length or width of the finished IC) is less than or equal to 10 mm, and the user is listed as a valid user of the GCA aligner. When the premise is satisfied, the conclusion installs the value `gca` into the `aligning` slot of the `process-plan` frame.

HPRL rules allow local variables to be used within each rule. Local variables in a rule are introduced by a question mark (?). Each time a

```
% Define the rule domain aligning-step-domain
(rule-domain aligning-step-domain backward-chain)

% Define the generic rule aligning-step-rule
(fassert aligning-step-rule
         (ako ($value (backward-chain-rule)))
         (domain ($value (aligning-step-domain))))

% When to use the GCA stepper
(rule use-gca aligning-step-rule
      (premise (and (?process-plan specs ?specs)
                    (?process-plan wafer ?wafer)
                    (?specs minimum-linewidth ?lw
                                          (geq ?lw 1.2))
                    (?wafer field-size ?wfs (leq ?wfs 10))
                    (*usable-equipment* aligner gca)))
      (conclusion (?process-plan aligning gca)))
```

Code List 7.1: Example of a rule and rule domain in the knowledge base.

rule is being considered, it may be applied to many different frames in the data base. The rule is "instantiated" once each time a unique potential match is found for it. Each instance of the rule then has its local variables assigned specific values according to the data in the frames that this instance is working with.

For instance, assume that the rule in Code List 7.1 is being run by HPRL's inference engine. and the process plan it is being applied to is process-plan-4. Assume further that the frame holding the specs step

(the "Process Requirements") is named specs-4 and the frame holding the wafer step (the "Wafer State") is wafer-4. In the frame specs-4 the minimum-linewidth slot has the value 2.0 and the field-size slot has the value 7. The variables in this instance of this rule will now have the following bindings, or values:

Rule's Local Variable	Binding
?process-plan	process-plan-4
?specs	specs-4
?wafer	wafer-4
?lw	2.0
?wfs	7

The second clause of the rule's premise,

```
(?specs minimum-linewidth ?lw (geq ?lw 1.2))
```

contains an extra form which only allows the clause to conclude successfully if the current binding of the local variable ?lw is greater than or equal to 1.2. The third clause has a similar form that requires the wafer's field size to be less than or equal to 10 mm.

Each HPRL rule domain is one mini-expert. The rules contained in a rule domain form a self-contained bit of knowledge for the design decision they are assigned to. HPRL's inference engine may be instructed to consider only the rules contained in a single rule domain while performing its backward chaining, by assigning the name of the rule domain to a global variable:

```
(setq *backward-chain-domain* 'aligning-step-domain)
```

This requires that HPRL only use rules contained in the rule domain `aligning-step-domain`.

A mini-expert type might be considered as being an abstract data type. The code shown in Code List 7.2 is the code for the `solve-specific` method of the mini-expert of type `backward-chain-control`. This is the major algorithm corresponding to the abstract data type. The major work in the method is in the call to the `solve-all` function, a built-in HPRL function that runs the inference engine on the goal given as the argument and using only the rules contained in the rule domain given by the value of the `*backward-chain-domain*` variable.

7.1.2 Numeric Tables, or Graphs

In addition to heuristic knowledge, much of the information used by an expert IC process designer is in the form of numeric tables. Once an initial IC process design has been sketched out, an expert will often refer to tables or graphs to derive specific values for a design decision. For instance, once a resist has been chosen and a desired film thickness has been decided on, it is a simple matter of looking up in a published table or graph to find the spin speed at which the resist should be applied to the wafer. The problem is the availability of this data to the researcher.

Cameo's knowledge base contains data in the form of two- or three-dimensional numeric tables, and can find values from these tables by using simple linear interpolation. Each numeric table is a separate mini-expert

in the system's knowledge base. From the user's point of view, there is no difference between finding information with heuristic knowledge or a numeric operation.

Code List 7.3 shows an example of the definition of a three-dimensional numeric table (or graph) in Cameo's knowledge base. Once the graph is entered into the system by the commands shown in the code list, a lookup is performed by sending the newly created graph object the "lookup" message with arguments for the thickness and viscosity values. A separate function, called `find-graph-by-name`, finds the appropriate graph given its name. The entire lookup operation is performed by issuing the following command inside Cameo's Lisp environment:

```
(=> (find-graph-by-name 'kti820-spin-speed-graph)
    lookup 20.0 1.0)
```

The function whose name is "=>" is how message sending is invoked in PSL. Its first argument, the result of the function call

```
(find-graph-by-name 'kti820-spin-speed-graph)
```

is the object which is to receive the message; its second argument, `lookup`, is the actual message; and the remaining arguments are arguments of the message itself. The first one, `20.0`, is the viscosity curve to use, and the second, `1.0`, is the desired thickness.

The `solve-specific` method for the `three-d-graph-control` object is shown in Code List 7.4. Thus the `three-d-graph-control` mini-expert

type can be considered as an abstract data type that represents data in the form of a numeric table and supports the `solve-specific` algorithm.

This graph's `lookup` method, which is defined in the definition of the generic object `three-d-graph`, performs a linear interpolation between x-axis points and between curves of the graph. In this case, there is an exact match between the curve value, 20.0, and a curve in the graph, but interpolation is required between two points specified on this curve, (1.03 5000.0) and (0.96 6000.0). The value returned by the message in this example is a spin speed of 5428.5714. Only the first two digits are meaningful for most resist spinners.

A graph in Cameo's knowledge base is viewed on a graphics screen when the user requests further references on the corresponding design decision. A plot of the graph defined in Code List 7.3 is shown in Figure 7.1. This plot appears on a separate graphics screen when the user types the R key for references on this design decision.

Graphs with only one curve may be entered into Cameo's knowledge base either as three-dimensional graphs like the example above but with just one curve, or as an object of type "two-d-graph." From the user's point of view, there is no difference. From the programmer's point of view, the only differences are a slight syntactic difference in the initialization (since there is only one curve, the initialization of the curve variable is slightly simpler), and the `lookup` message requires only one argument, the x-axis value.

Data that are best represented by graphs are purely numeric relationships that cannot be expressed analytically with the required accuracy.

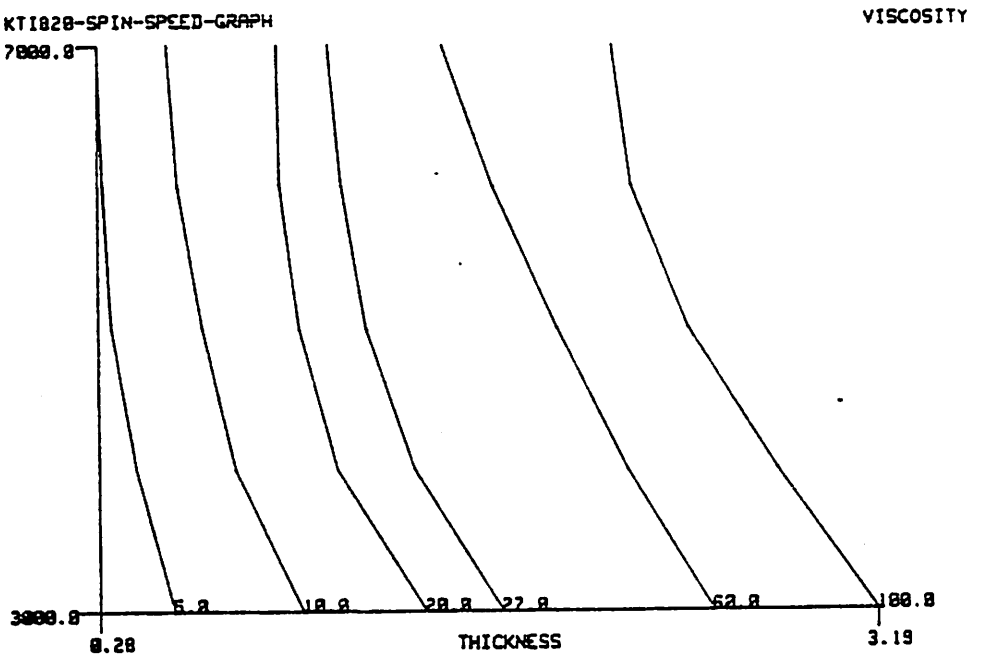

Figure 7.1: Plot of spin speed graph for KTI 820 resist. Each curve corresponds to one viscosity value, shown at the bottom of each curve.

Graphs do, however, have a limited numeric range. They also take up much more memory than a simple analytic relationship, for example. Other mini-experts that are (or could be) expressed as either two- or three-dimensional graphs include:

- Projected image contrast as a function of line pitch and optical system defocus.
- Exposure time as a function of resist absorption and exposure energy

density of the aligner.

- Required degree of anisotropy as a function of etched-film thickness and desired maximum undercut.
- Developing time as a function of resist thickness and exposure dose.
- Etch time (for wet etching) as a function of etched-film thickness and etchant concentration.
- Etch time (for anisotropic dry etching) as a function of etched-film thickness and ion energy.
- Etch time (for isotropic dry etching) as a function of etched-film thickness.

Note that the design decisions that are best implemented by graph lookup are all relatively low-level decisions, occur late in the design process, and are involved mainly in refining the plan. The data expressed by the graph is usually data that appears as a graph or as numeric tables in literature. This was a general observation made throughout this research. It is an important observation because of the consistent manner in which it appears to be true.

7.1.3 Symbolic Tables

Symbolic tables are similar to the numeric tables of Section 7.1.2 but have a few significant differences, the most important of which is that exact matches

to values in the symbolic table are required. This implies that no interpolation can be performed. Each symbolic table is a mini-expert.

An example of the use of a symbolic table is when the user asks Cameo to derive which previous layer the current mask should be aligned to. This is a critical decision which involves a basic trade off between the finished circuit's performance and manufacturing yield. When one mask layer is aligned to an existing layer on the wafer, the alignment error between those two layers is minimized. If the mask is aligned to layer A which was aligned to layer B, the maximum alignment error between the current layer and layer B will be greater than if the current layer were aligned directly to layer B.

The yield-performance tradeoff occurs because optimizing performance requires minimizing overlap areas between layers, while optimizing yield requires maximizing some of these areas. Due to accumulating alignment errors, critical layers whose feature sizes need to minimized for optimizing performance need to be aligned to each other. This increases the alignment errors of these layers to other layers. If the alignment errors accumulate sufficiently, two unrelated features on the wafer may be connected, causing a catastrophic failure.

The knowledge involved in deciding which layer to align to can be expressed as a simple symbolic table. The table in Table 7.1 shows the knowledge about aligning to layers for a simple single-gate-layer CMOS process (this is the same table as Table 6.2 on Page 71). An excerpt from the actual code that defines this table is shown in Code List 7.5. Here again an object

Which Mask Layer to Align to

Current Mask Layer:	Optimizing For: Yield	Performance
N Well	—	—
Active Area	N Well	N Well
Field Implant	Active Area	Active Area
Gate	Active Area	Active Area
N+ Implant	Active Area	Active Area
P+ Implant	Active Area	Active Area
Contact	Gate	Active Area
Metal	Contact	Contact

Table 7.1: knowledge in the form of a symbolic table. This table shows the previous mask layer that should be aligned to as a function of which quantity the user is trying to optimize.

of type `matrix` is instantiated. The next three lines initialize the variables called `matrix-name`, `columns-label`, and `rows-label`, and then the variable `init-values` is initialized with the list of three-element lists, each of the form

```
(value row column)
```

Again, the way to find a value is by sending the `lookup` message with two arguments, the row and column values, to the object representing this symbolic table.

7.1.4 Formulas and Arbitrary Procedure Calls

For knowledge to be best expressed as a formula, the relationship between the independent variable(s) and the dependent variables must be very close to the behavior of the physical phenomenon over a wide range of variable values. The formula should probably express knowledge about ideal physical phenomena, such as an inverse square law of light intensity. The relationship seen earlier with spin speed as a function of resist viscosity and desired film thickness would not be a suitable candidate because the relationship does not correspond closely to an analytic formula. It is easier and more "natural" to express this particular knowledge as a numeric graph. Knowledge in the form of a formula will probably be much more compact in memory than a graph, however, so a tradeoff must be made between accuracy and size.

In Cameo formulas and procedure calls are handled by a special use of backward-chaining rules. HPRL's rule syntax allows formulas and arbitrary procedure calls to be invoked a simple and elegant way while the inference engine runs. An example of a mini-expert implemented as a formula is shown in Code List 7.6. If all clauses of the premise of this rule return true, the rule concludes. This then calls the function * with two arguments, the current binding of `?ms` (1.2) and 2. The function's return value is stored in the `thickness` slot of the `resist-scheme-4` frame.

The function called to provide the data in the conclusion of an HPRL rule may be any function whatsoever with any combination of arguments. This is how arbitrarily complex procedures are used as mini-experts for Cameo.

7.2 Meta Knowledge

Meta knowledge is knowledge about knowledge. In Cameo, meta knowledge is used to choose a mini-expert from among a number of potential mini-experts for a specific design decision. Other applications for meta knowledge abound; much current research is investigating possible applications (see Chapter 7 of [Haye83]). The general idea of a system using meta knowledge to reason about and modify its own structure is appealing for many applications.

In the method for a "solve" message, if the mini-expert for the current design decision has not been assigned yet, Cameo changes the backward chaining domain to `select-mini-expert-domain`, which contains all the rules about choosing mini-experts, and tries to find the appropriate mini-expert by backward chaining. This situation occurs commonly, since the three major photolithography steps are chosen from a library either by Cameo or the user, and many further decisions will depend on this choice.

For instance, while the ideal photoresist might be perfectly characterizable for the purpose of determining the spin speed resulting in a desired film thickness, real resists are not so ideal. Each type and brand of resist has different curves relating viscosity, film thickness, and spin speed. The choice of mini-expert for this design decision cannot be made until the resist scheme is chosen. The rule shown in Code List 7.7, a member of the `select-mini-expert-domain`, says that the graph

kti820-spin-speed-graph should become the mini-expert for this problem if this resist scheme's resist is kti820.

7.3 Example

The example introduced in Section 1.3 on Page 5 presents some informative examples of the kinds of knowledge expressed in Cameo. Some of the design decisions relating to choosing and refining the resist scheme are presented here.

The first decision to make about a resist scheme is choosing the actual resist material. This decision is based largely on heuristics: the usual approach is to use the same resist that someone else with similar requirements had used successfully before. At the time this system was being developed, a favorite resist for any high-definition work over about 1 μm was KTI Chemicals's KTI 820 positive photoresist. Thus one rule in the resist-scheme-step-domain is

```
(rule use-kti820 resist-scheme-step-rule
  (premise (in specs minimum-linewidth ?mlw (geq ?mlw 1.0)))
  (conclusion (?process-plan resist-scheme kti820)))
```

When HPRL's backward chaining begins to solve for the resist-scheme slot of process-plan with the appropriate backward chaining domain set, it will consider this rule. The rule will succeed as long as the minimum-linewidth slot in the specs frame is greater than or equal to 1, which for this process

is equal to 2. When this rule succeeds and the backward chaining process installs the value kti820 in the slot, a function that is called in this case will find the complete library step associated with kti820 and install that step in the process plan.

Other resist schemes may also be potential candidates. When more than one resist scheme is found by HPRL's backward chaining, the user is shown the choices and is asked which he would like to consider further. The current plan is copied once for each user choice, and one choice is installed in each newly generated plan. Thus the user has one independent plan for each choice that he would like to consider further.

Once the resist is known, one of the next steps is to find out how to deposit the resist on the wafer's surface. Design decisions for which a value has been found can be "expanded," meaning the design decisions underneath it in the outline format of the plan can be exposed. Exposing as many items as possible, we run into some that have not been given values yet either by our actions or by inheritance from the step chosen from the resist scheme library.

One of these design decisions is finding out if a special adhesion film called HMDS should be deposited on the wafer before the resist is applied. HMDS improves the adhesion of a resist to the wafer surface, which is important if resist liftoff[1] is to be avoided. The rule for deciding on using HMDS is shown in Code List 7.8. In this example, HMDS would definitely

[1] Resist liftoff occurs when the etching action cuts underneath the resist layer and begins to etch there. Typically the resist "lifts off" from the wafer, and the etching quickly proceeds along underneath the resist.

be needed because the minimum linewidth requirement is only 2.0 μm and the optimized quantity is dimension control.

The next step is actually planning how to apply the resist. The first decision is the resist's thickness. Again, a heuristic rule decides this. The rule will compute a thickness that is exactly 2.3 times the maximum vertical step on the wafer, but in no event will choose a thickness under 1.2 μm.[2]

Now that the desired thickness of the resist has been found, the most appropriate of the standard viscosities available is chosen. The heuristic for choosing the viscosity is that the best results will be obtained by using the thickest possible resist spun on at a speed under 7000 RPM. This is because higher-viscosity resists result in more even coverage of vertical steps on the wafer surface.

The mini-expert for this decision is a rule domain which contains one rule for each standard viscosity. Each rule's premise matches the desired thickness to a range known to be good for the corresponding viscosity; it will succeed if the desired resist thickness falls in that range. An example is shown in Code List 7.9 for the KTI 820 resist.

Now that the viscosity curve has been chosen, all that remains is to find the spin speed of the resist spinner. This is a simple numeric graph lookup with interpolation. Thus the mini-expert for this last design decision is a three-dimensional graph. First a rule in the `select-mini-expert-domain`

[2]This heuristic is one used by Professor A.R. Neureuther of UC Berkeley's Electronics Research Laboratory.

is consulted to find the specific graph for the KTI 820 resist, and the lookup is performed by sending the lookup message to that graph.

To perform all the operations described here, the user only would have had to know two commands. The first, expanding an item on Cameo's display, uncovers the next set of design decisions. The second is the "derive" command which calls the mini-expert for the current design decision to find an answer to it. By knowing two more commands, the user could have requested further references on the current design decision if he had any questions, and could have circumvented any or all of the mini-experts and provided his own solutions.

7.4 Chapter Summary

The ability to assign specialized bits of knowledge, or "mini-experts." to particular design decisions is an important requirement for a CAD expert system. The kind of knowledge assigned to a certain design decision should reflect the method an experienced designer uses to solve the problem. Heuristic knowledge, although necessary, is not sufficient to form a complete knowledge base for a CAD expert system. Other forms of knowledge that have been shown to be valuable are numeric and symbolic tables and formulas. Another form of knowledge not investigated in this research but expected to be of great future value is an interface to a laboratory-wide data base system to provide data about the laboratory itself such as equipment status and inventory.

Most early decisions in the design process tend to be based on heuristic knowledge. The ability to make these decisions is often considered to separate the expert from the novice designer. Later decisions, involved more with refining a chosen design, use largely algorithmic or numeric techniques. This observation was made consistently throughout this research. This can be considered analogous to forming plans in other fields, where broad strategic decisions are made first, then tactical decisions are made to refine the earlier decisions.

A table summarizing these observations is shown in Table 7.2.

```
(defmethod (backward-chain-control solve-specific) ()
  (let* ((fr (=> self frame))
         (sl (=> self slot))
         (dont-ask (member 'dont (fdata-only fr sl '$ask))))

    % Only go through this if the mini-expert is already known.
    % It should have been found by the generic object's solve method
    % if it was not given statically in the description frame.
   (when (=> self mini-expert)
     (setq *backward-chain-domain* (=> self mini-expert))
     (when solve-init
       (eval (list solve-init self)))

       % First turn off asking the user for this slot (HPRL's default),
       % if it doesn't say to ask first.
       % Then solve for all possible values, re-enable asking
       % if it was enabled before, and
       % run the solve-finish function if given.
     (when (null dont-ask)
       (fput-datum fr sl '$ask 'dont))
     (setq results (solve-all '(,fr ,sl ?x)))
     (when (null dont-ask)
       (fremove fr sl '$ask 'dont))

     (when results
       (=> self set-previously-solved t))
     (when solve-finish
       (eval (list solve-finish self))))))
```

Code List 7.2: The `solve-specific` method for backward chaining. This is the major algorithm supporting the heuristic knowledge mini-expert's representation.

```
(add-graph-to-list
  (make-instance 'three-d-graph
    'graph-name 'kti820-spin-speed-graph
    'x-axis 'thickness
    'family-var 'viscosity
    'curves '((5.0 (0.56 3000.0) (0.42 4000.0) (0.33 5000.0)
                   (0.30 6000.0) (0.28 7000.0))
              (10.0 (1.04 3000.0) (0.79 4000.0) (0.67 5000.0)
                    (0.58 6000.0) (0.54 7000.0))
              (20.0 (1.50 3000.0) (1.17 4000.0) (1.03 5000.0)
                    (0.96 6000.0) (0.95 7000.0))
              (27.0 (1.79 3000.0) (1.46 4000.0) (1.28 5000.0)
                    (1.19 6000.0) (1.14 7000.0))
              (60.0 (2.58 3000.0) (2.26 4000.0) (2.00 5000.0)
                    (1.76 6000.0) (1.57 7000.0))
              (100.0 (3.19 3000.0) (2.82 4000.0) (2.49 5000.0)
                     (2.28 6000.0) (2.21 7000.0)))))
```

Code List 7.3: Example of a three-dimensional numeric table. Variable initialization is performed from line 3 to the end, the last initialization of which is the family of curves comprising the table. This table contains one curve per viscosity value, each of which relates the thickness (in μm) of KTI 820 photoresist for values of spin speed ranging from 3000 to 7000 rpm. The graph is internally stored as an object which inherits variables and methods from the generic object three-d-graph.

```
(defmethod (three-d-graph-control solve-specific) ()
  (let* ((graph (find-graph-by-name (=> self mini-expert)))
         (ref-frame (=> self frame))
         (x-fr (frame-in-same-plan x-frame ref-frame))
         (z-fr (frame-in-same-plan z-frame ref-frame))
         (val nil))
    (cond ((null (and graph x-fr z-fr)) nil)
          ((null
            (setq val
              (=> graph lookup (fvalue-only z-fr z-slot)
                               (fvalue-only x-fr x-slot)))) nil)
          (t (fput-value (=> self frame) (=> self slot) val)))))
```

Code List 7.4: The `solve-specific` method as implemented for the `three-d-graph-control` object. The first lines find the graph to use and the names of the specific frames in the current plan where values for lookup are to be found. After making certain all the frames and the graph exist, the `lookup` message is sent to the graph and the results, if any, installed in the current frame.

```
(make-instance 'matrix
  'matrix-name 'align-to-layer
  'columns-label 'current-layer
  'rows-label 'optimizing
  'init-values '((active-area gate yield)
                 (gate contact yield)
                 (active-area gate performance)
                 (active-area contact performance)))
```

Code List 7.5: An excerpt from the code defining a symbolic table.

```
(rule resist-thickness-rule resist-thickness-domain
  (premise (and (?process-plan wafer ?wafer)
                (?process-plan resist-scheme ?resist-scheme)
                (?wafer maximum-step ?ms (plusp ?ms))))
  (conclusion (?resist-scheme thickness ~(* ?ms 2))))
```

Code List 7.6: Example of a formula mini-expert. The rule is used only for type- and range-checking of the procedure's arguments. The rule concludes only when the local variables' bindings have been found to conform to the requirements in the premise. When the rule concludes, it calls the function * (multiplication) and "splices in" the function's return value.

```
(rule kti-820-spin-speed-rule select-mini-expert-rule
  (premise (and (*system* current-step-being-solved-for
                          (resist-scheme thickness))
                (?resist-scheme resist kti820)))
  (conclusion
    (?resist-scheme mini-expert kti820-spin-speed-graph)))
```

Code List 7.7: Example of meta knowledge rule. This rule says that if the current step (design decision) being worked on is the thickness of the resist-scheme, and the resist is kti820, then the mini-expert to use is kti820-spin-speed-graph.

```
(rule need-hmds need-hmds-rule
  (premise
    (or
     (and
      (or (in specs optimized-quantity dimension-control)
          (in specs optimized-quantity linewidth-minimization))
      (in specs minimum-linewidth ?mlw (leq ?mlw 5.0))
      (in specs current-layer ?cl (or (equal ?cl 'gate)
                                      (equal ?cl 'contact)
                                      (equal ?cl 'metal))))
     (and
      (or (in specs optimized-quantity dimension-control)
          (in specs optimized-quantity linewidth-minimization))
      (in specs minimum-linewidth ?mlw (leq ?mlw 3.0)))))
  (conclusion (?resist-scheme hmds-needed yes)))
```

Code List 7.8: Rule for deciding whether HMDS is needed. This is a fairly complex rule. It will succeed if the optimized-quantity is dimension-control or linewidth-minimization and the minimum linewidth requirement is 3.0 μm or less. If the linewidth requirement is greater than 3.0 μm but less than or equal to 5.0 μm, the rule will still succeed if the current mask layer is one of those listed.

```
(rule kti-use-viscosity-20-rule kti-820-viscosity-rule
  (premise (?resist-scheme thickness ?th (and (>= ?th 0.95)
                                              (< ?th 1.14))))
  (conclusion (?resist-scheme viscosity 20.0)))
```

Code List 7.9: Rule for choosing KTI 820 viscosity. If the desired resist thickness falls within the range of 0.95 μm and 1.14 μm, this rule succeeds and a viscosity value of 20.0 cst is chosen.

Mini-Expert Type	Best Applications
Backward Chain	Heuristic decisions; decisions based on experience rather than theory.
Simple Formulas	Decisions that involve ideal or near-ideal physical processes.
Graphs	Decisions that involve numeric interpolation of theoretical or experimental data; data that is numeric in nature but not accurately characterizable by a formula.
Symbolic Tables	Decisions that involve simple mappings among symbolic values. No interpolations are done.
Arbitrary Procedures	Only to be used if no simpler mini-expert can be used, this allows an arbitarily complex function to be called. Used when a design decision cannot be decomposed to fit the other mini-expert types.

Table 7.2: Summary of mini-expert types and what kinds of knowledge they might best express.

CHAPTER 8

Cameo's Data Base

Cameo's data base consists of five major elements. These elements and their overall structure were developed for the specific application to IC process design, although many of the principles used can be extended to other areas of CAD.

Examples and listings of contents of the data base are given throughout this chapter. It should be understood that Cameo is a constantly evolving system, and that the specific code fragments or listings shown in this chapter may not be what is contained in the most recent version of the system. They should be considered only as informative illustrations of the principles being discussed.

8.1 Elements of the Data Base

The five major elements of Cameo's data base are detailed in this section.

8.1.1 The Plans Data Base

The plans data base consists of two major parts, the plan skeleton and the working plans. First an overview of the rationale behind this structuring is presented, then the plan skeleton and working plans are described in detail.

Overview of Plans Data Base Structure

One of the characteristics of the design process in engineering is that many tentative decisions about the design are made and their ramifications studied before committing to those decisions. Therefore a CAD system that supports this type of exploratory design process may have to keep many potential designs in its data base. Further investigation of the design process shows that these designs are closely related to one another, so a method should be found that takes advantage of this property and organizes the evolving designs in a "natural" way.

When a point is reached where the designer would like to evaluate several alternative choices (such as deciding whether to use wet or dry etching), the designer mentally "splits" the current design he is working on into two, one using wet etching and the other using dry etching. The remainders of both designs are identical to each other and to the previous design. This is shown graphically in Figure 8.1.

This suggests the use of a frame system supporting hierarchical inheritance for storing the plans data base. When a design is split into several

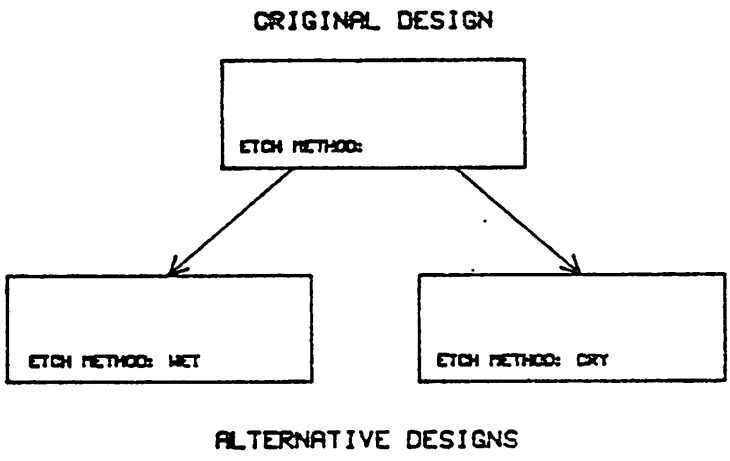

Figure 8.1: Splitting a design. The only difference between the two new designs is the etching method, dry for one and wet for the other. Both new designs inherit all structure and data from the original design, and just add the data for the etching method.

alternative designs, one for each potential answer to one design decision, a new plan is instantiated from the original plan for each potential answer. The frame system's hierarchical inheritance properties give each new plan the structure and data of its ancestors through inheritance.

The new plans can be modified independently of each other with no restrictions. If the designer decides to modify an inherited value, the data will be entered in the new plan and will "hide" the inherited value. thus leaving all other plans unaffected. An important part of this scheme is making sure that the user can only work on plans that have no children. Cameo is designed so that this is always true.

Cameo's *plan skeleton* is an empty plan (no values entered into it) which serves strictly as a structural template for the *working plans*, which form the hierarchical frame data base of evolving plans.

The Plan Skeleton

The plan skeleton is a set of frames that simply define one slot for each design decision. The skeleton also provides up to four pieces of data in each slot for very general purposes:

- Whether this slot is limited to holding only one value (answer) or multiple values.
- The prompt to use when asking the user for the value(s) of this slot.
- The units for a numeric value, such as `mm` for millimeters or `deg-C` for degrees Centigrade.
- The type of the value(s), such as `number` or `symbol`.

The working plans (described in detail in Section 8.1.1 on Page 114) inherit all the slots and the data defined in the plan skeleton.

The skeleton itself consists of six frames arranged as a two-level tree as shown in Figure 8.2. The top-level frame of type `process-plan` serves mainly to tie together the remaining frames of the plan skeleton. Pointers to two lower-level frames, of type `specs` and `wafer`, are installed in the top-level frame. The three remaining frames of the plan skeleton serve as

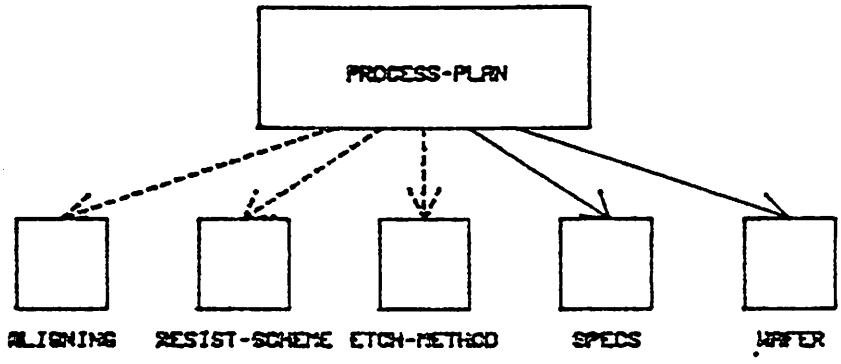

Figure 8.2: Structure of the plan skeleton. The top-level frame, a frame of type `process-plan`, serves to tie together the other five frames. Pointers to two of the remaining frames, `specs` and `wafer`, are installed in the plan skeleton. The three remaining frames serve as skeletons for entries in the three step libraries. Instances of these frames are not installed in the plan skeleton, but will be installed in a working plan when chosen from the library.

skeletons for actual step frames chosen later from the step libraries (described in Section 8.1.3). When the actual step frames are chosen, pointers to them are installed into the top-level frame of the current working plan (see Section 8.1.1 on Page 114).

An example of the top level frame of the plan skeleton is shown in Code List 8.1. For the sake of simplicity, only the most important slots of this frame have been shown. An excerpt of the plan skeleton frame for the resist scheme step is shown in Code List 8.2.

```
(deframe process-plan
  (ako ($value (process-frame)))
  (resist-scheme ($ask (single))
                 ($type (resist-scheme))
                 ($prompt ("Which resist scheme to use?"))
                 ($step-slot (t)))
  (aligning ($ask (single))
            ($type (aligning))
            ($prompt ("Which aligning scheme to use?"))
            ($step-slot (t)))
  (etch-method ($ask (single))
               ($type (etch-method))
               ($prompt ("Which etch method to use?"))
               ($step-slot (t)))
        % For the wafer and specs slots, install the value.
        % Only these have it because the others will be
        % chosen later from step libraries.
  (wafer ($ask (single))
         ($value (wafer))
         ($type (wafer))
         ($step-slot (t)))
  (specs ($ask (single))
         ($value (specs))
         ($type (specs))
         ($step-slot (t))))
```

Code List 8.1: Example of the plan skeleton's process-plan frame. The form for defining a frame begins with the (deframe frame-name line. Each slot being defined is a complete list, starting with the ako slot, which shows the parent from which this frame inherits, process-frame in this case. Only the data installed in the $value facet (subfield) of the slot is the actual value. Data stored in other facets, such as $ask and $prompt, hold ancillary information specific to the slot.

```
(deframe resist-scheme
  (ako ($value (process-step-frame)))
  (step-type ($value (resist-scheme)))
  (resist
      ($ask (single))
      ($type (symbol))
      ($prompt ("What type of resist to use?")))
  (surface-preparation
      ($ask (single))
      ($type (symbol))
      ($prompt ("Any special surface preparation?"))
      ($legal-values (acid) (piranha)))
  (hmds-needed
      ($ask (single))
      ($type (affirmative))
      ($prompt ("Is HMDS needed for this layer?")))
  (spin-speed
      ($ask (single))
      ($type (number))
      ($units (rpm))
      ($prompt ("What spin speed for resist application?")))
  (thickness
      ($ask (single))
      ($type (number))
      ($units (microns))
      ($prompt ("What resist thickness (in microns)?"))))
```

Code List 8.2: Excerpt from a plan skeleton's step frame. This is part of the `resist-scheme` step frame. The slots shown here, `resist`, `surface-preparation`, etc., each represent an individual design decision.

The Working Plans

The *working plans* form a hierarchical tree of instances of the plan skeleton. The depth and width of the tree of working plans is not restricted except by memory or addressing limitations of the hardware or operating system of the computer. Minimal memory is used for each plan, however, because each plan contains only data unique to it. All other data and structure is inherited from its ancestors.

Two events can split a plan into two or more new plans. Either the user requests Cameo to split a plan, or Cameo generates new plans because more than one possible answer to a design decision was found. The case for the user splitting a plan is simple: two new plans are generated that are identical and hold no data of their own. The user may then manipulate these two new plans independently.

The other case, where Cameo generates new plans because multiple answers were found, is of greater theoretical interest. During early research for this project, it was not clear what should be done when a mini-expert returns more than one possible answer to a problem. Should all answers be placed into the plan? Should all but the "best" one be rejected? If so, how could the system decide which was "best?" After some reflection, it became clear that all answers should definitely be retained, but that each answer, together with the rest of the current plan, formed a new, unique parallel plan.

The method to deal with this is very easy with a frame-based system, and is the approach taken here. The new plans share all structure and data with the current plan except for the single design decision that resulted in the multiple answers. Thus one new plan is generated for each answer, inheriting directly from the current plan, and one of the answers is installed in it. Figure 8.3 shows this graphically.

The only important caution that must be taken is that the plan from which the new ones were generated can no longer be accessible, since any change made to it will be reflected in all its children. Thus the user only has access to "leaf" plans, those that have no children. This is also true when the user tells Cameo to split a plan into two. The user no longer has access to the original plan.

8.1.2 The Description Frames

As described earlier (Section 6.8.3 on Page 74), the *description frames* are the "glue" of the system. The information stored in these frames gives the plan skeleton structure by specifying order or dependencies among the slots in a plan, assigns mini-experts to design decisions, controls how the evolving plans appear on the screen, and more.

In earlier versions of Cameo, the information now kept in the description frames was entered into the plan skeleton. Although initially this was a good approach, other important factors eventually ruled out this implementation. The most important issue was that HPRL's rules can only use the data stored in the `$value` facet of slots for pattern matching.

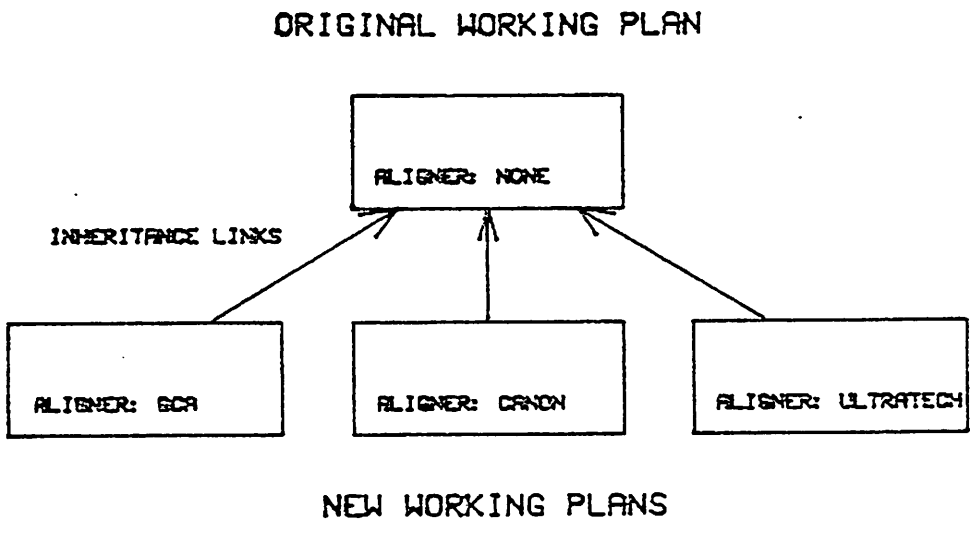

Figure 8.3: Splitting a working plan into multiple new plans because more than one possible answer was found. Each new working plan inherits all the data and structure of its parent, but adds its own unique answer of those found. In any plan split due to multiple answers, each new working plan will be unique.

When kept in the plan skeleton, this information was stored in the slot it was describing, but in facets other than $value. Thus no rules forming meta knowledge (see Section 7.2 on Page 95) could be written. Another appealing result of storing the structure information in separate frames is the increased modularity of the system.

The approach taken in Cameo for the system's description is to define

```
(deframe choose*resist-scheme*thickness-descr
  (ako ($value (choose-descr-frame)))
  (frame ($value (resist-scheme)))
  (slot ($value (thickness)))
  (view ($value (choose)))
  (mini-expert-type ($value (backward-chain-control)))
  (label ($value ("Resist Thickness")))
  (sort-key ($value ("b")))
  (show-next ($value ((resist-scheme viscosity))))
  (depends-on ($value ((wafer maximum-step))))
  (solvable ($value (t)))
  (explain-file ($value ("r-thick.text")))
  (mini-expert ($value (resist-thickness-domain))))
```

Code List 8.3: Example of a description frame. This description frame is for the `thickness` slot of the `resist-scheme` frame.

a single description frame for each slot of the plan skeleton that might ever represent a design decision shown on Cameo's screen. The name of each description frame is formed by concatenating the names of the frame and slot of the plan skeleton. An example of a description frame is shown in Code List 8.3, which shows the description frame for the `thickness` slot of the `resist-scheme` frame.

Following is a description of the slots shown in this description frame:

- `ako`

  The frame type of which this frame is a child. "AKO" stands for "a kind of."

- `frame`

  The frame type of which this description frame describes a slot.

- `slot`

  The slot name for which this is the description frame.

- `view`

  The view for which this is a description frame. A *view* is like an abstraction space. Some earlier work in this project identified the possible need for different views of the same data. Each slot can have one description frame for each view in which it appears, which can change the way the information in the slot is presented and how it relates with other slots. At present only the "choose" view is implemented, which is the view for "choosing" process steps. Other views might be for viewing plans at different levels of abstraction or detail, or evaluating the process step or executing it.

- `mini-expert-type`

  The type of mini-expert used for this design decision, chosen from among the four listed in Section 6.6 on Page 66.

- `label`

  The text string that represents this design decision on Cameo's screen.

- `sort-key`

  When more than one design decision appears at the same level under a heading, their order can be determined by entering a text string in this

slot. The sorting is alphabetic by sort key; if no sort keys are given, sorting is alphabetic by the `label` string.

- **`show-next`**

  When the design decision represented by this slot has an answer, it may be "expanded" by the user to show the next set of decisions that may be made. This slot lists which plan skeleton slots are the ones to appear. Since there are up to six frames in a working plan, the slots are listed as two-element lists, the first element being the frame type and the second the slot name.

- **`depends-on`**

  This slot lists, in the same format as the **`show-next`** slot, the slots on which the value for this slot depends. This slot can be used for backtracking when an answer is invalidated or overridden, although this capability is not yet implemented. Another application of this slot is to tell the user which design decisions he must make first before he can find an answer to this one.

- **`solvable`**

  Some items shown on Cameo's screen serve only as headings to organize information, and do not actually take values or have solution methods assigned to them. If this slot were one of these, the value in this slot would be `nil`. Otherwise it is `t`.

- **`explain-file`**

  The file name of the reference file associated with this design decision.

- `mini-expert`

  The actual mini-expert to use when deriving an answer for this design decision. This slot may be left empty if meta knowledge is entered into the system that allows the mini-expert to be found dynamically.

In addition, the following slots are defined for the use of specific mini-expert types. They hold information specific to each solution method.

- `solve-init-func` and `solve-finish-func`

  For backward-chaining, these are the names of functions that are run before and after the backward chaining occurs, respectively.

- `column` and `row`

  For matrix lookups, these two slots indicate the plan slots used as the column and row values respectively. Each holds a two-element list like the `show-next` slot.

- `x-variable` and `z-variable`

  For graph lookups, these two slots indicate the plan slot(s) used as the independent variables. Each holds a two-element list like the `show-next` slot. The `z-variable` slot is empty if a two-dimensional graph is being used.

8.1.3 The Step Libraries

One separate *step library* exists for each of the three major photolithography steps identified in Section 5.3 on Page 48. During early research for this

```
(deframe gca
  (ako ($value (aligning)))
  (description ($value ("GCA 4800 Wafer Stepper")))
  (template? ($value (t)))
  (machine ($value (gca)))
  (reduction ($value (10)))
  (resolution ($value (1.2)))
  (alignment-method ($value (fresnel-zone)))
  (type ($value (projection)))
  (alignment-error ($value (0.3)))
  (wavelength ($value (4360)))
  (field-size ($value (10)))
  (numerical-aperture ($value (2.8))))
```

Code List 8.4: Example of a library step frame. This frame is a member of the "aligning" library, and is for the GCA 4800 wafer stepper.

project, it was observed that most IC processes are developed by assembling existing process steps chosen from libraries (mental or otherwise) and then refining them to meet specific requirements. For this reason, the first design decisions made by Cameo's users are choosing standardized and documented steps from the three step libraries.

Each frame in a step library is a child of one of the three frames of the plan skeleton that are not yet installed in the top-level plan skeleton frame. The step frames in the library simply install values in the slots defined in the corresponding skeleton frame. An example of a step frame for the GCA 4800 wafer stepper is shown in Code List 8.4.

When Cameo or the user has chosen to use this step as the step for the "Aligning and Exposure" design decision, a pointer to an instance of the step is installed in the current working plan. A graphic representation of inheritance links at this point is shown in Figure 8.4.

When a step frame has been chosen from a step library for inclusion into the current working plan, the working plan consists of one more frame than before. If this plan is split, the new working plans will have corresponding constituent frames instantiated from those of the parent plan. This is shown graphically in Figure 8.5.

8.1.4 The Reference Files

One reference file is provided for every design decision ever shown on Cameo's screen. The reference file is supposed to answer the "Why?" questions about the decision, or supply enough information for the user to find out more. The user can view the reference file for any item on Cameo's screen by positioning the cursor on that item and typing the 'R' key for "References." The reference files are intended to be easily read and understood and their contents are kept to a single screenful if at all possible.

An example of a reference file for the "HMDS Needed?" decision is shown in Figure 8.6.

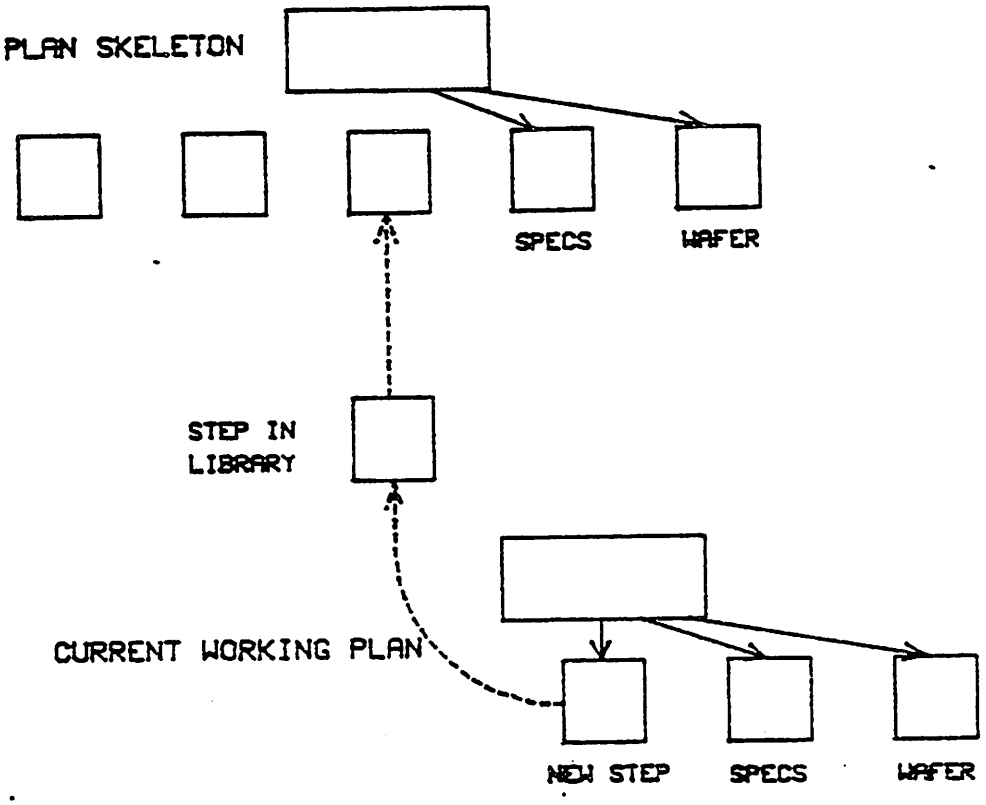

Figure 8.4: Links between frames after choosing a library step. Dotted lines are inheritance links, solid lines are actual pointers.

8.2 An Example

One of the possible design decisions involved in the example introduced in Section 1.3 on Page 5 is the determination of which standard viscosity of a given resist to use. This section traces the data base activity involved in a typical session where the user is deriving the viscosity. It is assumed that

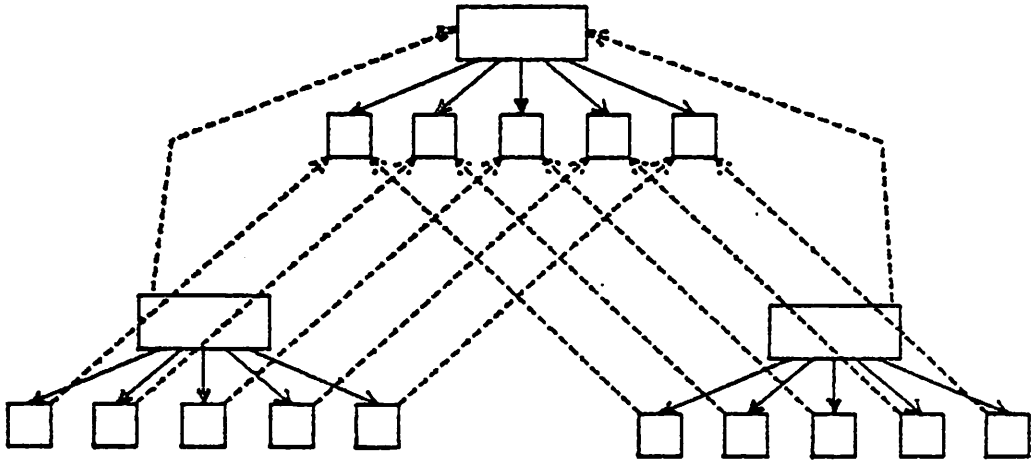

Figure 8.5: Detail of links among working plans after a two-way split. Dotted lines are inheritance links, solid lines are actual pointers.

the resist material has been chosen to be KTI Chemical's KTI 820 resist, the most popular positive resist used in UC Berkeley's Elecrtronics Research Laboratory processing line at this time.

After having chosen the standard KTI 820 step from the resist scheme library, the user will expand the "Resist Scheme" heading to uncover a few items, one of which is the "Resist" item (Chapter 9 describes the user interface in detail). The chosen resist step already has a value for this item (KTI 820) so it can also be expanded. One of the items under the "Resist" heading is "Thickness," which must be determined before viscosity is selected, so let us assume that the desired thickness has been chosen to be 1.2 μm. Now the "Thickness" heading can be expanded, and underneath it the "Viscosity" item appears.

STEP NAME: HMDS Needed?

PURPOSE: Determining whether HMDS application is needed.

MODIFIED: 01-May-85 21:10:56 (Mike Klein)

NOTES: HMDS (hexamethyl disilazane) is applied as a "primer" to improve adhesion of the resist to the wafer surface to minimize lift-off. HMDS application is desirable when oxide is present on the wafer, when feature sizes are small (under 5 microns), or when linewidth control is important.

HMDS is applied to the wafer after other preparation and before resist application. It is typically applied by vapor deposition or spin coating at low RPM (2000) for up to 30 seconds.

REFERENCES:
Shipley Company, "Microposit S1400 Series Photo Resist", technical brochure.

Tokyo Ohka America, Inc. (TOA), "OFPR-2 Photo Resist Series", technical brochure.

Figure 8.6: Example of a reference file.

The user positions the cursor on the "Viscosity" item. He may want to understand what the decision process for selecting a certain viscosity is, so he presses the 'R' key to see the reference file for this item. Cameo consults the description frame for this design decision, found by concatenating the view, frame, and slot names to generate

```
choose*resist-scheme*viscosity-descr
```

The `explain-file` slot of the description frame contains the text string "r-visc.text." Cameo reads this file into a buffer and displays it on the screen. The contents are shown in Figure 8.7.

If the user decides to let Cameo derive a viscosity value, he presses the 'D' key. This causes the following chain of events to occur.

First, Cameo checks the `mini-expert` slot of the description frame to see if a mini-expert has already been given for this design decision. Since the actual mini-expert will depend on the particular resist type (KTI 820 in this case), it will not be known yet. Thus the system must change the backward chaining domain to the `select-mini-expert-domain` and use backward chaining to find the specific mini-expert to use. This domain contains the rule shown in Code List 8.5, and so this backward chaining succeeds by installing the value `kti820-viscosity-domain` as the mini-expert for this particular design decision.

Once Cameo has found the mini-expert for this design decision, it can use that mini-expert to attempt to derive an answer. The rules in the

```
STEP NAME: Viscosity of Resist

PURPOSE: enter the viscosity of the resist to use.

MODIFIED: 29-Apr-85 09:57:24 (Mike Klein)

NOTES: Most available resists come in standard dilutions
for obtaining different viscosity values.  A high viscosity
resist will result in a thicker film.

This step determines which of the standard viscosities for
this type of resist will be a good candidate for your
lithography.  In general, best results (step coverage)
will be obtained by using the highest viscosity.

REFERENCES: Technical brochures for each resist type.
```

Figure 8.7: Reference file for viscosity design decision.

```
(rule kti-820-viscosity-rule select-mini-expert-rule
  (premise
    (and (*system* current-step-being-solved-for
                   (resist-scheme viscosity))
         (?resist-scheme resist kti820)))
  (conclusion
    (?resist-scheme mini-expert kti820-viscosity-domain)))
```

Code List 8.5: Rule for finding mini-expert for selecting KTI 820 resist viscosity.

`kti820-viscosity-domain` take the form of the rule shown in Code List 7.9 on Page 106. One rule's thickness range includes the desired value of 1.2 μm. so it succeeds and installs the viscosity value of 27.0 cst in the `viscosity` slot of this `resist-scheme` frame of the current working plan.

Since only one answer to this design decision was found, no plan splitting was done. If, however, more than one answer had been found, such as the viscosities of 20.0 and 27.0 cst, two new working plans would have been generated, inheriting all data from the current working plan. The current working plan would have been made inaccessible to the user, and both new plans made accessible. The 20.0 value would be installed in one new plan's `viscosity` slot, while the 27.0 value would have been installed in the same slot of the other new plan.

8.3 Chapter Summary

Cameo's data base is structured primarily for supporting the design of IC processes. It is highly modular. consisting of independent parts that are tied together by "description frames." Since much of the structure of the system is kept as data in frames, HPRL rules can be written to understand and modify the system's structure.

The evolving design plans are stored as a hierarchical tree. taking advantage of the inheritance capabilities of the frame system. Each design plan in the plans data base is unique but holds only the data unique to it, inheriting the rest. This plans data base schema is a "natural" representation of

the structure that an expert IC process designer gives to evolving designs. It also allows many design plans to be stored with efficient use of memory.

From observing experienced IC process designers, it was seen that most process designs are modifications of assemblies of previously proven steps. Thus the data base also contains libraries of documented major photolithography steps. The user may browse through the libraries and choose a step or have Cameo choose one for him.

Each possible design decision has a "reference file" associated with it. Reference files are most useful for general information on the design decision, and also as pointers to more detailed references. They are always short, less than one screenful of information if at all possible. The reference files answer the "Why?" questions of novice designers.

It appears that the combination of these data base elements makes Cameo's data base unique. Many recent software systems have used the approach of separately defining the system's structure and description from the code that implements it. However. very few software systems use the remaining structuring principles used in Cameo's data base. In particular, the concept of reference files for every design decision does not appear to be implemented in any other CAD system. These reference files were found to be one of Cameo's most important contributions.

CHAPTER 9

Cameo's User Interface

Any computer system's user interface is an extremely important but often overlooked part of the system. With recent advances in price and performance of computers more emphasis on the user interface, which is often computationally expensive, is feasible. A good user interface makes the system more accessible to a wider range of users.

The remaining parts of Cameo's software structure are described in this chapter. In reference to the description given in Section 6.4 on Page 64, these remaining parts are the *display manager* and the *program control manager*. Since these two are heavily dependent on one another, they will be discussed together under the general heading of user interface.

A three-step method is proposed that was used to arrive at Cameo's user interface design. This method is detailed in Section 9.2 and its implementation in Section 9.4.

9.1 Importance of the User Interface

As the costs of computer resources steadily decrease, these resources can be made available to more potential users. Users who are not trained in the conventional line-oriented computer system interface must be considered when designing new computer system applications. Even Cameo's intended users are not always well versed in the details of interacting with a typical program.

On the other hand, the user interface cannot be so simplified that it impedes the use of a system aimed at a complex technical task. The user interface must balance these two conflicting requirements. The ideal user interface allows new users to learn how to use the system rather quickly, but allows experienced users to use it without impeding them unnecessarily.

For the kind of application that Cameo addresses, the user interface is of undeniably high importance. Jack D. Grimes of Intel Corporation, in a Guest Editor's Introduction to a series of articles on human factors, writes, "It is about time (several [contributers] have said) that users be given their proper place as the key 'factor' in the system" [Grim84]. A poor user interface can make the system virtually useless, either by making the system so difficult to use that the potential time saved in using it is nullified by the time spent in learning and using it, or by so simplifying the problem the system addresses that it can only help perform trivial tasks.

9.2 User Interface Design Considerations

Several important considerations to designing a user interface for a particular application must be addressed. These considerations revolve around making the computer "transparent," giving the user the feeling that he is manipulating the objects the display represents.

The three main steps used to design Cameo's user interface are:

1. Find the users' "conceptual operations"
2. Find a data presentation comfortable to the users
3. Implement the user interface as conveniently as possible on available hardware and software.

The first two steps are described in detail in the following two subsections, and the third is described in Section 9.4 on Page 137. Other considerations important to the implementation, having to do with psychological factors, are reviewed in Section 9.2.3.

9.2.1 Conceptual Operations

Beginning the design of a user interface involves identifying the "conceptual operations" that potential users would perform. In learning a new computer system, the user is faced with something like learning a foreign language. The "language" of this user interface should be "*efficient* and *complete* and should have a *natural grammar*" ([Fole82] p. 219). This topic has been addressed for

graphics systems, identifying a number of generic operations that graphics systems users perform [Fole84].

For this application, a CAD system for IC process design, these conceptual operations were identified by interviewing many process designers about how they go about designing an IC process. Their methods included enough common ground to identify the following conceptual operations:

- *Provide a value*—the designer provides a value for a design decision.
- *Replace a value*—the designer changes a value for a design decision.
- *Split and Merge*—The designer would like to try several alternative designs. The first step is to split a design into several which can be treated independently of each other. Afterwards, the designer may merge the designs back together so that they are treated identically.
- *Find further information*—The designer will often have a question about a certain aspect of the design or wish to refer to other resources. This information should be available at any time in the design process.
- *Verify*—the designer verifies the state of the current design.
- *Throw away*—the current design will no longer be considered.
- *Start fresh*—a new, fresh, design is needed.
- *Show more or less detail*—Too much detail clutters the designer's mind and the system's display. The user interface needs to support filtering

out unwanted levels of detail in the design but retain detail where it is needed.

Additional conceptual operations certainly exist, but were not observed with sufficient consistency to establish a few fundamental operations to represent them. These remaining conceptual operations have mostly to do with "viewing" the design in different ways depending on the current design activity (see Section 8.1.2 on Page 117 for a short discussion of how views might be implemented). For instance, a simulator requires a different way of representing the design than the designer does. The designer sometimes would like to check the representation given to the simulator directly. This is an area that deserves further work.

9.2.2 Data Presentation

Once the conceptual operations have been identified, the next step is to find a way to represent the evolving designs to the user naturally and allow the user to manipulate them naturally with the conceptual operations. The rule for representing designs effectively appears to be to present them in a way that the user easily recognizes and expects to see. This will typically use much more computer power and display bandwidth than conventional program interfaces, in part because the interface is optimized for efficiency on the user's part, not the computer's.

For IC process design, displaying the evolving designs naturally appears to be by modeling the display after a process run sheet, the sequence of processing steps that a wafer goes through. Other representations considered

were graphics-based, but no natural way of using graphics to display IC process steps could be identified. As a result, Cameo's display is line-oriented, each line showing one design decision.

The lines are grouped and indented to form an outline. When the design session begins, only the highest-level outline headings are shown. The user may expand each heading, finding the next level of detail underneath it. The items and their grouping are chosen so that a user can proceed through one related and relatively independent set of design decisions by visiting the items lying underneath a single heading.

9.2.3 Psychological Issues

The user interface must be highly interactive and allow the user to feel in control. The following passage is from [Shne80], p. 227:

> Nothing can contribute more to satisfactory system performance than the conviction on the part of the terminal operators that they are in control of the system and not the system in control of them. Equally, nothing can be more damaging to satisfactory system operation, regardless of how well all other aspects of the implementation have been handled, than the operator's conviction that the terminal and thus the system are in control, have "a mind of their own," or are tugging against rather than observing the operator's wishes.

Another very important factor is consistency of the interface. Again referring to [Shne80], people have very limited "short-term memory," the part of human memory used to store information used over a short time span (30 seconds is typical). An inconsistent user interface forces the user consciously to place the interaction techniques in long-term memory, a far more difficult

task than leaving them in short-term memory, and requires the user to recall the appropriate set of interaction techniques for each situation. This leads to frustration and exhaustion. If the user interface is kept consistent and simple, the user can retain most of the interaction techniques in short-term memory.

The grouping method described in Section 9.2.2 helps produce the feeling of "closure" [Shne80] by allowing the user to concentrate on a single set of related design decisions and make those decisions relatively quickly. The user proceeds to the next set of decisions with the feeling that the previous ones are finished. This also reduces the amount of information the user needs to keep in short-term memory.

The combination of a highly interactive system with operations that the *user* is comfortable with and structuring design decisions so the user can quickly attain closure should come close to guaranteeing that the user will feel in control of the system.

9.3 User Interface Description

Cameo's interface technique, based on the line-oriented outline model, was found to be a natural way of displaying the evolving design. The user can position the system's display cursor on any line item of the outline, and select a command from a short menu displayed at the bottom of the screen. This serves both to allow the user to make his selection unambiguously, and to inform the system of the context of the user's next actions. Most of the

commands operate on the currently selected outline item. The commands are virtually exact analogs of the conceptual operations listed earlier in this section, with the exception of a few not yet implemented, and the addition of the "Derive" command.

For almost all operations the user performs, this is the only mode of interaction with Cameo. Any of the commands listed in the menu can be used on any of the outline items shown on the screen. The total number of commands, ten, are slightly more than the seven that might be ideal [Mill56], but this has not been found to be a problem. The commands correspond closely with the conceptual operations the user wishes to perform, so most of Cameo's commands do not need to be learned as entirely new commands. The names of the commands were carefully chosen so novice users can easily recognize what they do just by seeing their names. See Table 6.1 on Page 67 for a table listing these commands. A menu at the bottom of screen lists the commands grouped by type of operation, such as those that find design decision answers, modify the display, or do global operations, so the user can easily refresh his short-term memory if needed.

A complete example of the screen display at a certain stage of interaction with Cameo is shown in Figure 9.1.

9.4 Implementation of the User Interface

Once the design of the user interface was decided, actual implementation was relatively straightforward. The version of Portable Standard Lisp (PSL)

```
    CAMEO PHOTOLITHOGRAPHY DESIGN

PLAN: Initial Plan
    --PROCESS REQUIREMENTS--
        Alignment Accuracy (3 sigma): 1 micron
        Field Size: 5 mm
        Minimum Feature Size: 2 microns
        Optimizing for: DIMENSION-CONTROL
    --WAFER STATE--   [more...]
    Aligning and Exposure: GCA-1
        Machine: GCA
            Type: PROJECTION
            Field Size: 10 mm
            Resolution: 1.2 microns
            Alignment Error (3-sigma): 0.3 microns
            Reduction Ratio: 10
            Exposure Wavelength: 4360 angstroms
            Numerical Aperture: 0.28
        --Contrast and Linewidth Variation Calculations--   [more...]
        Align To
    Resist Scheme: KTI820-1
        --Preparation--   [more...]
        Resist: KTI820
            Resist Thickness: 1.04 microns
                Viscosity: 20.0 cst
                    Spin Speed: 4928.47143 RPM
            Refractive Index: 1.62+0.02j   [more...]
    Etch Method: DRY-1   [more...]


--------------------------------------------------------------------------
? Refs [Derive/Provide/Override] [Expand/Collapse] [Split] [New/Kill-plan]
```

Figure 9.1: Complete screen example. The user has gone through the design decisions for `Resist Thickness`, `Viscosity`, and `Spin Speed` derivation, and has completed this set of decisions, attaining "closure" (see text). Headings ending in -- serve as placeholders only and do not actually represent design decisions. When a design decision is given a value, it appears on the display with a colon, the value, and the units if appropriate, as in the `Viscosity` item. Design decisions with no values yet are shown as in the `Align to` item.

available for the Hewlett-Packard 9836 computer has browser extensions built into the NMODE editor (see Section 6.5 on Page 66). These browsers are based on a line-oriented display buffer where the user can position the cursor on a line and specify a command to be executed for that line. Of the many different display and user interface options available for this system (including graphics), this was the closest to the desired interaction style and was therefore chosen.

NMODE's browsers are implemented with object-oriented programming. A browser is an object that maintains a list of items (browser items), and supports messages that affect both the browser itself (primarily its display) and the browser items belonging to it. The main function of the browser object is to interpret keystrokes typed by the user and send the appropriate message to the browser item indicated by the cursor position when the user types a key.

A browser item can be any object that responds to the messages that the browser object might send it. Most of these have to do with keeping the display updated. Others include killing (deleting) the browser item or temporarily removing it from view.

The parts of Cameo's code that handle these basic capabilities form the *display manager*. The objects implementing the actual browser items are more complex than just this because they also respond to messages that are logically part of the *program control manager*. The actual code is kept separate by using PSL's object inheritance of code and variables. Figure 9.2

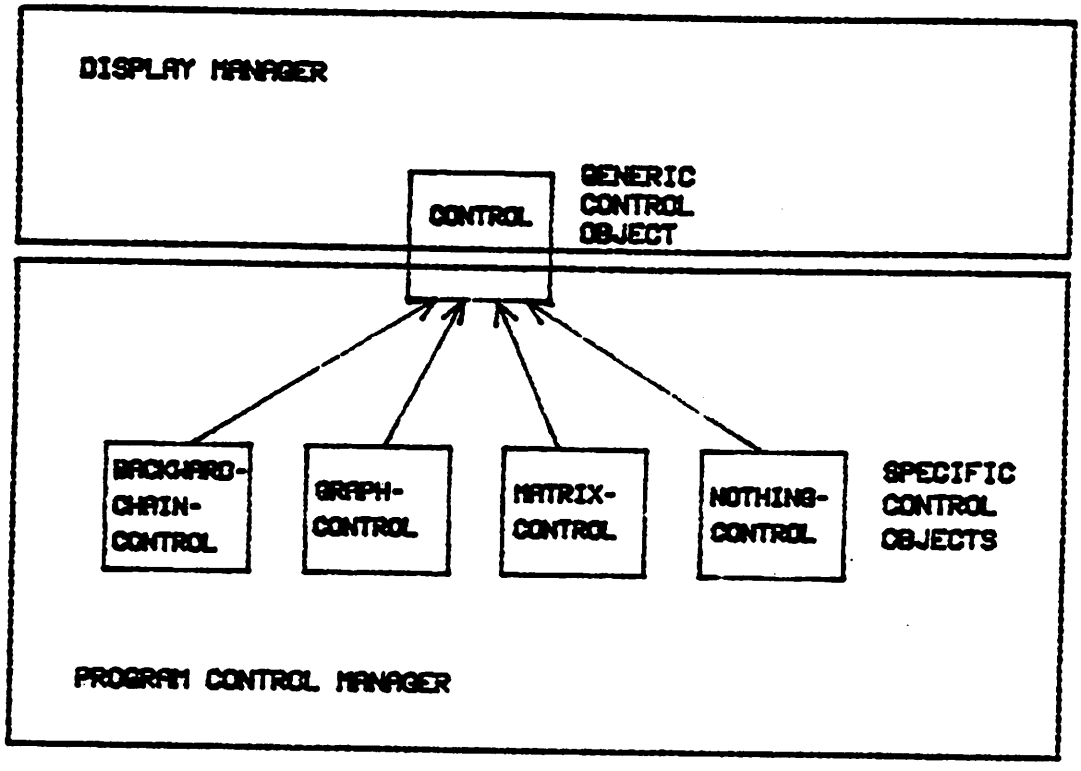

Figure 9.2: Structure of browser item objects. The generic object, `control`, defines methods common to all solution methods, including those for messages required of any browser objects. The specific objects define methods specific to each solution method. All items appearing in the browser are instances of the specific object definitions.

shows how inheritance is used to separate the code and definitions of the display manager from that of the program control manager. Table 9.1 lists most of the actual messages supported by the objects, and shows where they are implemented.

Messages Logically Belonging in the Display Manager:

Message	Defined in	Description
`display-text`	Generic	Returns the text to be shown on screen.
`expand`	Generic	Finds items underneath this one, informs browser object they are to be shown.
`collapse`	Generic	Sends the `kill` message to itself.
`kill`	Generic	Removes itself from the browser display and sends the `kill` message to all items underneath it.

Messages Logically Belonging in the Program Control Manager:

Message	Defined in	Description
`explain`	Generic	Shows reference file for item.
`solve`	Generic	Finds the mini-expert for this item if not known, sends the `solve-specific` message to itself, takes care of splitting plans if more than one potential answer returned.
`user-input`	Generic	Prompts user for manual input. Also takes care of splitting plans if more than one answer.
`replace`	Generic	Replaces existing answer, else same as `user-input` method.
`solve-specific`	Specific	The only method that needs to be written to add a new solution method. This method activates the mini-expert for this problem to attempt to solve it.

Table 9.1: The "most interesting" browser object messages. More are implemented but these are the most illustrative.

9.5 Example

To illustrate the operation of Cameo's user interface, the example introduced in Section 1.3 on Page 5 will be used. Say that the user is solving the set of design decisions referred to in Figure 9.1 on Page 138, and is specifically working with the item representing the design decision for finding the resist viscosity to use.

We assume that the user will perform the following actions: first he will expand the previous heading (`Resist Thickness`), then ask for references on the viscosity, then have Cameo derive an answer for it. The following three sections detail the sequence of events that will occur.

9.5.1 Expand Previous Heading

The user must have already found an answer for the `Resist Thickness` item for that item to be expanded. The user receives an informative error message if not. To expand the item, the user positions the cursor on the item and presses the 'E' key. The browser sends the `expand` message to the object that this item represents. This message is defined in the generic `control` object, so the actual object receiving the message must use inheritance to find the method for this message.

When the method is located, it is executed and goes through the actions described in Section 8.2 on Page 123 to find the description frame for this item. In the `show-next` slot of the description frame are listed the next

item(s) that should be displayed. One new browser object is instantiated for each one, and the new objects are added to the browser. The browser takes care of sorting the new items so they appear under the current item.

9.5.2 Show References on the Item

The user positions the cursor on the item for the viscosity design decision and presses the 'R' key. Cameo's browser sends the `explain` message to the object that this item represents. This message is defined in the generic `control` object, so the actual object receiving the message must use inheritance to find the method for this message. When the method is located, it is executed and goes through the process described in Section 8.2 on Page 123 to find the reference file associated with this design decision. The file is displayed to the user. The only commands available to the user while viewing this file are scrolling commands and the `Quit` command. When finished reading the reference file, the user types 'Q' to return to the browser.

9.5.3 Derive an Answer

With the cursor positioned on the same `Viscosity` item, the user presses the 'D' key. The browser then sends the `solve` message to the object that this item represents. This message is defined in the generic `control` object, so the actual object receiving the message must use inheritance to find the method for this message.

When the method begins executing, it first checks if the mini-expert is known. In this case, it is not, because a different mini-expert is needed to select the viscosity for each brand of resist. Thus the method goes through the actions described in Section 8.2 on Page 123 to find the required mini-expert. The next step taken is to send the current object the `solve-specific` message. This message is defined by each specific object type, and so each solution method, or mini-expert type, has its own way of responding to this message. The only requirement imposed by the rest of the system is that the value or values found as answers be left in the current working plan's frame and slot for this design decision.

In this case, backward chaining is used to find the answer. The mini-expert is the rule domain to be used. As described in Section 7.3, the backward chaining domain is changed to the mini-expert and HPRL's inference engine is called to find any answers.

After completion of the backward chaining, control returns to the `solve` method inherited from the generic `control` object. The remainder of the method checks the answers left in the current slot. If no answers were found, that is reported and no data base updates are made. If one answer was found, that is entered into the current working plan and the display is updated. For instance, if a viscosity value of 27.0 cst had been identified as being the best choice, the browser item would have changed from

`Viscosity`

to

`Viscosity: 27.0 cst [more...]` The `[more...]` indicates that this heading can be expanded further, in this case to expose the **Spin Speed** decision.

If more than one answer had been found,[1] the user would be presented with the derived answers and be asked to choose one or more of the answers (or none), depending on whether or not the plan skeleton specified that only one answer was allowed. The `solve` method would then proceed through the same actions as above, leaving all as is if none were chosen, or installing the answer if just one was chosen.

If multiple answers were chosen by the user, and the plan skeleton only allows one answer, the `solve` method instantiates a new plan from the current one for each answer and installs one of the answers in each one. It then removes the current plan from the display by sending the object for the main heading of the plan a `kill` message, and builds a new set of browser items for each newly generated plan. The display is updated with these changes.

Thus it is guaranteed that only "leaf" plans (those with no children) can be operated on by the user. No other plans can be seen on the screen, and since a plan can only be operated on if the cursor can be positioned over one of its items, it is inaccessible for further modification.

---

[1] Multiple answers are not possible for this particular decision because of the way the rules are structured. Most design decisions will have more than one potential answer.

9.6 Chapter Summary

With the growing numbers of potential users of a computer system, user interfaces are growing in importance. The quality of a system's user interface can be the most important element of the system's success or failure. However, a good user interface is usually computationally expensive, and so may not be feasible for some applications.

A three-step method is proposed that was used to arrive at Cameo's user interface design. First, the *conceptual operations* that users perform were first identified. Then a data presentation scheme that the *users* find natural for the application was found. Lastly, an implementation using available hardware and software was designed. Other application-independent psychological issues need to be considered when implementing the user interface.

Cameo's user interface is modeled after an outline of an IC process, similar to filling out a "run sheet." The user only needs to remember a few commands which apply to any object shown on the screen in a consistent fashion.

CHAPTER 10

The Application

In order to evaluate Cameo's underlying framework, a specific application was chosen for implementation. Of three proposed applications, the one chosen was to make Cameo a general-purpose expert for image transfer from the mask to the resist for a few selected layers of the IC process.

Adapting Cameo from a previous simple demonstration system to this real application involved a major restructuring and the implementation of many new mini-experts. Aside from the time spent learning about the application, the actual coding took only a few days.

Cameo was tied to the SAMPLE process simulator for verification of the plans. Simulation results appear to bear out Cameo's initial estimates of linewidth variation.

10.1 Choosing a Specific Application

Three types of applications for Cameo were proposed and considered.[1] The main requirements of the application are that it be practical to implement, be useful once implemented, and provide some way of evaluating the system's usefulness. These applications were proposed:

1. Tailor Cameo to solving a specific, unique photolithography design problem.

2. Find a "standard" photolithography benchmark problem with whose solution experts are familiar, and evaluate the improvement in productivity of novice IC process designers using Cameo.

3. Make Cameo an expert on a few important problems of general photolithography design.

An example of a unique problem for the first proposed application is the fabrication of integrated sensors. Experts familiar with the problem would be able to give qualitative estimates of the system's usefulness. An advantage of this approach is that the system would address a real problem and would be immediately useful to at least a small group of users. Its application would also be bounded, and thus hopefully practical to implement. A disadvantage is that the system would be useful only to a relatively small group of users.

[1] These applications grew out of discussions with Professors F. Balderston, UC Berkeley Graduate Business School, R. Katz, UC Berkeley Computer Science Division, and W. Oldham, UC Berkeley Electronics Research Laboratory.

A benchmark photolithography design problem, in order to be useful in a controlled evaluation environment, would most probably have to be artificially crafted. The main advantage of aiming Cameo at this application is that it comes close to an ideal controlled experiment. This application's major disadvantage is that once developed to help with this particular IC process design problem, Cameo may not be useful for most real problems. Another important disadvantage is that potential users will not be self-motivated to use the system, since they will feel it is being used to test them, not to help them.

The last application would make Cameo useful for a large variety of typical design problems, and thus useful for a large group of users, but does not provide the more controlled tests of the previous two applications. Evaluation of the system's usefulness would probably best be accomplished by asking each user to fill out a short questionnaire after using Cameo.

After considering these applications and the original application requirements, Cameo's application was chosen to be the last proposal above. Even though evaluating its contribution is more difficult, Cameo would make a more immediate contribution to IC process design. Evaluation would be difficult, relying on users' comments, but would be under the conditions of real IC process design problems.

Once the type of application was chosen, it remained to find the specific application. One of the major problems in shrinking device geometries is controlling the pattern dimensions. Limiting dimension variation becomes

very important as the dimensions themselves shrink. A typical linewidth variation in an IC process where linewidths are 2 μm could be as high as 0.25 μm on a polysilicon layer or 0.5 μm on a metal layer. In some cases, such as analog circuits that require extremely close component matching, such a linewidth variation is unacceptable.

Linewidth variation is affected by, among other factors, the distance of the resist from the exposing machine's focal plane (due to a focusing error or vertical features already on the wafer) and the types and thicknesses of the layers underneath the resist. Linewidth variation is an inescapable fact of optical lithography. The challenge is to keep it within allowable limits.

10.2 Understanding the Application

The first steps in applying Cameo to linewidth control are understanding the causes of linewidth variation, finding the methods used by experts to estimate linewidth variation, and finding the heuristics used by experts to suggest remedies if the linewidth variation is too high.[2] These topics are discussed in more detail in the following sections.

10.2.1 The Causes of Linewidth Variation

The single most important cause of linewidth variation in transferring an image from mask to resist is non-uniformity of the exposing radiation energy

[2]Special thanks are in order to Professor A. Neureuther of UC Berkeley's Electronics Research Laboratory for the time and effort he spent putting his knowledge into the context needed for developing this application.

in the photoresist. A perfect exposure operation will couple all available exposure energy into the resist with the boundaries from light to dark areas of the mask preserved. In reality this never happens, for a number of reasons which can be classified into the following groups:

- *Optical limitations*—a given combination of optics and exposure radiation wavelength imposes a fundamental resolution limit.
- *Reflective interference*—some exposing radiation will be reflected at all boundaries where refractive indices do not match. The reflected radiation will interfere with the incoming radiation and cause standing waves inside the resist. In severe cases (which happen often), the resist at standing wave minima will be underexposed.
- *Defocus*—degradation of the projected image due to focusing error, a tilted wafer, and large vertical features on the wafer, all of which serve to displace the desired exposure plane from the optics' focal plane.

Other causes of final linewidth variation occur during the image transfer from resist to wafer, the etching step. Cameo does not currently treat the etching step in detail.

10.2.2 Estimating Linewidth Variation

A. Neureuther and W. Oldham have developed some useful heuristics for estimating linewidth variation [Neur81,Oldh81]. The methods are based on first finding values for horizontal contrast (image contrast) and vertical contrast

due to interference from reflections. A high horizontal contrast means that the image projected into the resist has a large intensity variation between lightest and darkest areas. A high vertical contrast means that substantial interference is occurring in the resist; the vertical contrast reflects the standing wave ratio of the exposing radiation in the resist.

The exposure radiation at any point in the resist can be found by a superposition of these two effects. For best linewidth control, the horizontal contrast should be high and the vertical contrast should be low. The heuristic derived by Neureuther and Oldham says that linewidth variation becomes essentially uncontrollable when the vertical contrast exceeds the horizontal contrast.

Thus one of the first things a user should do with Cameo is to derive values for vertical and horizontal contrast, compare them, and make a decision on whether or not the plan he is building will have a good chance at meeting his requirements.

Since some of the causes of linewidth variation depend on the specific optics and exposure wavelength used, the user will first have to choose (or let Cameo choose) an aligner step from the aligner step library. The GCA 4800 Wafer Stepper is used in an estimated 95% of the processing in UC Berkeley's Electronics Research Laboratory's IC processing line, so this is currently the only choice in Cameo's aligner step library. Once the user has specified his process requirements and the current wafer state and an aligner has been chosen, the vertical and horizontal contrasts can be found.

The sequence of decisions made by an expert (Professor Neureuther) is as follows:

1. Find the vertical contrast. This is found using a formula, from refractive indices, exposure wavelength, and the resist's absorption.

2. Express the maximum defocus distance in terms of fundamental units. The maximum defocus distance is the sum of the maximum vertical features on the wafer, wafer tilt, and the focusing error of the aligner itself. This distance can be expressed in terms of the optics' Rayleigh limit,[3] an indication of the optics' depth of field.

3. Express the desired minimum linewidth in terms of the optics' fundamental resolution limit.[4]

4. Use the normalized defocus distance and normalized spatial frequency to look up the horizontal contrast in an experimentally derived graph [Oldh81].

5. Use the ratio of horizontal to vertical contrast to estimate actual linewidth variation. This is also a lookup into an experimentally derived graph [Oldh81].

---

[3] The Rayleigh limit is $\frac{\lambda}{2(NA)^2}$ where λ is the wavelength of the exposing radiation and NA is the optics' numerical aperture.

[4] This fundamental resolution limit is $\frac{\lambda}{NA}$.

6. Based on the estimated linewidth variation and the user's requirements, make a heuristic decision on whether or not the requirements can be met with this plan. If so, suggest further courses of action that may need to be taken to guarantee acceptable linewidth variation. This might include using special coatings on the wafer before resist is applied to enhance horizontal contrast, or using a special type of resist.

10.3 Implementing the Application

This section is divided into three subsections. The first examines how the decision sequence for evaluating linewidth control was implemented. The second shows the implementation of selected mini-experts. The third subsection looks at verification of a complete plan with the SAMPLE process simulator.

10.3.1 Implementing the Decision Sequence

The sequence of decisions outlined in Section 10.2.2 is straightforwardly implemented under Cameo's framework. Whenever one decision depends on another, the dependent decision will appear on Cameo's screen underneath and indented from the decision it depends on. Thus the dependent decision is not accessible to the user until a value has been found for the decision it depends on.

The fragment of Cameo's fully-expanded screen that shows these decisions is shown in Figure 10.1. The initial work in implementing a decision

```
------------------------------------------------------------------------

      CAMEO PHOTOLITHOGRAPHY DESIGN

Plan: Initial Plan
    Aligning and Exposure: GCA-3
        --Contrast and Linewidth Variation Calculations--
            Vertical Contrast: 87 percent
            Normalized Defocus Distance: 1.47
                Normalized Spatial Frequency: .3871
                   Horizontal Contrast: 93 percent
                      Estimated Linewidth Variation: 0.217 microns
                      Recommended Resist Treatment: ANTI-REFLECTION-COATING
                      Recommended Etch Treatment

------------------------------------------------------------------------
? Refs [Derive/Provide/Override] [Expand/Collapse] [Split] [New/Kill-plan]
```

Figure 10.1: Screen fragment showing the decisions involved in estimating linewidth variation.

sequence is the most difficult. This is finding the actual sequence of decisions an expert uses and structuring them so that they represent easily-understood concepts. If the values are numeric, hopefully they can be expressed in normalized unitless form like "Normalized Defocus Distance" and "Normalized Spatial Frequency"[5] shown in Figure 10.1. Normalized unitless quantities are much easier to work with since a single graph or formula can be used for more than one special case.

[5] "Normalized Spatial Frequency" is the inverse of the ratio of line pitch to the exposing machine's fundamental resolution limit. Its inverse would probably be called "Normalized Minimum Line Pitch." Convention dictates this quantity be in units of spatial frequency.

Once the sequence of decisions has been identified, incorporating them into Cameo's framework is quite easy, consisting of four steps for each design decision:

1. Add a slot for the decision to the plan skeleton.

2. Make a description frame for the new slot. A special Lisp command has been written that generates a template description frame automatically.

3. Write a reference file that describes the *how* and *why* of this decision. A special Lisp command has also been written that generates a template reference file.

4. Build the mini-expert for this decision.

For most decisions, the most difficult or time-consuming step is building the mini-expert. For some decisions where the mini-expert is a simple rule or formula, the most difficult step actually turns out to be writing a descriptive reference file. Since the reference files are often as important as the mini-experts in Cameo's knowledge base, this is not surprising.

When work began on applying Cameo specifically to linewidth control, it had previously been structured for giving simple demonstrations. A nearly complete restructuring was necessary. One measure of the success of Cameo's framework was that it took approximately 15 to 20 hours of programming time to convert Cameo from a simple demonstration system to a system that

provides a useful IC process CAD service. This does not include the time spent initially learning about the causes of linewidth variation and the initial decision structuring, which will vary a great deal from one set of decisions to another.

A typical user can progress through the decisions shown in Figure 10.1 on Page 155 in under five minutes, including estimating linewidth variation and finding any suggestions for further special treatment of either the resist scheme or the etch method. Thus Cameo enables the user to explore a number of alternatives in a matter of minutes. This should be compared to the typical observed case where the IC process designer can only explore one or a very few alternatives, assuming he has the resources and knowledge available to make accurate evaluations of these alternatives.

10.3.2 Implementing the Mini-Experts

Implementing the mini-experts is usually the most difficult and time-consuming task once a decision sequence has been found. Some generic examples of mini-experts are given in Chapter 7. This section details the implementation of two mini-experts, one that relies on a complex procedure and one that uses a graph lookup. Relevant examples of heuristic mini-experts are given in Section 7.1.1 on Page 84.

Formulas and Procedures

As explained in Section 7.1.4 on Page 94, mini-experts that use formulas or procedure calls are implemented using HPRL's rule facility for collecting ar-

guments and performing type and range checking, and calling a Lisp function and "splicing in" its returned value when the rule concludes. The example shown here is the calculation of the vertical contrast.

The mini-expert for this decision is shown in Code List 10.1. The vertical contrast depends on the intensity of the light reflected at the boundary between the resist and the top layer of the wafer. The maximum intensity is where there is full constructive interference and the minimum intensity is where there is full destructive interference, or

$$I_{max} = (1 + R)^2$$
$$I_{min} = (1 - R)^2$$

where R is the relative amplitude of the reflected radiation. The vertical contrast is expressed as

$$C_v = \frac{I_{max} - I_{min}}{I_{max} + I_{min}} = \frac{2R}{1 + R^2}$$

R depends on two factors, the reflection at the boundary and attenuation in the resist due to absorption. The reflection coefficient A_r depends on the refractive indices

$$A_r = \frac{n_1 - n_2}{n_1 + n_2}$$

In the general case, refractive indices are complex numbers, so A_r can be expressed as

$$A_r = A_{r0} e^{i\theta}$$

where

$$A_{r0} = \left| \frac{n_1 - n_2}{n_1 + n_2} \right|$$

$$\theta = \tan^{-1} \frac{\mathrm{Im}(n_1 - n_2)}{\mathrm{Re}(n_1 - n_2)} - \tan^{-1} \frac{\mathrm{Im}(n_1 + n_2)}{\mathrm{Re}(n_1 + n_2)}$$

```
(rule-domain vertical-contrast-estimate-domain backward-chain)
(fassert vertical-contrast-estimate-rule
         (ako ($value (backward-chain-rule)))
         (domain ($value (vertical-contrast-estimate-domain))))

(rule estimate-vertical-contrast vertical-contrast-estimate-rule
      (premise
        (and (?process-plan aligning ?aligning)
             (?process-plan wafer ?wafer)
             (?aligning wavelength ?wl (plusp ?wl))
             (?wafer refractive-index ?ri (plusp ?ri))))
      (conclusion
        (?aligning vertical-contrast
                   `(contrast-from-refractive-indices
                                '(1.65 -0.02) ?ri ?wl))))

(defun contrast-from-refractive-indices (index-1 index-2 lambda)
  (let* ((num-real (- (car index-1) (car index-2)))
         (num-imag (- (cadr index-1) (cadr index-2)))
         (den-real (+ (car index-1) (car index-2)))
         (den-imag (+ (cadr index-1) (cadr index-2)))
         (refl (sqrt (/ (+ (* num-real num-real) (* num-imag num-imag))
                        (+ (* den-real den-real) (* den-imag den-imag)))))
         (angle (- (atan (/ num-imag num-real))
                   (atan (/ den-imag den-real))))
         (internal-lambda (/ lambda
                             (sqrt (+ (* (car index-1) (car index-1))
                                      (* (cadr index-1) (cadr index-1))))))
         (atten 1.0)
         (r 0.0))
    (cond ((minusp angle) (setq angle (+ 3.1416 angle)))
          ((geq angle 3.1416) (setq angle (- angle 3.1416))))
    (setq atten (exp (- (* .000026 internal-lambda (/ angle 3.1416)))))
    (setq r (* refl atten))
    (/ (* 2.0 r) (+ 1.0 (* r r)))))
```

Code List 10.1: The mini-expert for calculating the vertical contrast. The mini-expert is a rule domain, but the rule is used only for argument collection and range checking. When the rule concludes, the function is called and its return value is "spliced in" to the appropriate frame's slot.

θ becomes important when calculating the attenuation, as the phase shift introduced by imaginary components of the reflection alters the distance of the I_{max} and I_{min} points from the boundary. The distance of the I_{max} point from the boundary is

$$d = \frac{\theta}{2\pi}\lambda_{internal}$$

where $\lambda_{internal}$ is the wavelength of the exposing radiation inside the resist. Attenuation is an exponential function of the distance traveled; since the reflected radiation travels to the boundary and back, the attenuation coefficient is

$$A_a = e^{-2kd}$$

where k is a known value of the resist, typically about 0.2–0.3 μm^{-1}.[6] The final relative amplitude of the reflected radiation R is then

$$R = A_r A_a$$

The function `contrast-from-refractive-indices` shown in Code List 10.1 performs these calculations. It is passed the complex refractive indices, each as a two-element list, and the exposuring radiation wavelength in Angstroms. The resist's refractive index is assumed to be $1.65 - 0.02j$. The function first finds the real and imaginary parts of the boundary reflection. It then calculates A_{r0} and θ (called `refl` and `angle` respectively). After correcting for θ being negative or greater than π, it calculates A_a (called `atten`), finds R, and returns C_v.

[6] A value of 0.26 μm^{-1}, the approximate mean of the popular resists in use at ERL, was chosen for this calculation.

Graph Lookup

The graph lookup mini-expert presented here is the one used to find horizontal contrast of the projected image. This is a three-dimensional graph consisting of six curves, each showing the relationship between horizontal contrast and normalized spatial frequency for a single value of normalized defocus distance. The graph is shown in Figure 10.2. The code for this mini-expert is shown in Code List 10.2.

10.3.3 Verification or Simulation

An important part of the IC process design loop is verifying or simulating a proposed design. For this reason, the "Verify" command was added to Cameo's main browser. At this time, Cameo only prepares a process description for the SAMPLE [Oldh79,Oldh80] process simulator, and can only perform verification once a complete plan has been designed. Ideally the user would be able to verify plans incrementally in addition to globally, perhaps with a "check consistency" command that would perform fast local consistency checks of the current design decision. Since SAMPLE is used heavily at UC Berkeley for IC process simulation and is regarded as an accurate process simulator, a good agreement between Cameo's predictions and SAMPLE results should help validate Cameo's performance.

A separate frame not associated with any plan, `*sample*`, has one slot for each SAMPLE command. When the user gives the "Verify" command, Cameo changes to the `collect-sample-parameters-domain` rule domain and runs HPRL's inference engine once for each SAMPLE command

```
(add-graph-to-list
  (make-instance 'three-d-graph
                 'graph-name 'horizontal-contrast-for-sigma-7
                 'x-axis 'relative-spatial-frequency
                 'family-var 'relative-defocus
                 'curves '((0.0 (0.0 100.0) (.2 100.0) (.36 99.0)
                                (.43 98.0) (.53 96.0) (.70 94.0)
                                (.85 91.0) (1.0 80.0) (1.38 32.0)
                                (1.58 9.0) (1.7 0.0) (2.0 0.0))
                           (0.4 (0.0 100.0) (.2 100.0) (.36 98.0)
                                (.45 96.0) (.54 94.0) (.62 92.0)
                                (.82 88.0) (.93 81.0) (1.04 71.0)
                                (1.3 39.0) (1.55 11.0) (1.7 0.0)
                                (2.0 0.0))
                           (0.8 (0.0 100.0) (.2 100.0) (.36 98.0)
                                (.41 94.0) (.5 90.0) (.96 71.0)
                                (1.54 10.0) (1.7 0.0) (2.0 0.0))
                           (1.2 (0.0 100.0) (.14 100.0) (.33 97.0)
                                (.5 83.0) (.56 81.0) (1.0 52.0)
                                (1.52 9.0) (1.7 0.0) (2.0 0.0))
                           (1.6 (0.0 100.0) (.14 100.0) (.32 93.0)
                                (.84 45.0) (1.46 9.0) (1.7 0.0)
                                (2.0 0.0))
                           (2.0 (0.0 100.0) (.12 100.0) (.26 95.0)
                                (.32 89.0) (.39 81.0) (.51 60.0)
                                (.76 33.0) (1.45 5.0) (1.7 0.0)
                                (2.0 0.0)))))
```

Code List 10.2: Code for the horizontal contrast graph. The graph consists of six curves, one for each value of normalized defocus distance from 0.0 to 2.0.

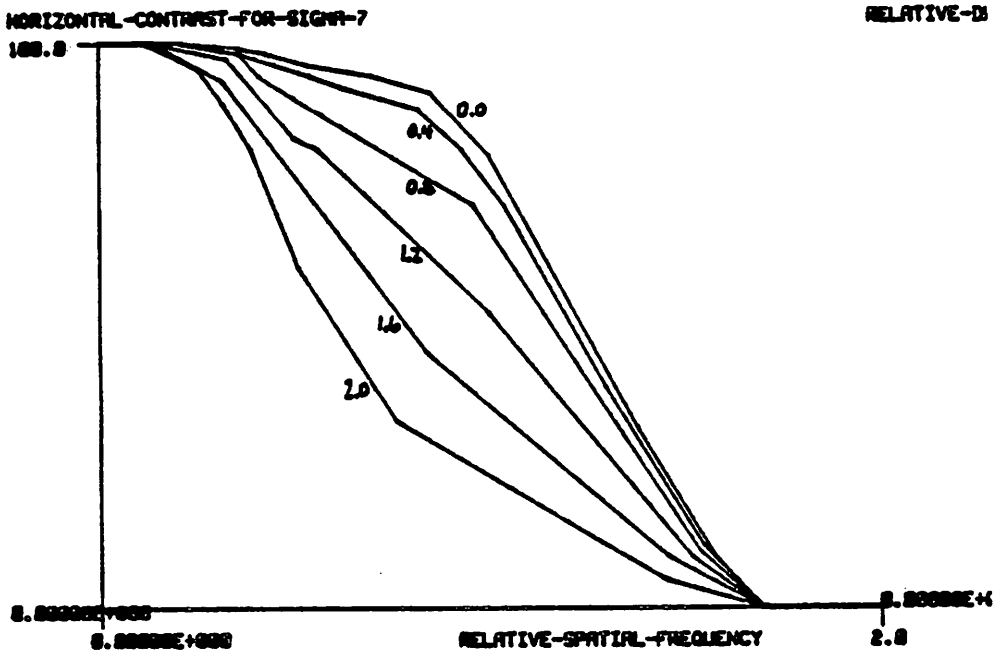

Figure 10.2: Graph for finding the horizontal contrast given normalized spatial frequency of the projected pattern and the normalized defocus distance.

needed by solving for each slot in the *sample* frame. The rules in the collect-sample-parameters-domain domain perform largely simple conversions of data in the current plan to data in SAMPLE's format. An example of one such rule is shown in Code List 10.3. When HPRL's inference engine has been called on all slots in the *sample* frame, the data in the frame is written out to a file.

Ideally, at this point a procedure would be called that transferred this file to a "simulation server" running on a remote computer which has ex-

```
% The DEVTIME command is given a set of arguments to simulate
% at 10, 30, 50, 70, and 90 seconds per micron of resist thickness.

(rule devtime-generator collect-sample-parameters-rule
      (premise (and (*sample* plan ?process-plan)
                    (?process-plan resist-scheme ?resist-scheme)
                    (?resist-scheme thickness ?th (plusp ?th))))
      (conclusion (*sample* devtime `(list (round (* ?th 10))
                                           (round (* ?th 90))
                                           5))))
```

Code List 10.3: A rule for generating SAMPLE process simulator input.

cellent floating-point performance, and the output would be sent back to Cameo in a graphic form for display. Currently, due to some minor technical interface difficulties between the H-P 9836 computer and the VAXes used as the remote computers, this is not yet done. The user may request a hardcopy of the prepared SAMPLE input file and type it in to the remote computer manually.

An example of the interface to SAMPLE is given here. The user has prepared the photolithography plan shown in Figure 10.3. The SAMPLE input file generated is shown in Figure 10.4, and SAMPLE's output is shown in Figure 10.5. The output shows a cross-section through the resist for the five values of development time from 12 to 108 seconds. Note that the variation of the linewidth at the 60-second mean development time, an increase of about 0.11 μm, corresponds fairly closely to the estimated linewidth variation predicted by Cameo, 0.07 μm.

The capability of tying Cameo to existing resources such as SAMPLE is a key element of Cameo's attractiveness. This interface was developed in about two hours of programming time, and is very simplistic but already useful.

```
Plan: Initial Plan
    --PROCESS REQUIREMENTS--
        Minimum Feature Size: 2 microns
        Maximum Allowable Linewidth Variation: 0.2 microns
        Alignment Accuracy (3 sigma)
        Optimizing For: DIMENSION-CONTROL
    --WAFER STATE--
        Wafer Diameter: 4 inches
        Wafer Shape: ROUND
        Maximum Non-Planarity (Warpage): 0.5 microns
        Current Layer: POLY
            Thickness of Current Layer: 1 microns
            Refractive Index of Current Layer: 4.82-0.12j
        Largest Die Dimension: 5 mm
        Maximum Step Size: 0.4 microns
        Previous Layers
    Aligning and Exposure: GCA-1
        Machine: GCA   [more...]
        --Contrast and Linewidth Variation Calculations--
            Vertical Contrast: 73 percent
            Normalized Defocus Distance: 1.40
                Normalized Spatial Frequency: .389
                   Horizontal Contrast: 89 percent
                      Estimated Linewidth Variation: 0.06788 microns
                      Recommended Resist Treatment
                      Recommended Etch Treatment
   Resist Scheme: KTI820-1
       --Preparation--   [more...]
       Resist: KTI820
           Resist Thickness: 1.2 microns
               Viscosity: 27.0 cst
                   Spin Speed: 5888.88889 rpm
               Exposure Dose: 140 mj/cm-cm
           Refractive Index: 1.62-0.02j   [more...]
       --Pre-Bake--   [more...]
       --Develop--   [more...]
   Etch Method
```

Figure 10.3: The plan verified with SAMPLE.

```
lambda 0.4358;
proj 0.28;
linespace (2 2);
resmodel (0.4358 0.51 0.031 0.014 (1.62 -0.02) 1.2);
layers (1.47 0.0001 (4.82 -0.12) 1);
dose 140;
devrate 1;
devtime (12 108 5);
trial (20 0 0.7 3.9);
run;
```

Figure 10.4: The SAMPLE input file prepared from the plan.

```
Symbol:     time:      resist-substrate intersection:    sidewall angle estimate
                                    (by a straight line fit to all the CDmins)
   a      12.0 sec
   b      36.0 sec         x =   0.1885 micrometers            79.5 degrees
   c      60.0 sec         x =  -0.0559 micrometers            81.0 degrees
   d      84.0 sec         x =  -0.1668 micrometers            81.2 degrees
   e     108.0 sec         x =  -0.2586 micrometers            80.9 degrees

        The window is 2.0000 micrometers wide in x.
        The edge   is 1.0000 micrometers from the left side of the window.
```

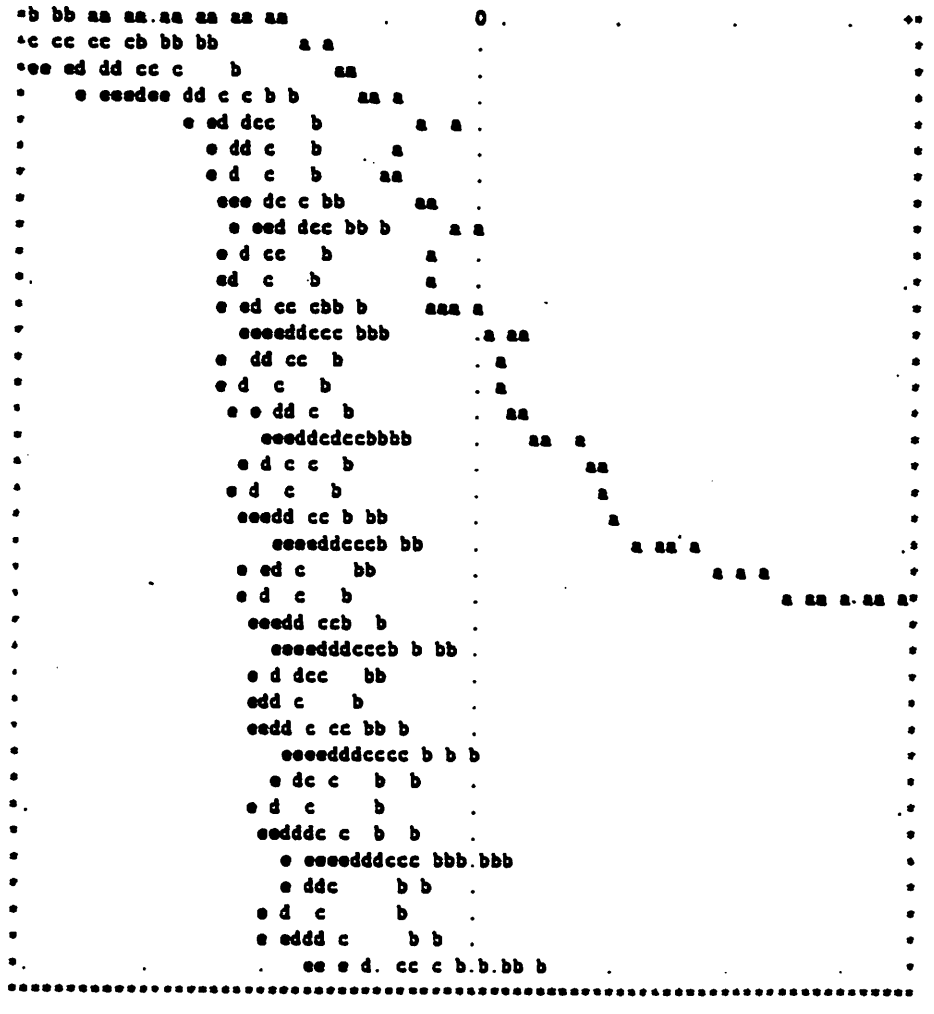

Figure 10.5: SAMPLE output. This is a cross-section through the resist.

CHAPTER 11

Conclusions

This research, being interdisciplinary across the fields of IC processing and Computer Science, contributes to both fields by tying them together. Its contributions to IC processing lie mainly in finding a structure to experts' problem structuring and solution methods. Its contributions to Computer Science are more in how its implementation was driven by its application.

Some of Cameo's unique qualities differentiate it from other expert systems in a way that users find especially useful. Other aspects of Cameo's structure should prove to be valuable for easing future extensions or maintenance. Some limitations have been pointed out that suggest directions for future work.

Cameo's highly modular structure makes it very easy to extend and modify. The ability to represent knowledge and data in their "most appropriate" forms eliminates one difficult programming task. The Lisp language

made Cameo's implementation unusually easy.

11.1 Contributions

This research being interdisciplinary in nature, its contributions lie mainly in tying the disciplines together rather than making fundamental contributions to them. The most important contributions are that a structure to a subset of the IC process design activity has been found, and that that structure has been used to design a computer-based design assistant that leads the user through the same structured decisions that an expert IC process designer would.

These contributions are nonetheless important and valuable. IC process design is currently in a state where it has little structure and few generally-accepted design principles. One of the most difficult parts of this research was to convince experts to think in general terms and specify their "rules of thumb" about their problem structuring and solution methods. One specific contribution, then, is an attempt to simplify and generalize the decision process of an expert IC process designer evaluating potential approaches to linewidth control.

Some of the ideas and principles behind Cameo's software implementation have been used before in other CAD systems. Cameo's implementation, however, is unique because its specific application is unique (aiding novice IC process engineers at UC Berkeley's Electronics Research Laboratory (ERL) understand and evaluate approaches to linewidth control). Drawing from

comments of people to whom Cameo was demonstrated, it appears that many of the ideas behind Cameo's implementation are applicable more generally than to its current application. Thus it may be instructive to study the derivation of the implementation in response to the application's requirements.

11.2 Observations

11.2.1 Observations from Potential Users

As of this writing, Cameo has not been used for designing IC photolithography steps, but has been demonstrated to approximately twenty potential users including both professional IC processing experts and complete novices. Thus the following observations are not the results of a rigorous user study but are distilled from questions and comments voiced during these demonstrations.

Most people approached for giving demonstrations and potentially using Cameo were initially reluctant to try an AI-based CAD system. Without exception, however, once these people actually saw the system in use, they were eager to use it. Four reasons were largely responsible for Cameo's positive impression:

- Cameo is highly interactive and under the user's control much more than most expert systems.

- Most doubters asked something like, "What if the user doesn't know what this item on the screen means?" This question quickly disappeared once they saw that every item shown on the screen had an associated references file that answered this question.

- Cameo allows the user to work on more than one alternative design during a single session. Most CAD systems make it very difficult for users to evaluate alternative designs.

- Cameo could be considered to be a "front end" for existing IC process CAD tools like simulators. Thus these popular and well-known (among Cameo's users) tools were being enhanced, not replaced.

Less important was Cameo's potential for establishing a "corporate memory" of photolithography steps. Most expert IC process designers are skeptical that such a library of photolithography steps will prove to be of great value. Novice IC process designers, however, welcome this feature because it gives them a starting point from which to design their own process steps. Evaluation of this aspect of Cameo's structure will require more people actively using the system.

Some potential users raised questions about whether Cameo could support the highly iterative nature of design, and whether its relatively static decision sequence structure would work against this. Most were satisfied that they could split a plan at any time and work on the resulting ones independently. A truly general solution to this issue encompasses dealing with the

dependencies among the design decisions. This is a limitation that should be addressed by future work; a later section (Section 11.3.3 on Page 178) concentrates on what this work might entail.

Concerns were raised by most experts about the difficulty of maintaining the system's knowledge base. This is a real concern, and although Cameo's modularity should prove to minimize this problem, a substantial amount of maintenance will be necessary. This maintenance is useful in its own right, however, because it increases the understanding of the IC photolithography steps themselves and serves to document new developments in the field in an easily accessible system.

11.2.2 Observations from Cameo's Development

Cameo is a large system. Aside from PSL and HPRL itself, the code involved with supporting its framework consists of about 3500 lines of well-commented Lisp. The code that forms the knowledge base and the specific structure of Cameo's application totals about 1500 lines of Lisp. The underlying PSL and HPRL provide the basic foundations of the system such as the Lisp language itself with object-oriented programming extensions, the screen-oriented editor, the browser mechanism, and the frame and rule system.

Programming effort, however, was proportionally low for a system of this size. Much of this is because an advanced Lisp system like PSL offers an exploratory programming environment where new ideas can be tried within moments and do not require recompilation. Bugs can be identified quickly

with PSL's interactive debugger that takes over immediately whenever a system exception occurs or a user-set breakpoint is reached. Another important programming feature is object-oriented programming which allows an unusually highly modular system to be developed.

HPRL's frame and rule system is equally easy to program with. The easy interface between PSL, HPRL's frames and rules, and object-oriented programming allows combinations of these seemingly limited only by the programmer's imagination. The inheritance properties of HPRL's frames and rules allow the grouping of similar data structures in efficient and conceptually simple ways. HPRL I's inference engine for backward chaining is simple but sufficient and allows an example expert system to be built very quickly.

There is, of course, a price to be paid for these advantages. Without unconventional hardware support, Lisp runs slowly and inefficiently on standard computers. This is not a fundamental drawback of Lisp, but is a reflection on conventional styles of hardware design. Programming languages have evolved over the last 20 to 30 years to take best advantage of the hardware they are intended to run on. Their structure closely reflects the hardware's architecture. Languages such as Lisp and Smalltalk [Gold83], on the other hand, are designed for special applications and are not designed primarily for efficient execution on commonly available hardware. Hardware designed specifically for efficient execution of Lisp or Smalltalk has shown that systems written in these languages can run very fast at high efficiency.

From a system development standpoint, dividing a large and complex knowledge base about a technical field into small, relatively independent

chunks makes the system easy to understand, highly modular, and easily extended and modified. The flexibility of being able to represent these chunks of knowledge in their most appropriate forms eliminates one of the more difficult implementation problems, deciding how to fit a known bit of knowledge or data into an inflexible representation scheme. If new knowledge or data representations are needed, they can easily be added to the existing repertoire.

The four types of mini-experts used in Cameo's specific application are production rules, numeric graph lookups with interpolation, symbolic tables, and calls to formulas or arbitrary procedures. Once the application's structure, or decision sequence, was found, it was never difficult to decide on the mini-expert types.

11.3 Directions for Future Work

11.3.1 Extending Cameo

Cameo currently addresses a small subset of the overall IC photolithography design problem, although the problems it addresses are especially important. Cameo currently only handles resist exposure and development on metal and polysilicon layers. Three directions suggest themselves immediately:

- Extend Cameo to handle the etching step.
- Extend Cameo to the remaining layers of the IC process.

- Improve the generation of simulator input from data in each process plan.

11.3.2 Increase Cameo's Portability

While HPRL is an excellent environment for developing an expert system, it has a major limitation in that it can currently only be run on a few select Hewlett-Packard computer systems. The operating system used on these computers does not allow easy access to other computers or a high-speed local area network. One of the highest priority further activities is to port Cameo to more commonly available hardware and a standard operating system. There are a number of possible ways to do this. Each must be evaluated in terms of the benefits versus the risks and the costs (monetary and time) involved.

Cameo is implemented in HPRL I. The expert systems group at Hewlett-Packard Laboratories has recently announced a new version of HPRL, HPRL II. HPRL II is not upward compatible with HPRL I but its basic structure is similar, so porting Cameo to HPRL II should not be difficult. With Hewlett-Packard's recent corporate commitment to HP-UX, its version of the UNIX[1] operating system, it can be assumed that HPRL II will eventually be available for UNIX systems. Once available for a /mboxUNIX system, it should be much easier to port to other systems. This strategy

[1]UNIX is a trademark of AT&T.

would probably be the least expensive and least time-consuming, but its success depends on various factors that cannot be controlled.

As another porting strategy, the subset of HPRL I's capabilities actually used by Cameo could be rewritten in a version of Lisp that is highly portable. This might include PSL, Franz Lisp, Zeta Lisp, or Common Lisp. This porting strategy probably entails substantially more work than the former, but does not rely on HPRL II becoming available on a UNIX or UNIX-like system. Thus this strategy entails less risk but will cost more in development effort.

Another strategy would be to rewrite Cameo in a different expert systems development environment. This has already been done with one expert system developed in HPRL I at UC Berkeley, RUBICC [Spic85], which was ported to run under OPS5 [Forg81]. A number of expert systems development environments currently exist or are being prepared as commercial products. Again, this strategy entails substantially more work than the first, but probably gives the most long-term flexibility at a low risk since the development environment used will be complete and may offer increased capabilities over the other porting strategies. The monetary cost may be substantial, especially if a commercial product is used as the development system.

11.3.3 General System Improvements

During the application of Cameo's framework to linewidth control, certain limitations were found that suggest general improvements to the system.

Dependencies

Cameo does not have a clear-cut way to deal with data dependencies. Each description frame has a slot called depends-on which lists all the slots this slot's value depends on, but the current version of Cameo does not use this information.

One confusing problem occurs when a user asks Cameo to derive an answer to a design decision, and Cameo responds that it cannot find an answer. This is usually because some of the data needed has not yet been found. The user does not know which data is needed, so the only real choice he has is to go through all possible design decisions and make sure they have values, then try again to derive the answer to the original design decision.

Since the dependency data is stored in the description frames, all data necessary to solve the problem already exists. One method would be to use a graphic representation of the complete decision tree showing all dependencies. The user could point to the desired decision and Cameo could tell him which decisions need to be solved first. This should be interactive and tied in to Cameo's main browser. This could also make use of the "view" concept explained in Section 8.1.2 on Page 115 since the graphic data dependencies form a different view of the same IC process plan.

Another dependency problem, related to the previous one, is that the outline form of Cameo's browser will allow the user access to a design decision whenever its superior decision in the outline has been given a value. In practical design decisions, each decision depends on more than one other

decision, and should not be accessible to the user until all the decisions it depends on have been solved. Thus the plan structuring should actually be dynamic, allowing the user access only to those decisions where all the decisions they depend on have been solved. Again, since the entire structure of the browser is kept in description frames, meta knowledge can be developed that performs this function.

Another dependency problem occurs when the user overrides a value already in a plan. Currently Cameo simply walks down the browser outline and removes values from design decisions that appear subordinate to the overridden decision. While this is often sufficient, in reality some design decisions in a different part of the plan outline may depend on one of the values overridden or removed. The correct action is to invalidate all dependent decisions. Invalidation may involve removing the dependent value from the plan, but it may be more helpful to the user to leave a marker with the decision's display indicating that its value may be invalid. The user may decide to keep the value as it is or derive or provide a new one. Highly relevant work to this problem is in the area of constraints [deKl80].

Extending Cameo to Help Experts

Cameo's current structure enforces a relatively rigid structure on the decision sequence and thus does not allow users to break away from this predetermined structure. For Cameo's current application this is desirable. Breakthroughs in synthesis, however, are often accomplished by violating an accepted structure. For Cameo to be most useful for experts, it should allow its

novice-oriented decision structure to be broken, offering just "housekeeping" services to the user to keep the plan structures intact.

Some parts of this extension can be accomplished by transferring more of Cameo's structure to meta-knowledge and by letting the user override its own structuring. This is an interesting and potentially fruitful area of research, as experts using Cameo's capabilities could also realize significant productivity gains if they are not hindered by too-rigid structuring.

Enabling Users to Add Knowledge

As users gain experience with Cameo, they will want to add notes and advice to the reference files. This capability could be added relatively easily by using existing capabilities of the NMODE editor. The users could be placed in a special buffer and asked to contribute a short note. The contents of the buffer, along with the user's name and the current time, could be inserted into the existing reference file.

A long-term possibility is to allow users to add their own steps into these libraries and make comments about any library steps, thereby making this information immediately accessible to the entire user community. The potential benefits of this possibility need to be evaluated carefully, as this may lead to an explosion of data.

Programming Aids

Modification or addition to the existing mini-experts requires some knowledge of Cameo's structure that is not evident from interacting with it on the user's level, such as HPRL frame structure and syntax, and frame and slot names of both the plan skeleton and the description frames. If HPRL rules are used by the mini-expert, the rule syntax and semantics would need to be understood too. Someone familiar with Lisp should be able to understand this without too much difficulty.

A good long-term solution would be to have functions that build templates of mini-experts and fill out or modify relevant slots. This is an interesting research project in itself. Some of the more advanced Lisp systems and Xerox's Smalltalk-80 system have capabilities like this that could be used as models.

Saving and Restoring Plans

Once a user has assembled one or more plans, a capability to save them on mass storage is needed. An ideal way to do this is to use internal HPRL functions that can dump the contents of a frame into a given file. HPRL also provides companion functions to restore a dumped frame. The frame hierarchy for the plan to be saved would have to be flattened so no inheritance is used to find values. The restored plan would become a direct instance of the plan skeleton. Some work would also need to be done to eliminate session-specific data from the frame like pointers to specific frames.

Object Type	Representative Messages
item-display	expand collapse kill display-text
mini-expert	solve user-input explain replace
bw-chain-expert	solve-specific
two-d-graph-expert	solve-specific
three-d-graph-expert	solve-specific
matrix-expert	solve-specific
null-heading-expert	solve-specific

Table 11.1: Improved internal structure. The display and problem-solving actions are explicitly separated.

Internal Structure

Cameo's internal software structure evolved over a period of about a year, and so reflects ideas from earlier development that may not be applicable to its current state. One structure improvement is explicitly dividing display and problem solution actions. These are currently intermingled in the generic `control` object and its five specific objects.

An improved structuring might be as shown in Table 11.1. Each item on the screen is now represented by an object of type `item-display` which controls only how items are to be shown. Each mini-expert is an instance of one of the five specific expert objects. This is in contrast to the current

implementation where each item on the screen is represented by an instance of one of the five specific `control` objects and the division between display and problem-solving actions is not nearly as explicit.

This restructuring would improve the modularity of the system and ease modifications and extensions. Since the display mechanism, which will probably vary from one computer system to another, is independent of the problem-solving mechanism, Cameo's portability should be improved. With the "view" concept introduced in Section 8.1.2 on Page 115 and suggested for use with dependencies in Section 11.3.3 on Page 178, this structuring could help show different views of IC process plans by defining a different kind of `item-display` object for each view.

Performance Monitoring

Monitoring of certain aspects of Cameo's performance and use would be useful for eventual optimizations. For instance, if a certain library step is chosen much more often than others, then it might be useful to provide a few popular versions of that library step.

Other monitoring that might be useful would be for the aim of improving the user interface. A record can be kept of the operations performed by users. Certain sequences of operations will undoubtedly be found to occur often. Examining these operation sequences could point to an inefficiency in the user interface, a missing feature, ambiguities, poor plan structure, or simply new commands that could replace these sequences of commands.

Other conventional performance monitoring would also be useful in a longer term, such as identifying performance bottlenecks in the code and finding more optimal data structures or algorithms that would speed Cameo's execution. Two bottlenecks are already known by observation:

- Each time one or more items are added to Cameo's browser, all items are resorted. Sorting over all items shown in the browser is the method used to keep design decisions in the form of an outline. As the number of items in the browser increases, this sorting time becomes quite long, on the order of 10 seconds in some practical cases.

- When expanding a heading that results in many new items added to the browser display, a great deal of time is spent finding the actual text displayed for each item. While this is necessary to retain the dynamic nature of the system, it costs quite a bit of performance.

11.4 Other Applications for Cameo's Framework

While Cameo is intended specifically for synthesis of IC processes, most of the code and structure do not limit it to this application alone. Any application where the decision process can be structured into iterative paths along decision trees with references and mini-experts at each node are candidates. With more work on solving the problems referred to earlier on dependencies, the decision process may not even need to be limited to a decision tree.

Some potential applications of Cameo's framework are in diagnosis and educational instruction.[2] Both of these application areas appear to be relatively straightforward using Cameo's present framework.

Although Cameo is designed to work with libraries of fundamental IC photolithography steps, the code that assumes this is not extensive. Cameo does not make assumptions about the number of libraries (three in Cameo's current application). Cameo could even be modified to work with a plan skeleton that is a single frame, but this is probably not particularly desirable or necessary.

As an example of an application to educational instruction, a library "step" could hold one lesson for each level of student expertise. Each library would hold all the experience levels for a single lesson, and heuristic rules would select the appropriate experience level based on the user's answers to some initial questions that would take the place of Cameo's current "Process Requirements" or "Wafer State." Each lesson would take the place of one of the current fundamental IC photolithography steps.

[2] These two applications were suggested by William Holton of the Semiconductor Research Corporation on a visit to UC Berkeley's Electronics Research Laboratory in July 1985.

Bibliography

[Anto79] Antoniadis, D.A., Dutton, R.W., "Models for Computer Simulation of Complete IC Fabrication Process," *IEEE Transactions on Electron Devices*, volume ED-26, Number 4, April 1979, pp. 490–500.

[Beeb83] Beeby, W.A., "The Heart of Integration: A Sound Data Base", *IEEE Spectrum*, vol. 20, number 5, IEEE, New York, NY, May 1983, pp. 44–48.

[Boni85] Boning, D.S., Antoniadis, D.A., "MASTIF—A Workstation Approach to Fabrication Process Design," submitted to ICCAD 1986.

[Brow77] Brown, A.L., "Qualitative Knowledge, Causal Reasoning, and the Localization of Failures," Ph.D. dissertation, MIT AI Labs Technical Report No. 362, 1977.

[Brow83] Brown, H., Tong, C., Foyster, G., "Palladio: An Exploratory Environment for Circuit Design," *IEEE Computer Magazine*, volume 16, number 12, December 1983, pp. 41–56.

[Bush83a] Bushnell, M.L., "Delilah II—An Enhanced Menu-Driven Input Processor," *Research Report CMUCAD-83-7*, SRC-CMU Center for Computer-Aided Design, Department of Electrical and Computer Engineering, Carnegie-Mellon University, February 1983.

[Bush83b] Bushnell, M.L., et. al., "DIF: The CMU-DA Intermediate Form," *Research Report CMUCAD-83-11*, SRC-CMU Center for Computer-Aided Design, Department of Electrical and Computer Engineering, Carnegie-Mellon University, February 1983.

[Bush85] Bushnell, M.L., Director, S.W., "ULYSSES—An Expert-System Based VLSI Design Environment," *SIGART Newsletter*, Association for Computing Machinery Special Interest Group on Artificial Intelligence, number 92, pp. 82–84, April 1985.

[Byer83] Byers, T., "Optimal and Near-Optimal Policies for Discrete Deterministic Dynamic Programming Models," Ph.D. dissertation, Graduate School of Business, University of California at Berkeley, 1983.

[Clan84] Clancey, W.J., "Knowledge Acquisition for Classification Expert Systems," presented at ACM '84, San Francisco, CA, 1984.

[Clin85] Cline, T. et. al., "Photolithography Advisor," *SIGART Newsletter*, Association for Computing Machinery Special Interest Group on Artificial Intelligence, number 92, pp. 42–43, April 1985.

[Date81] Date, C.J., *An Introduction to Database Systems*, Third Edition, volume 1, Addison-Wesley Publishing Company, Reading, MA. ISBN 0-201-14471-9.

[deKl80] de Kleer, J., and Sussman, G.J., "Propagation of Constraints Applied to Circuit Synthesis," *Circuit Theory and Applications*, volume 8, John Wiley & Sons, 1980, pp. 127–144.

[Dire81] Director, S.W., et. al., "A Design Methodology and Computer Aids for Digital VLSI Systems," *IEEE Transactions on Circuits and Systems*, CAS-28, July 1981, pp. 534–645.

[East81] Eastman, C.M., "Database Facilities for Engineering Design," *IBS Research Report No. 14*, Department of Architecture, Carnegie-Mellon University, March 1981.

[Elli82] Elliot, D.J., *Integrated Circuit Fabrication Technology*, McGraw-Hill, Inc., 1982. ISBN 0-07-019238-3.

[Fole82] Foley, J.D., Van Dam, A., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Reading, MA, 1983. ISBN 0-201-14468-9.

[Fole84] Foley, J.D., Wallace, V.L., Chan, P., "The Human Factors of Computer Graphics Interaction Techniques," *IEEE Computer Graphics and Applications*, volume 4, number 11, IEEE Computer Society, November, 1984, pp. 13–48.

[Forg81] Forgy, C.L., "OPS5 User's Manual," *Research Report CMUCAD-81-135*, SRC-CMU Center for Computer-Aided Design, Department of Electrical and Computer Engineering, Carnegie-Mellon University, July 1981.

[Gajs84] Gajski, D.D., Bozek, J.J., "ARSENIC: Methodology and Implementation," Digest of Technical Papers, IEEE International Conference on Computer-Aided Design, November 1984, pp. 116–118.

[Gene84] General Motors Corporation Public Relations Staff, "MAP: The Tie That Binds — Communication on the Plant Floor," *Public Affairs Newsletter*, volume 14, number 8, September 1984.

[Grim84] Grimes, J.D., Guest Editor's Introduction, "Human Factors—Part 1," *IEEE Computer Graphics and Applications*, November 1984, pp. 10–11.

[Gold83] Goldberg, A., Robson, D., *Smalltalk-80—The Language and its Implementation*, Addison-Wesley Publishing Company, Reading, MA, 1983.

[Gyur85] Gyurcsik, R.S., private communication, Electronics Research Laboratory, UC Berkeley, June 1985.

[Haye83] Hayes-Roth, F., Waterman, D.A., and Lenat, D.B., eds., *Building Expert Systems*, Addison-Wesley Publishing Company, Inc., Reading, MA., 1983.

[Ho83] Ho, C.P., Hansen, S.E., "SUPREM III—A Program for Integrated Circuit Process Modeling and Simulation," Stanford University Technical Report SEL 83-001, July 1983.

[Kell82a] Keller, K., "A Symbolic Layout Design System," *Proceedings of the 1982 IEEE International Symposium on Circuits and Systems*, 1982.

[Kell82b] Keller, K., "A Symbolic Design System for Integrated Circuits," *Proceedings of the 1982 Design Automation Conference*, 1982.

[Kram84] Kramer, G.A., "Brute Force and Complexity Management: Two Approaches to Digital Test Generation," SM Thesis, Massachusetts Institute of Technology, June 1984.

[Kram85] Kramer, G.A., "Helios Design Consultant System," *SIGART Newsletter*, Association for Computing Machinery Special Interest Group on Artificial Intelligence, number 92, pp. 76–78, April 1985.

[Kowa83] Kowalski, T.J., Thomas, D.E., "The VLSI Design Automation Assistant: A Prototype System," *Proceedings of the 20th ACM/IEEE Design Automation Conference*, June 1983, pp. 479–483.

[Kowa85] Kowalski, T.J., Thomas, D.E., "The VLSI Design Automation Assistant: What's in a Knowledge Base," *Proceedings of the 22nd ACM/IEEE Design Automation Conference*, June 1985, pp. 252–258.

[Lana83a] Lanam, D., Letsinger, R., Rosenberg, S., "Guide to the Heuristic Programming and Representation Language Part 1: Frames," AT-MEMO-83-3, Applications Technology Laboratory, Computer Research Center, Hewlett-Packard Company, Palo Alto, CA, 1983.

[Lana83b] Lanam, D., Rosenberg, S., Letsinger, R., "Guide to the Heuristic Programming and Representation Language Part 2: Rules," AT-MEMO-83-4, Applications Technology Laboratory, Computer Research Center, Hewlett-Packard Company, Palo Alto, CA, 1983.

[Leop84] Leopold, G., "Factory Nets Follow a MAP," *Electronics Week*, December 17, 1984, pp. 20–21.

[Lob84] Lob, C., "RUBICC—A Rule-Based Expert System for VLSI Integrated Circuit Critique," M.S. thesis, Electronics Research Lab, UC Berkeley, 1984.

[Math77] Mathlab Group, The, *MACSYMA Reference Manual*, MIT Laboratory for Computer Science, Cambridge, MA, 1977.

[Mead80] Mead, C.A., Conway, L., *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, Reading, MA, 1980.

[Mill56] Miller, G.A., "The Magical Number Seven—Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Review*, 63, 1956, pp. 81–97.

[Nage75] Nagel, L.N., "Spice 2: A Computer Program to Simulate Semiconductor Circuits," UCB/ERL Memo M520, Electronics Research Laboratory, University of California at Berkeley, May 1975.

[Nand84] Nandgaondkar, S.N., "A Family of Simulation Programs for IC Fabrication Processes (Their Structure, Design and Implementation)," Ph.D. dissertation, UCB/ERL Memo M84/90, Electronics Research Laboratory, UC Berkeley, October 1984.

[Nass83] Nassif, S.R., Strojwas, A.J., and Director, S.W., "FABRICS II: A Statistically Based IC Fabrication Process Simulator," *IEEE Transactions on CAD of ICAS*, volume 3, number 1, January 1984, pp. 40–46.

[Neur81] Neureuther, A.R., Jain, P.K., Oldham, W.G., "Factors Affecting Linewidth Control Including Multiple Wavelength Exposure and Chromatic Aberration," *Semiconductor Microlithography VI*, Society of Photo-Optical Instrumentation Engineers, volume 275, Bellingham, WA, 1981, pp. 110–116.

[Nye83] Nye, W.T., "DELIGHT: An Interactive System for Optimization-Based Engineering Design," Ph.D. dissertation, UCB/ERL Memo M83/33, Electronics Research Laboratory, UC Berkeley, 1983.

[Oldh79] Oldham, W.G. et. al., "A General Simulator for VLSI Lithography and Etching Processes: Part I—Application to Projection Lithography," *IEEE Transactions on Electron Devices*, volume ED-26, number 4, April 1979, pp. 717–722.

[Oldh80] Oldham, W.G. et. al., "A General Simulator for VLSI Lithography and Etching Processes: Part II—Application to Deposition and Etching," *IEEE Transactions on Electron Devices*, volume ED-27, number 8, August 1980, pp. 1455–1459.

[Oldh81] Oldham, W.G., Subramanian, S., Neureuther, A.R., "Optical Requirements for Projection Lithography," *Solid-State Electronics*, volume 24, number 10, pp. 975–980, Great Britain, 1981.

[Oldh85] Oldham, W.G., private communication, Electronics Research Laboratory, UC Berkeley, January 1985.

[ONei79] O'Neill, L.A. et. al., "Designer's Workbench—Efficient and Economical Design Aids," *Proceedings of the 16th ACM/IEEE Design Automation Conference*, June 1979, pp. 185–199.

[Oust84] Ousterhout, J., et. al., "Magic, a VLSI Layout System," *Proceedings of the 21st ACM/IEEE Design Automation Conference*, June 1984, pp. 152–159.

[Pan83] Pan, Y-C., "Qualitative Reasonings with Deep-Level Mechanism Models for Diagnosis of Dependent Failures," Ph.D. dissertation, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Report No. T-132, Dec. 1983.

[Pete83] Peterson, J.L., Silberschatz, A., *Operating System Concepts*, Addison-Wesley Publishing Company, Reading, MA., 1983. ISBN 0-201-06097-3.

[Pint84] Pinto, M.R., Rafferty, C.S., Dutton, R.W., "PISCES II: Poisson and Continuity Equation Solver," Stanford Electronics Laboratories Report, September 1984.

[Poly57] Polya, G., *How to Solve It—A New Aspect of Mathematical Method*, Doubleday & Company, 1957.

[Rich79] Rich, C., and Shrobe, H.E., "Design of a Programmers Apprentice," *Artificial Intelligence, an MIT Perspective*, volume 1, Winston and Brown, eds., The MIT Press, Cambridge, MA, 1979.

[Robe77a] Roberts, R.B., Goldstein, I.P., "The FRL Manual," MIT AI Memo 409, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, June 1977.

[Robe77b] Roberts, R.B., Goldstein, I.P., "The FRL Primer," MIT AI Memo 408, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, July 1977.

[Rose82] Rosenberg, S.,"HPRL: A Language for Building Expert Systems," *Proceedings of the International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, 1982.

[Shne80] Shneiderman, B., *Software Psychology—Human Factors in Computer and Information Systems*, Winthrop Publishers, Inc., Cambridge, MA, 1980. ISBN 0-87626-816-5.

[Siew83] Siewiorek, D.P., Giuse, D., Birmingham, W.P., "Proposal for Research on Demeter—A Design Methodology and Environment," *Research Report CMUCAD-83-5*, SRC-CMU Center for Computer-Aided Design, Department of Electrical and Computer Engineering, Carnegie-Mellon University, January 1983.

[Sing83] Singh, N., "MARS: A Multiple Abstraction Rule-Based Simulator," Memo HPP-83-43, Heuristic Programming Project, Stanford University, December 1983.

[Sing84] Singh, N., "Corona: A Language for Describing Designs," Memo HPP-84-37, Heuristic Programming Project, Stanford University, June 1984.

[Spic85] Spickelmier, R.L., Newton, A.R., "A General Knowledge-Based Circuit Critic," *SIGART Newsletter*, Association for Computing Machinery Special Interest Group on Artificial Intelligence, number 92, pp. 78–79, April 1985.

[Stro84] Strojwas, A.J., "Statistical Process/Device Simulation and Its Applications for the Control and Diagnosis of the IC Manufacturing Line," presented at UC Berkeley Extension's Continuing Education in Engineering one-day course titled Computer-Aided Manufacture for Semiconductor Fabrication Applications, February 17, 1984.

[Stro85] Strojwas, A.J., "CMU-CAM System," *Proceedings of the 22nd ACM/IEEE Design Automation Conference*, June 1985, pp. 319–325.

[Suss79] Sussman, G.J., Holloway, J., Knight, T.F. Jr., "Computer Aided Evolutionary Design for Digital Integrated Systems," AI Memo 526, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, May 1979.

[Wrig83] Wright, J.M., Fox, M.S., *SRL/1.5 User Manual*, Robotics Institute, Intelligent Systems Laboratory, Carnegie-Mellon University, 1983.