BED: The Implementation of a Process Specification Editor

by

Stephan R. Smoot

Memorandum No. UCB/ERL M93-46

16 June 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California
94720

BED: The Implementation of a Process Specification Editor

Stephen R. Smoot

Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, California 94720
smoot@CS.Berkeley.EDU
May 21, 1993

Abstract

The Berkeley Process Flow Language (BPFL) is an application-driven programming language for modern semiconductor processing. This report describes the design and implementation of a graphical editor for BPFL. Semiconductor process specifications are complex and require elaborate constructs to specify manufacturing operations, simulation inputs, and dynamic scheduling. As a programming language, BPFL simplifies the expression of such tasks. However because programming languages are not good interfaces for non-programmers, a powerful and usable editor for the language is important. A graphical editor for BPFL (BED) has been developed. BED uses a library of processing abstractions to simplify specifications. BED has iconification and hyperlinks to pack more information into a small screen area.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Manufacturing a useful silicon wafer is a complicated process. It typically takes several weeks during which the wafer is processed by many different people and machines. Each production step involves precise specification of multiple parameters, some of which depend on the results of previous steps. To manage the manufacturing process, Work-In-Progress (WIP) systems have been developed to track the wafers. To be really effective, however, these systems must understand the nature of the effects the wafer is to receive, called a wafer specification. The creation of a proper specification for a wafer is challenging and complicated; it is best accomplished via a specification editor. This report describes the creation of such an editor and its use in the manufacturing process.

A process specification must meet many varied criteria. It must represent the specification in a way that makes it simple to edit. It must have access to other specifications and libraries of common processing steps. It must allow the specification of detailed events in a reusable way. It must be readable by programs (e.g., process simulators) and by people (e.g., operators, technicians, engineers, etc.) with different levels of experience. The specification must be connected with the results from previous processing runs, called logs, so that the specification may evolve intelligently. Such processing logs can encompass a wide variety of data and data formats, adding additional difficulties.

This paper describes the design and implementation of the BPFL Editor (BED) that was designed to solve these problems. It employs a graphical interface to facilitate use by people with different levels of training. The user interface metaphor is based on an outline processor, enabling its users to hide information with which they are not concerned. The editor has a built-in interface to a library of procedures, leveraging off procedural

abstraction and reuse. BED also employs hyperlinks to permit arbitrary connections from specifications to data and other specifications.

1.1 Outline of the Report

The report begins with a description of its background, describing the semiconductor fabrication environment, the Berkeley Process Flow Language, and prior work in the area. In the next chapter, the capabilities that BED provides are detailed. These capabilities are then demonstrated in an extended example in Chapter 4. Chapter 5 describes the implementation, the concerns that motivated it, and the problems encountered. The final chapter summarizes the conclusions drawn.

Chapter 2

Background

2.1 Semiconductor Fabrication Environment

This section describes the semiconductor manufacturing environment and traditional approaches to process specification editors.

2.1.1 Design and Production of Semiconductors

A *process engineer* designs an integrated circuit (IC) manufacturing process. Process engineers determine the sequence of chemical and physical processes that act on the silicon wafer to produce the desired product. The goal of IC manufacturing is to produce defect-free silicon wafers as cheaply as possible.

The processing of wafers is carried out by *agents* [39, 9]. An agent can be either a machine, a machine operator, or a combination. Some processing steps can be performed by *micro-controlled* machines that download equipment-specific programs (called *recipes*) and execute them. Commands in a recipe control gas flows, temperatures in reactor chambers, or the depositing of materials from a reservoir. Other actions must be carried out by operators by hand, such as creating loads for larger batch processing (e.g. furnace runs) and moving wafers between processing areas.

There are many different processing steps that can be carried out on the wafers in the course of fabrication. These include the addition of thin isolating or conducting layers, the removal of layers (selectively, as controlled by masks), the introduction of impurities, and so forth. Wafers are also tested during processing to determine corrections to ensure

process quality. In addition, events are logged to document exactly what happened to the wafers to assist in debugging the process and in scheduling machine maintenance. Operation sequences can become complex, because of timing constraints between steps, the necessity of keeping wafers clean and possible interference between steps. Semiconductor manufacturing is done in runs on batches of wafers, called wafer-sets, that typically contain 25–50 wafers.

The requirements on a process specification vary with the kind of factory in which it is run. In research and development factories (or laboratories), a processes specification undergoes rapid change to determine the optimal settings for a particular process step on a particular type (or piece) of equipment. Successive runs change the specification based on information from previous runs. Specialty factories produce low volume application-specific custom IC products (ASICs). These plants run many different process types, and they must reconfigure for each run. The object is to reduce the cost of reconfiguration by proper scheduling. In high volume commodity product factories, it is essential that a single process has a high yield and that it is scheduled properly with the other processes being manufactured to maximize the output of the factory. Process specifications must be flexible enough to work in all of these environments.

In addition to being used during fabrication, semiconductor processes are also simulated. Many different simulators have been built to model the manufacturing environment [19, 15, 27, 41]. These simulators are used in several different ways. One can use a simulator to assist in creating new process specifications. Simulation results aid in translating process specifications to work effectively on new equipment. They can also assist in determining what parameters are necessary to achieve desired changes in a wafer. It is important to be able to translate a process specification into a format suitable for a simulation program.

2.1.2 Traditional Process Specification

The traditional mechanism for specifying a process is the *run-sheet.* A run-sheet can either be on paper or stored on a computer. It describes each operation to be performed on the wafers. Typically the run-sheet only summarizes the steps by naming them and giving a few key parameters (e.g., a recipe name or temperature setting). The complete definition of an operation is found in a large manual, called the process manual, that includes complete listings of equipment, recipes, parameters, and expected measurements

taken during processing. The run-sheets tend to be erratic in their specificity, because their audience is technicians and operators. A run-sheet is passed around the factory with the wafer-set it describes, thus there is little danger of operators mixing up wafers from different processes. Operators make mistakes by occasionally misinterpreting the sheet's instructions and by mis-entering parameters and recipe identifiers. Test measurements are recorded on the run-sheet for use by other operators down the line.

The introduction of micro-controlled machines that can download recipes has changed the way processing is specified. These machines greatly simplify operator tasks [1]. The run-sheet remains, but the process manual has changed to contain descriptions of the machine recipes. These recipes are downloaded to the controller from a central processing database which reduces errors due to misprocessing, although the possibility of data-entry errors remains. Of course using recipes introduces a possible update problem, as the processing manual and machine-readable recipe can become desynchronized, potentially giving inconsistent directions.

The introduction of computers to the manufacturing floor has increased efficiency by allowing run-sheets and process manuals to be stored electronically. Computers permit better collection of process data and better management of the manual and recipes. This switch has also caused a change in the way processing engineers and operators think about run-sheets. The sheets are organized hierarchically, consisting of levels of operations to be performed, rather than simple lists of steps.

This change in viewpoint and the increase in electronic data collection combined to inspire the creation of Work-In-Progress (WIP) systems. WIP systems track the progress of a process through its sequence of operations, manage equipment scheduling, test data and so forth. Early WIP systems did not have equipment controllers, so operators performed the individual steps by hand (i.e., operators put the wafers into the oven, load the recipe floppy disk into the controller, and enter the commands to start processing). Current WIP systems can download and execute recipes, monitor processing, and collect data about the process (e.g. temperature readings, etc.) for some equipment. Some research is currently being done on more integrated control that includes feedback and feed-forward control [4]. These systems require more sophisticated micro-controllers and computer systems [17]. For such systems, it is even more important to have a unified specification system in order to

[1] Of course, there are problems with this simplification too, see for example [7].

avoid writing detailed (and possibly different) specifications for each software system used in the process.

2.2 Introduction to BPFL

The Berkeley Process Flow Language (BPFL) was developed because semiconductor processing is so complex. A process-flow is composed of steps and procedure calls connected by the language control statements. Procedures are sequences of operations that form an isolated processing step. A procedure can have arguments so that different calls execute different operations. A process module is often represented as a procedure. Procedures are collected together in libraries, that are version-controlled (as are process-flows).

All processing steps operate on wafers. In this report, we refer to a group of wafers that receive identical treatment as a *wafer-set.* In previous BPFL papers, these were referred to as lots. We use *lot* to describe the entire set of wafers for a run (i.e., the union of all of the wafer-sets). Thus a split-lot produces two different wafer-sets. During the processing, some steps may be carried out on all of the wafers (the lot), while others are performed only on specific subsets of wafers, such as test wafers (which form a wafer-set).

BPFL has a wide array of control statements. It supports `while` loops, `for` loops, and `if-then-else` commands. In addition, it has `rework` loops that repeatedly execute operations until a test is satisfied. Another control primitive is the `constrain` command that puts a requirement to be satisfied between two operations (such as "must happen within one hour of each other") and has an alarm to raise if the constraint is violated. The `viewcase` statement separates particular instructions for the different uses BPFL can be put to (such as statements for controlling the WIP system and statements sending input to simulators). A complete description of BPFL is given elsewhere, see [13, 40, 39].

2.3 Prior Work Using BPFL

Process-flows specified in BPFL were initially edited using text editors, because BPFL is a programming language. However, text editors were unsatisfactory because the syntax of the language caused too many problems for its intended users, who as a rule have no programming background.

The next attempt at an editor was a structured editor called SEPS [33]. SEPS was

built using the Cornell Program Synthesis Generator [28]. SEPS uses a database to store different versions of BPFL programs, and it has routines for browsing through the database and performing queries. It provides templates for code writing and performs syntax and semantic checking. However, users did not like SEPS because the editor was difficult to use, it had poor error handling and it was too "confining." Users wanted to be able to (temporarily) enter incorrect syntax and use commands from typical text editors, such as *emacs*.

To address these problems a graphical editor called VED [35] was created. This editor converted BPFL code into a graph of steps. The steps were displayed, and browsed using a mouse (See Figure 2.1). Many operations could be performed without touching the keyboard. This approach had some merit, but was abandoned. The graphical notation required too much space on the screen to display a small number of processing steps. In addition, the graph notation resulted in poor performance because of layout computations. Finally, the space constraints in the nodes obscured the details of a step, requiring additional operations to disclose step parameters and other additional information. Thus, VED was abandoned in progress, and BED was developed.

2.4 Other Specification Editors

While the above work has all been done using BPFL at UC Berkeley, other researchers have produced specification editors. This section briefly describes the functionality of comparable systems.

Wenstrand's Process Design Aid (PDA) [38, 37] is an editor for process specifications in Stanford's Manufacturing Knowledge System (MKS) [26]. MKS provides a knowledge-based framework for CIM; it supplies persistent objects, has good support for multiple materials and outputs of processes, and encompasses a large knowledge base. PDA, like BPFL, provides for multiple uses of process specification — both for manufacturing and simulation programs. It supports a hierarchical model of process specification, including reusable components (they are similar to BPFL's process-flows in their reusability and function, but are objects rather than procedures). Until recently it did not support complex control-flow operations, however work by Basile has addressed this issue [2, 1].

The Computer Aided Fabrication Environment (CAFE) [20] developed at MIT supports functions similar to BPFL used in conjunction with the WIP system [13]. Similar

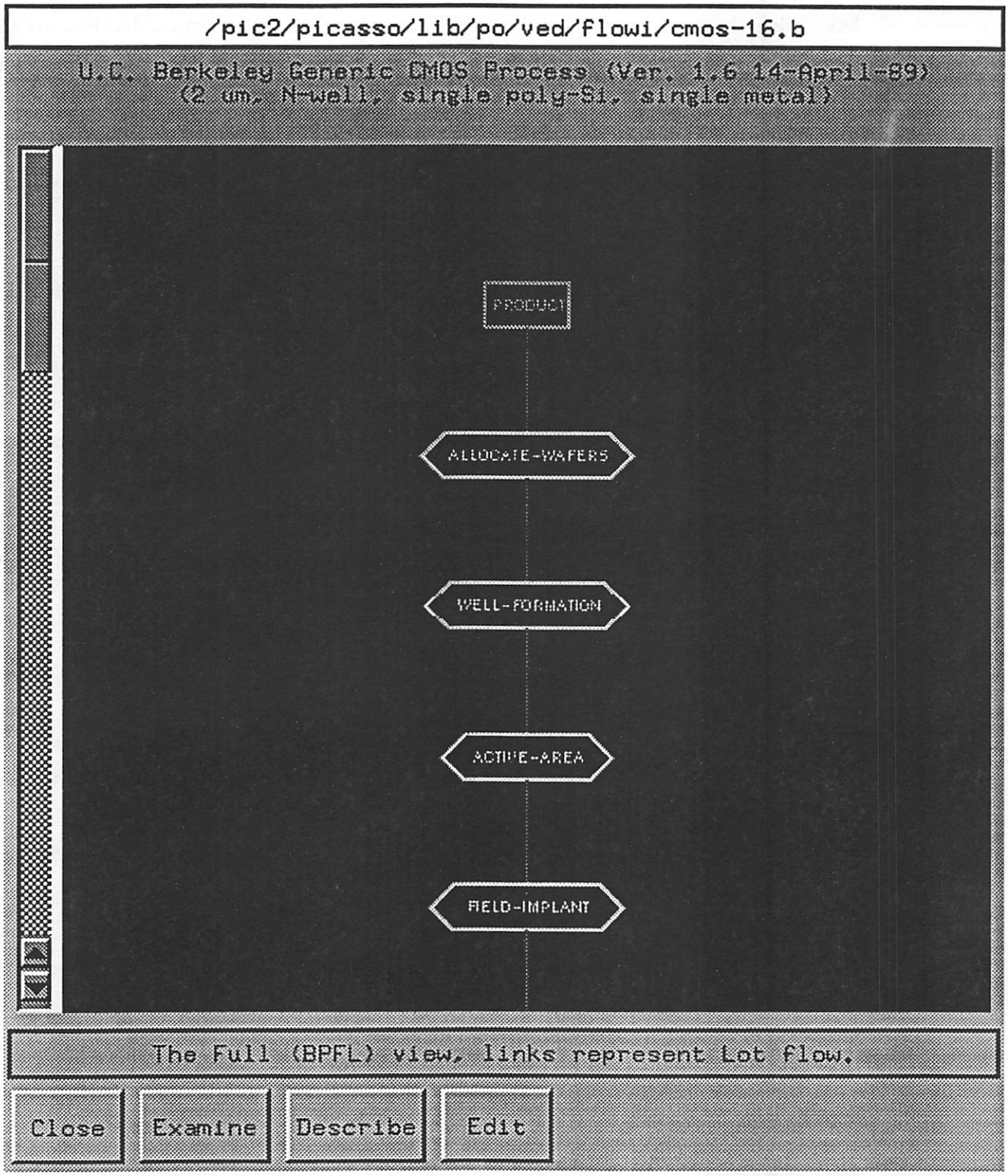

Figure 2.1: A VED edit window

to BED and the WIP system, CAFE uses a database to store processing information [14, 5]. CAFE's process editor is similar to early work on a structured editor for BPFL. It uses a programming-language style of process description, and edits the text representation of the process specification. Regular text editors are used, though forms-based panels are used to manipulate some stored data.

Recent work at Texas Instruments and Stanford has created the *Integrated Manufacturing/TCAD Specification System* [11]. This system integrates the different manufacturing systems in use in the Stanford Fab, encompassing recipes, electronic run-sheets, and simulators. It has a graphical interface and uses a library of modules and a persistent object store. The recipe integration and help system are better in this system than the analogous features in BED. However, its overall specification browser is weaker.

2.5 The Role of BED

BED further develops process specification editor technology. BED uses BPFL so that it has a very expressive, general, and extensible language for describing specifications. Through its use of the INGRES database, BED achieves the persistent information that PDA has through its objects. The graphical interface in BED is easier to use than CAFE's text approach. BED addresses the same issues as IM/TCAD system, but from a different viewpoint, and with an extended definition of a process specification.

Chapter 3

System Capabilities

This chapter briefly describes the facilities available in BED. They are demonstrated in detail in the extended example in the next chapter.

BPFL gains much over other specification languages because it is a full programming language rather than a sequence of simulator instructions, a list of machine settings, or a collection of nested objects. Anecdotal evidence has shown that a significant amount of time in a fab is spent handling exceptional conditions, something at which programming languages are good [10]. In BPFL, it is trivial to specify alternate operations to execute in cases of a failure, or automatic repair steps to perform. In addition general calculation and data management operations are easy to do.

BPFL takes advantage of experience gained in the fab via the processing libraries. Procedures that perform related functions are gathered into a library. This makes them easy to update together and provides a mechanism for the reuse of common operations. Procedures are common abstractions which may be used over and over, often to different effects through argument specification.

The version-control system allows a user to modify specifications for private runs. This feature is essential in a research environment. It also alleviates any worry that technologies will be "lost" as equipment is changed in the fab. All old specifications are saved and can be used years after their archival. Storing them in a database allows general queries to be used to locate particular specifications.

The graphical representation of the processing steps in BED makes the relationships between the steps much clearer than many other systems (See Figure 4.1), especially the important "contains" relationship. This form of presentation eases comprehension of

new specifications. BED also supports hypermedia to link together different types of data. Using links, specifications can contain other types of data, such as images of SEM pictures of a wafer surface, video clips demonstrating equipment operations, graphical displays of wafer profiles, and dynamic analysis of logged data, in addition to text notes and descriptions. This auxiliary data can be accessed by a general hypermedia system. The system allows arbitrary connection of data to particular specifications. One immediate use is the connection between processing steps and the log entries of operators from previous times they have run wafers using the specification. This connection gives easy access to comments and notations about possible problems or successful results. The system even has a graph-generating subsystem to enhance data visualization and provide "dynamic data" that can be displayed in different ways by its examiner.

Users can alter the process specification presentation to clarify the BPFL statements. Customizations range from changes in font size and color use to the invention of new presentation elements. This customization allows for local improvements over the initial design decisions in BED and extensions to BPFL.

From BED, one can connect to the different programs that make up the UC Berkeley CIM system:

1. **Faults** is a program for maintaining information about Fab equipment maintenance and repairs of equipment and other systems described in [21, 22].
2. **WIP** is the the Work-In-Progress System described in [13, 12].
3. **Facility** is a facility monitoring tool described in [34].
4. **BCAM** is a framework of equipment control and diagnostic applications, see [4].

BED also has a help system, so new users can get on-line assistance for their tasks. The information displayed by BED can be changed to prevent users from being overwhelmed by the complexity of the processes they are examining. For example, the user can scroll around the specification, so that it need not all fit on the screen at once, see Figure 4.18. Users can also iconify pieces that are uninteresting. BED supports multiple windows to facilitate sweeping changes. Which wafer-sets are displayed may be changed, and users can look up documentation for procedures when they desire to see it.

Chapter 4

Extended Example

This chapter demonstrates the capabilities of BED through an extended example. The example concentrates on the new features BED provides:

- The iconification of steps not desired by the user.
- The integration of the library system and versioning system in a database.
- The use of hypermedia to expand the capabilities of the specification language.
- The button strip of common actions to simplify editing operations.
- The online help system.

4.1 Starting up BED

The first step is running BED. The `run-bed` command is executed to start up the editor. At first, BED loads the specification last edited. In this case, the main window (shown in Figure 4.1) shows the CMOS-16 process. The specification appears in an edit window of which there may be several in use at one time.

4.2 The Edit Window

To meaningfully go through an example, one must first understand the various areas in the Edit Window and their function.

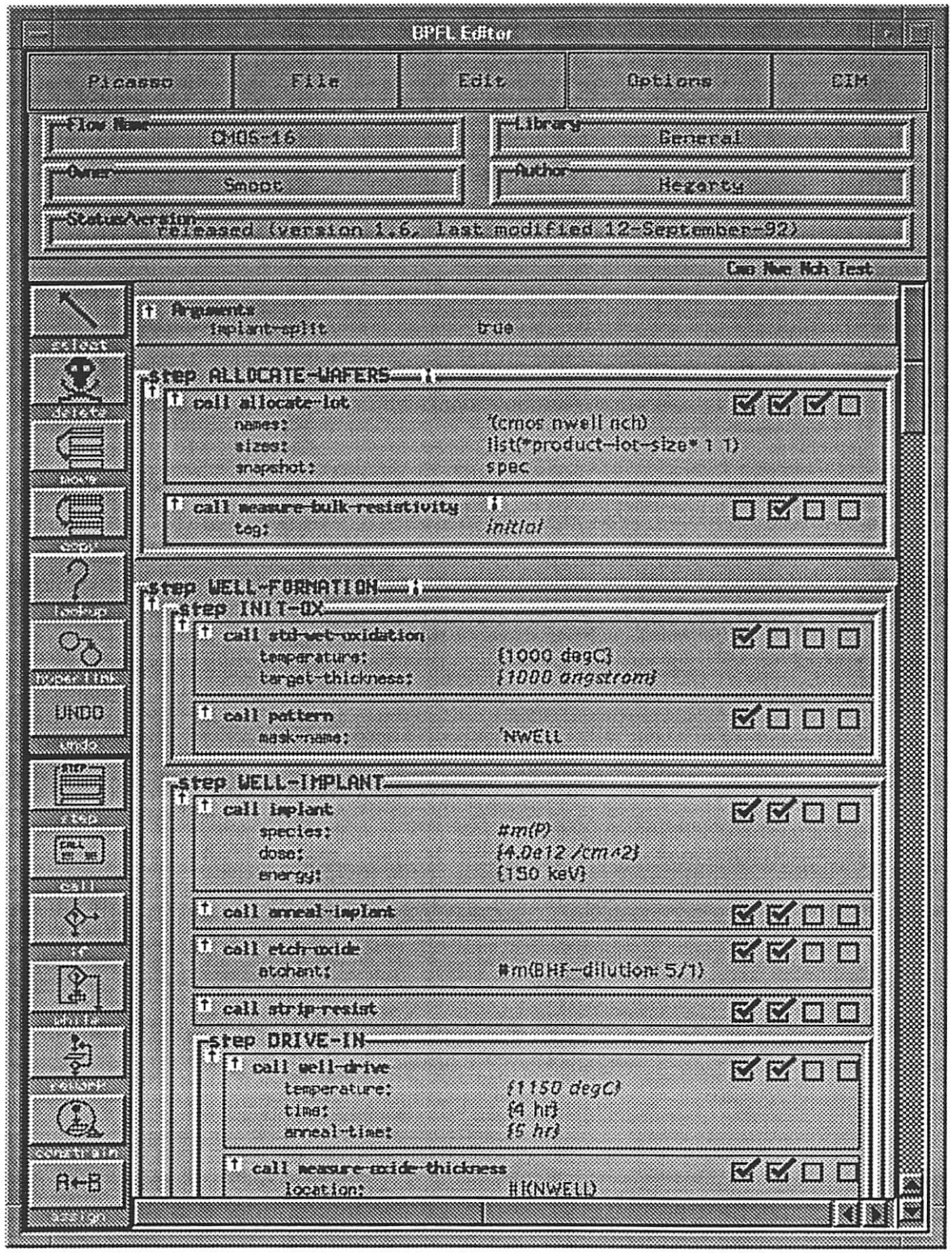

Figure 4.1: A BED edit window

4.2.1 Parts of the Edit Window: The Menu bar

At the top of the BED window is a menu bar. Currently BED has five different menus from which to select. The first menu, `Picasso`, has only the entry `Quit` which is used to finish the editing session.

The second menu is the `File` menu. It has the commands that relate to file operations (in reality these are database operations). These commands are detailed in Table 4.1.

The third menu, `Edit`, contains familiar editing operations: `cut`, `paste`, `delete`, `move`, and `copy`. These perform the same operations as the side-buttons, discussed later in this Section. However it also has some additional features, listed in Table 4.2

The fourth menu, `Options`, has several functions, listed in Table 4.3, to change the interface or the view of the process specification.

The final menu, `CIM`, provides interfaces with the other programs in the Berkeley CIM system, see Table 4.4.

4.2.2 Parts of the Edit Window: The Status Area

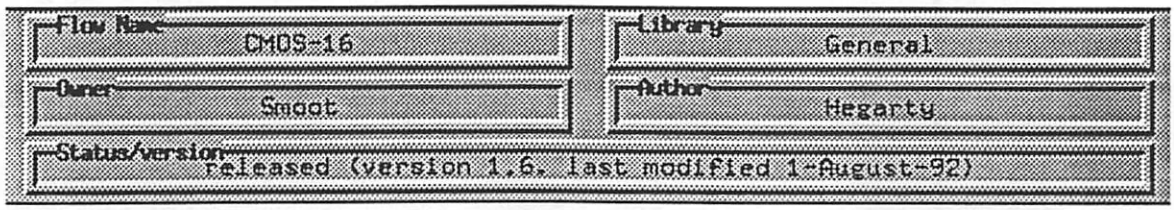

Figure 4.2: The status area.

Below the menu bar is an area that displays general information about the process being edited (See Figure 4.2). In the upper left corner, it shows the name of the process-flow being edited. To its right is the name of the process library in which the procedure is defined (possibly empty for top-level process-flows). Below these two fields are the Owner and Author fields. The Owner field names the person holding a lock on the process (as part of the version system), usually the current user. The Author denotes the original author of the code. The final line gives the Status/Version, displaying the status of the process-flow (i.e. private, testing, or released) along with its version number and last modification date.

Table 4.1: The operations provided in the `File` pull-down menu.

Open: Examine another flow pops up a dialog box (Figure 4.11) to select which flow to examine. It creates a new edit window with that process flow loaded.

Load: Examine A different flow also pops up a selection dialog box (Figure 4.11), but replaces the current flow in the current edit window with the selected process-flow.

Close: Close this window destroys the current edit window (if it is the only edit window, this is the same as selecting `Quit` from the `Picasso` menu).

New: Create new flow opens another edit window to permit the creation of a new process-flow.

Save flow installs the current process-flow as a new version of the process-flow. (It pops up a dialog box to permit renaming, new major versions, and selecting the status of the process-flow version.)

Revert to previous version undoes all changes since the last load by reloading the process-flow.

Change libraries included pops up a dialog box (Figure 4.17) to add requirements on which libraries (and which versions of the library) are required by the current process-flow.

Load previous flow performance turns on the display of hyperlinks to the runs of the process-flow. It pops up a dialog box to choose which runs are to be accessed (Figure 4.21).

Edit Flow Documentation pops up a panel to change the documentation provided with the process-flow (or library).

Documentation (Help) pops up a panel that displays a help file for using BED (Figure 4.18).

Print flow diagram currently does nothing. It is intended to print a version of the process-flow on an appropriate printed. The print-representation of process-flows has not been finalized.

Table 4.2: The additional operations in the third pull-down menu, `Edit`.

Make Hyper Link allows the user to create a link from a particular point in the process-flow code to another object.

Edit in Text Form pops up a window with the current process-flow in its text form (BPFL).

Edit User Dialogs pops up a dialog box to select a user-dialog, and then a window in which the user-dialog is displayed (allowing the BED user to change what text is displayed, stored, etc.).

Table 4.3: The `Options` pull-down menu's choices.

Select Wafer-Sets to display This allows the user to add or remove wafer-sets from the on-screen display (See Section 4.3).

Examine run-log is a facility to look at the results of running the process-flow in the Fab. It pops up a menu to choose which run of the process to examine (Figure 4.21), and then a menu of each log entry for that run (Figure 4.26), from which the user can examine particular log entries (Figures 4.23 and 4.24).

Change View (BPFL, Fab, etc) modifies how BPFL's `viewcase` instructions are interpreted for display.

Color Bindings allows the user to set how colors are mapped in the interface.

Table 4.4: The `CIM` menu entries.

Faults A program for maintaining information about Fab equipment maintenance and repairs.

WIP Begin a session with the Work-In-Progress System.

Facility A facility monitoring tool.

BCAM An equipment control and diagnostics tool.

The documentation (comments) for the process-flow can be reached through a menu-option, as described in this Section. All of these fields are display-only, and may be turned off by the user.

4.2.3 Parts of the Edit Window: The Wafer-set Descriptors

Immediately below the Status Area are the Wafer-set Descriptors (See Figures 4.1 and 4.3). Each operation in a process-flow acts upon the specified wafer-sets. The names of these wafer-sets are given at the top of edit window. The names actually form the top of a wafer-set selection column of the process code, as described below. Clicking on a wafer-set name will deactivate the display of information about that wafer-set.

4.2.4 Parts of the Edit Window: The Process-Flow Display

The edit window is the most important part of the interface — the part that actually displays a process specification. The BPFL code that controls the process is in the main body of the edit window. It has both vertical and horizontal scrollbars so that it is not constrained in size. At the top of this area are the arguments to the process-flow. In the example (Figure 4.3), there is one argument named `implant-split`, and it is set to `true`. If there were more arguments, they would be listed below this line. As the arguments are expected to be supplied to the process-flow when it is executed, the displayed value is merely the default. To change the default value, the user simply clicks on it, which pops up a dialog box (similar to the one shown in Figure 4.4), and changes the value through the dialog box.

Following the arguments is the first `Step` of the flow. In this case, it is the `Allocate-wafers` step. As can be seen in Figure 4.3, the step contains two procedure-calls: `allocate-lot` and `measure-bulk-resistivity`. These procedure calls have arguments. `allocate-lot` has three arguments: `names`, `sizes`, and `snapshot`. They are listed below the line with the procedure name on it. For each argument there is a value to its right. When the arguments are supplied, they are printed in regular type (as are all of the arguments to `allocate-lot`). However, when the default value supplied by the procedure being called is used, the printed value is italicized (as is the argument to `measure-bulk-resistivity`). To change the argument value, the user clicks on it, which creates a dialog box (Figure 4.4). This dialog box permits the entry of the value, or the selection of the default value, which

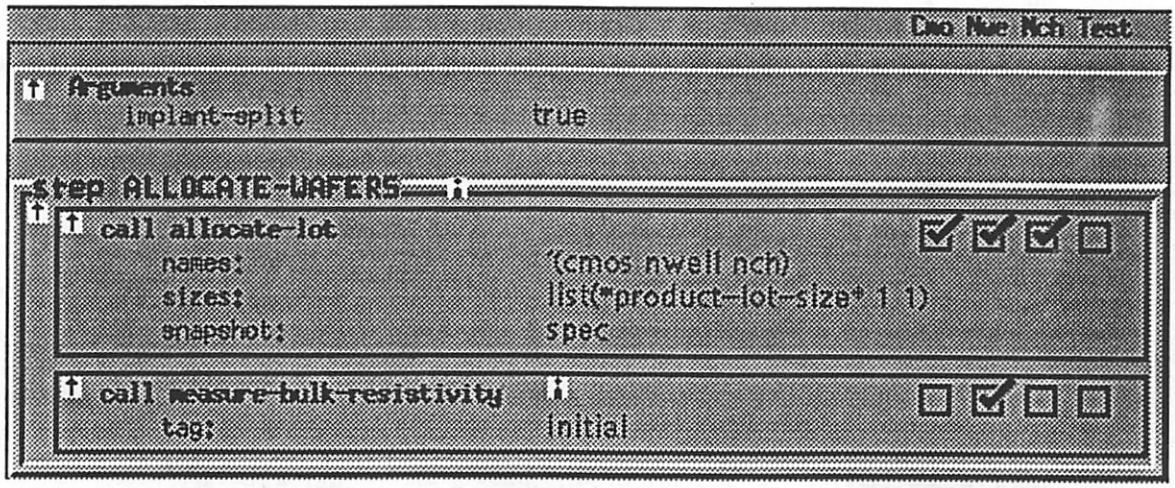

Figure 4.3: The topmost section of a process specification.

is done by clicking on the **use default** button.

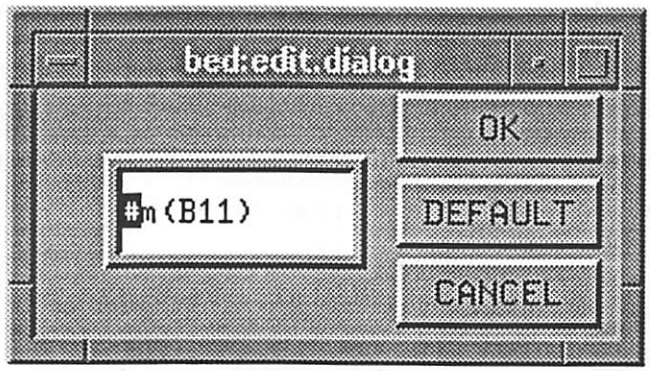

Figure 4.4: Dialog box for entering new argument values.

As mentioned above, all processes take place on entire wafer-sets. For example, the `measure-bulk-resistivity` procedure operates only on the Nwell wafer-set, as indicated by the checkboxes to the right of the process-name. Of the four boxes only the one in the second column is checked. If the box in the column corresponding to a wafer-set name is shown "checked," the process acts on that wafer-set (thus `allocate-lot` acts on three wafer-sets: Cmos, Nwell, and Nch). To change whether or not a wafer-set is acted upon requires a simple button click on the box, selecting or de-selecting it.

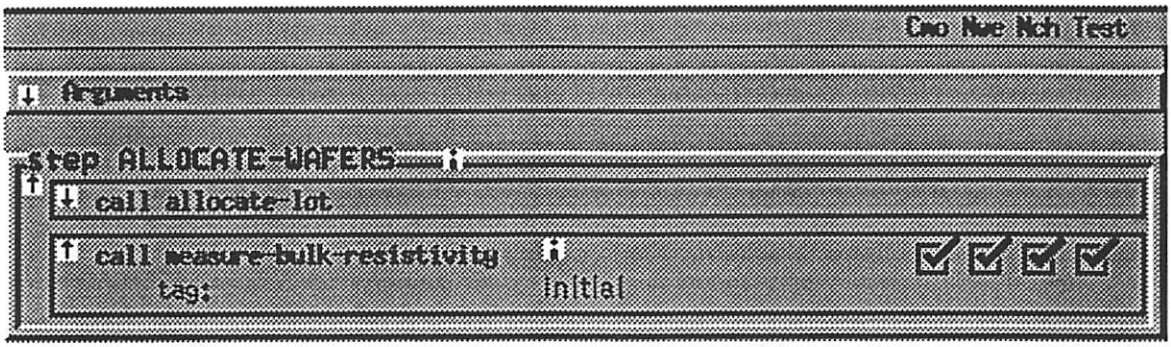

Figure 4.5: The topmost section of a process specification after two iconifications and a change in wafer-sets from that in Figure 4.8.

On the left of each box (i.e., procedure-call, step description, or argument list) there is an arrow. The arrow indicates the version of the particular item that is being shown; an arrow pointing up indicates a "long" version. To conserve screen real-estate, the user may use the mouse to click on an up-arrow to close the display of the item, in other words, the step or operation is iconified. An example of a display with the arguments and `allocate-lot` call iconified, along with the changes to the wafer-sets operated on by `measure-bulk-resistivity` changed, is in Figure 4.5. This required a total of five mouse clicks. De-iconification is accomplished by clicking on down-pointing arrows.

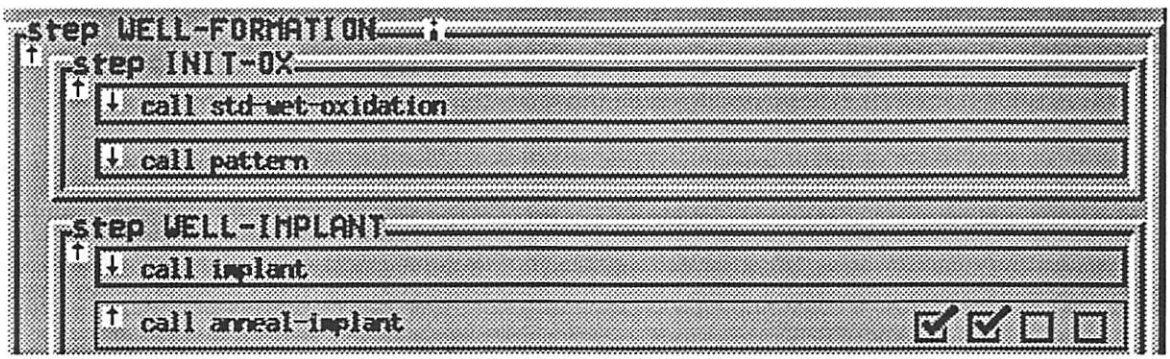

Figure 4.6: Nested steps with iconified procedure-calls.

In BPFL, steps can be nested, as Figure 4.6 demonstrates by examining the next step in the CMOS process-flow (most displayed procedure-calls are iconified). In addition

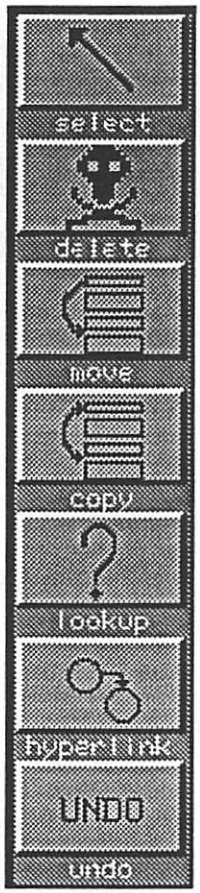

Figure 4.7: The Edit Buttons on the Edit Window.

Figure 4.8: The creation operations in the edit window.

to steps and procedure-calls, BPFL features while loops, if statements, constraints, rework-loops, etc. Each operator can have a customized display form.

The remaining feature of the edit window is the button labeled "i" next to the step names, these buttons denote a hyperlink which are discussed in Section 4.8.

4.2.5 Parts of the Edit Window: The Editing Operations

Commonly used editing operations are provided in a button strip on the left side of the window (as well as in the pull-down menus). The buttons are shown in Figures 4.7 and 4.8. Clicking on a button will change the cursor to the same shape as that which the

button displays, to make clear the current editing operation or mode (except for undo which happens immediately). After selecting an operation, clicking with the mouse middle button will execute the specified action. Thus, for example, entering delete mode and clicking on the `measure-bulk-resistivity` procedure call in Figure 4.5 will delete it; the results of this action are shown in Figure 4.9.

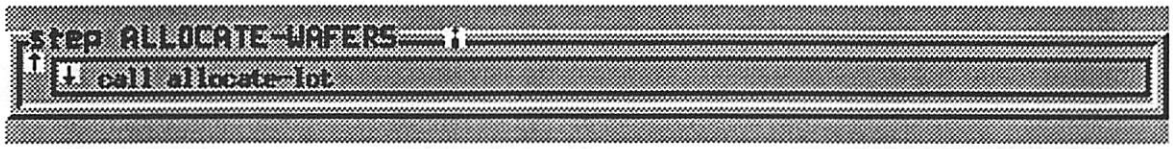

Figure 4.9: Top portion of Process flow with `measure-bulk-resistivity` deleted.

Clicking the middle mouse button while in `move` mode selects the specified steps (and any substeps) and moves them to another location. `Copy` works similarly, except that it leaves a copy of the initial steps in the original place. `Undo` does not actually act as a mode; it merely undoes the last action. Clicking on a procedure-call while in `lookup` mode brings up a new window that contains the definition of the called procedure. For example, clicking on the `implant` call in Figure 4.6 will open the window shown in Figure 4.10.

There is a different column of buttons to create new BPFL structures, see Figure 4.8. Clicking on one of these buttons will change the cursor to display the object being created. Clicking in the process-flow description area will insert the structure there. This action may cause a dialog box to pop up prompting for information about the structure being created.

4.2.6 Loading Another Routine to Edit

Pulling down the `File` menu and selecting `Open: Examine another flow` loads another procedure to edit. This command pops up a panel to select the library from which to load a specification (see Figure 4.11). After clicking in the first column on a library to use, the second column is filled with the choices of versions for the library. After selecting a version, the procedures within the library are listed in the third column. A fully selected routine is shown in Figure 4.12. Clicking on the **OK** button loads that procedure.

Loading another process-flow to edit, in our case `wet-oxidation`, causes a new edit window to appear (as displayed in Figure 4.13). This window is in all respects identical

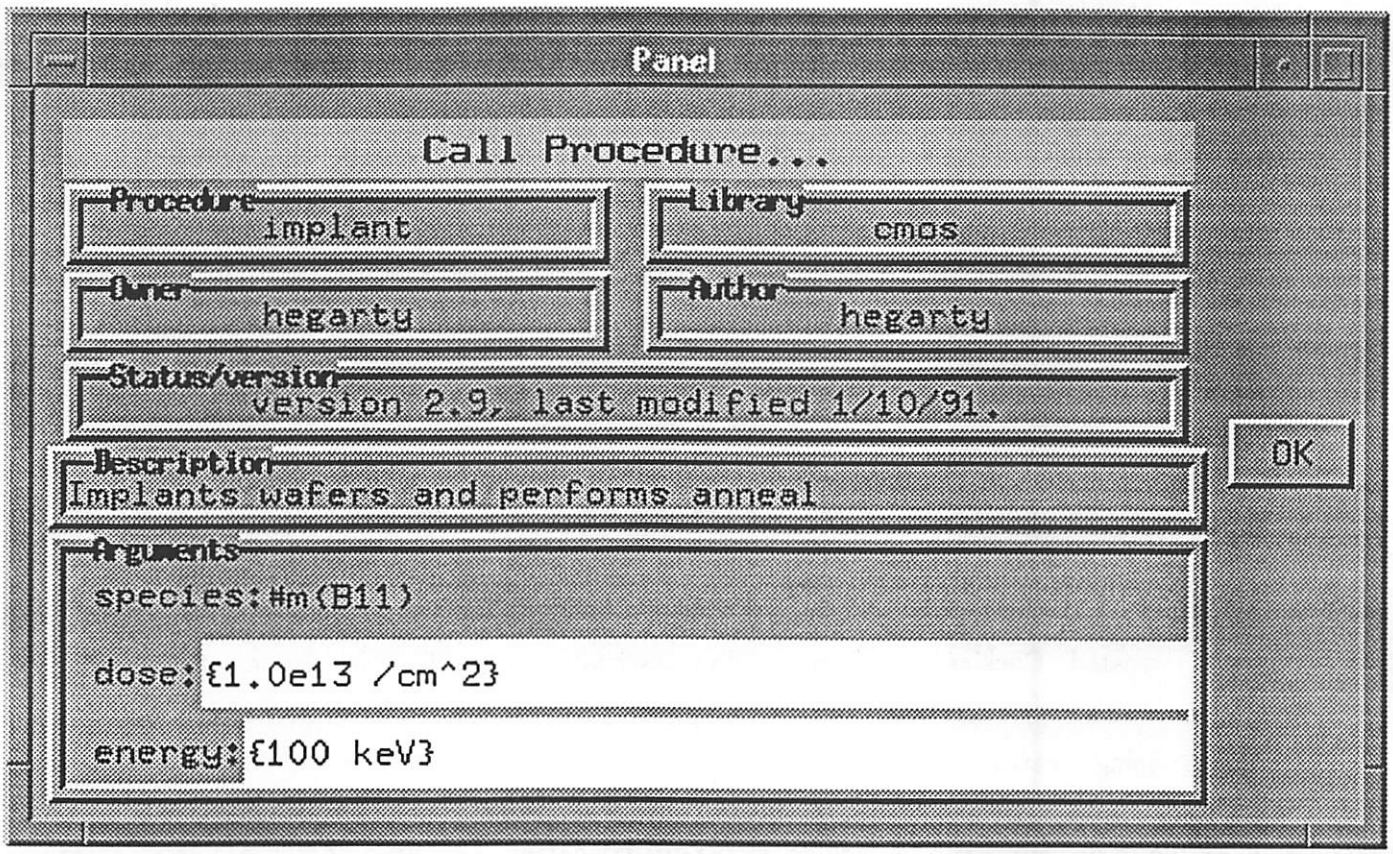

Figure 4.10: Information on the `implant` procedure.

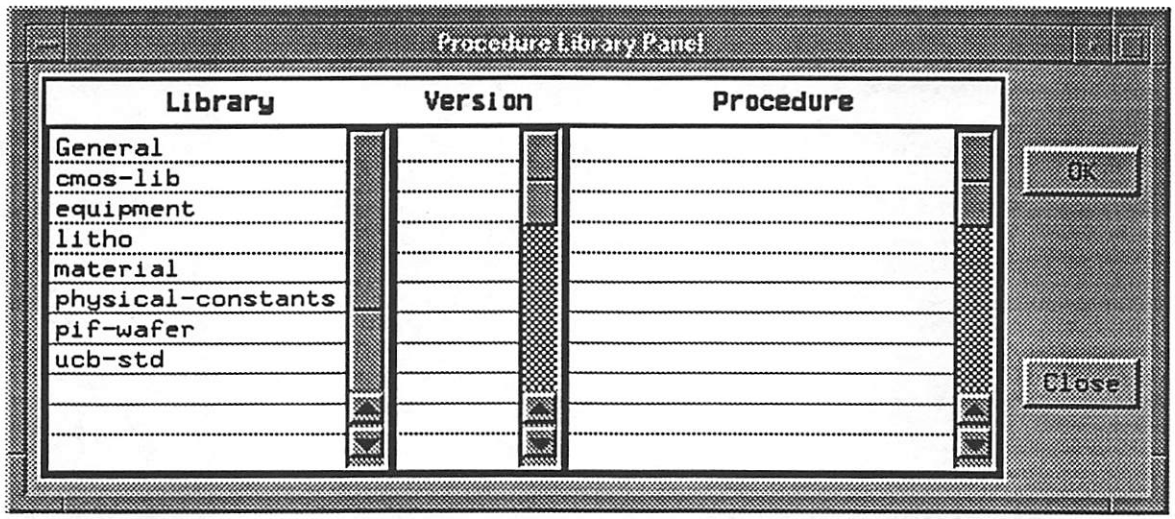

Figure 4.11: Initial Load Panel.

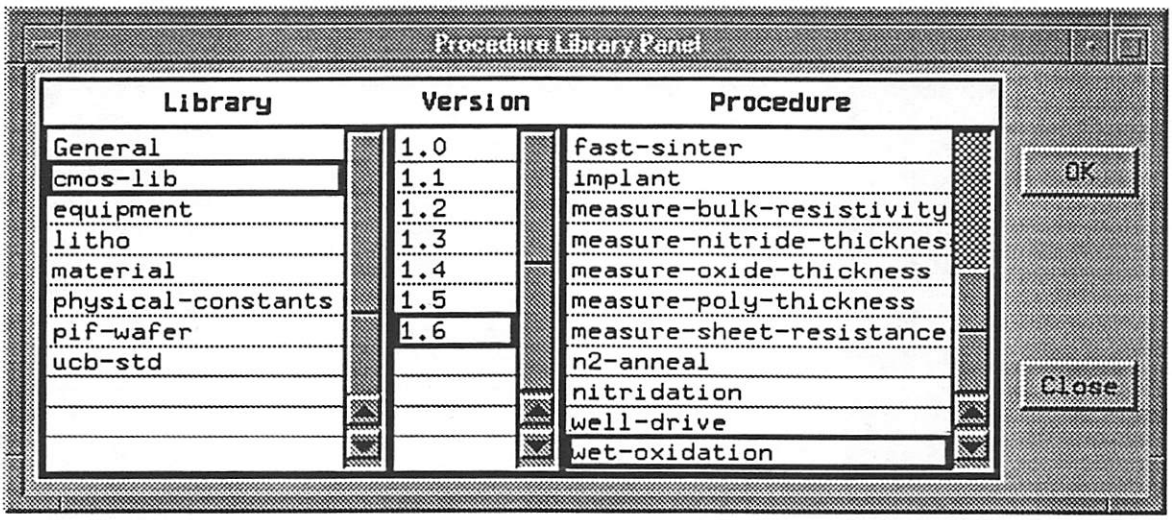

Figure 4.12: The BED load-procedure panel, where the process-flow `wet-oxidation` has been selected (shown highlighted in the lower right).

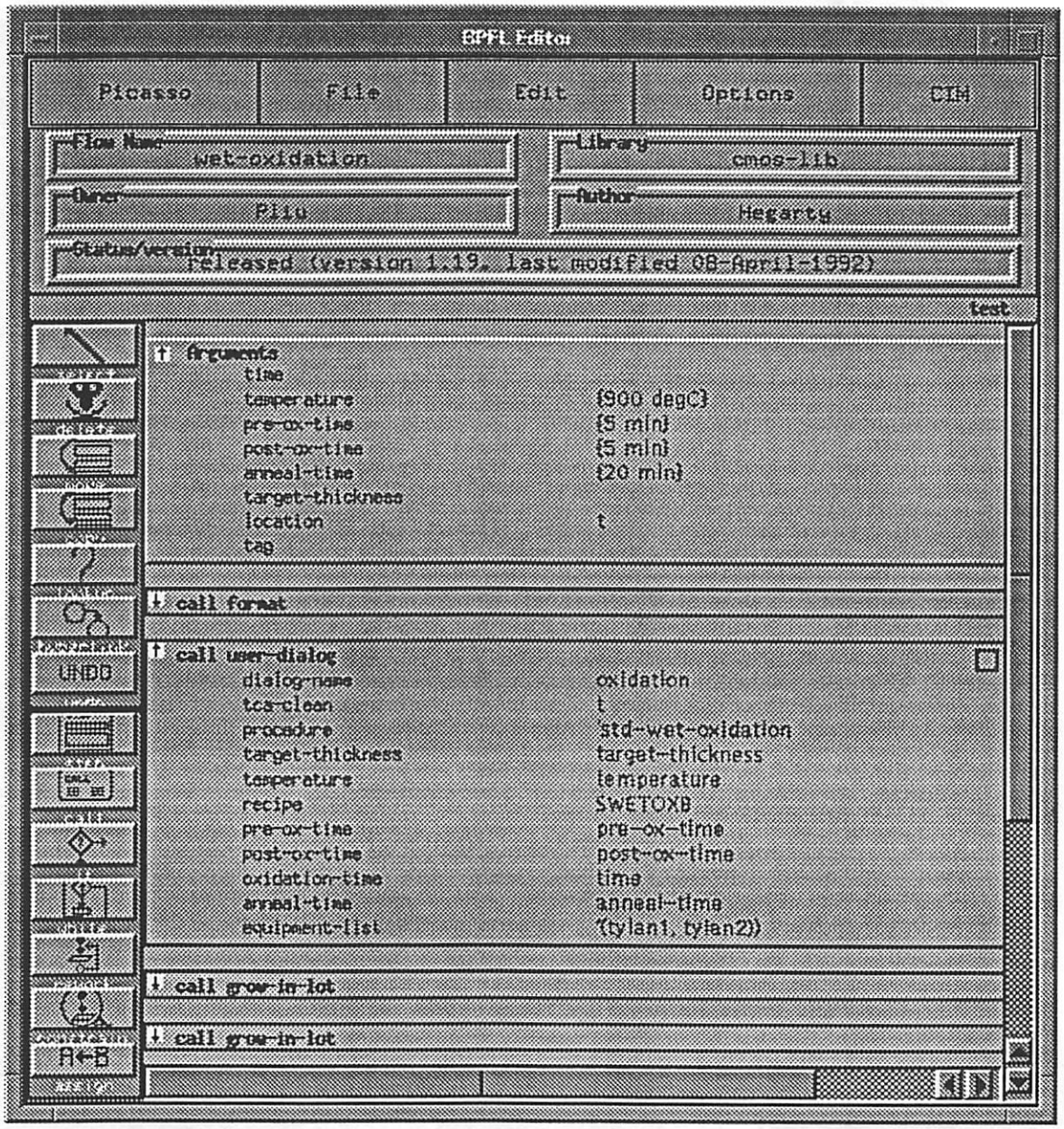

Figure 4.13: A second edit window, editing the `wet-oxidation` flow.

to our original example, so we shall return to it.

4.3 Adding a New Wafer-set

From time to time, users may wish to change which wafer-sets are displayed, or even to create a new wafer-set to use. To change the wafer-sets displayed, the user selects the `Select Wafer-set to Display` menu entry, and uses a dialog box to specify the changes, see Figure 4.14. (Of course, the user can also turn off the display of wafer-sets by clicking at the top of their column.) Clicking on the checkboxes in this dialog window controls whether the wafer-set is to be shown. Entering a name in the "new lot" text entry box will create it as a wafer-set and add it to the list of wafer-sets.

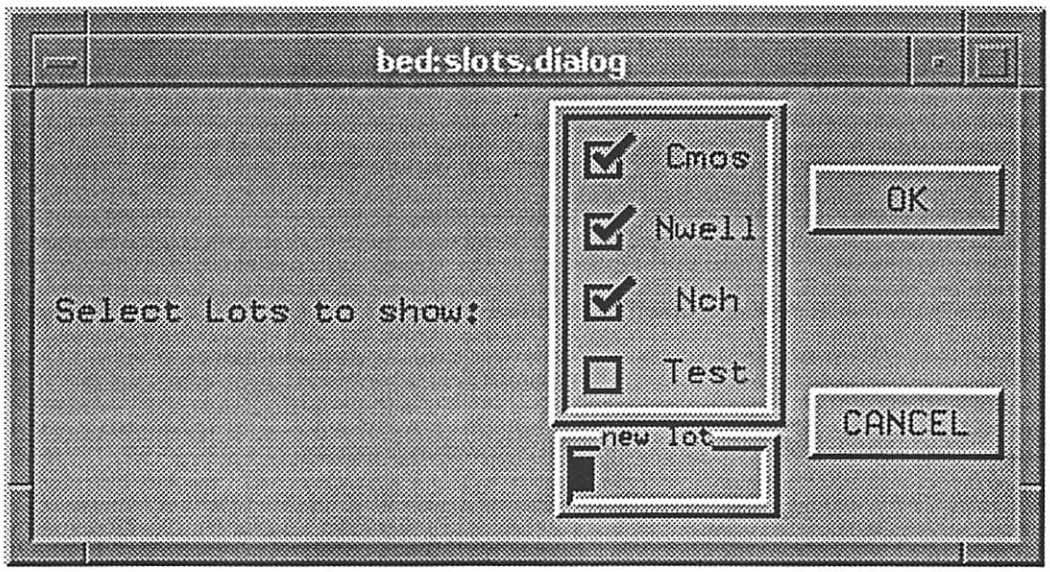

Figure 4.14: Panel for the addition, removal, and creation of wafer-sets.

4.4 Adding a new procedure call

While modifying a specification, it is often necessary to insert an operation between two others. Clicking on the type of operation (such as the procedure-call button from Figure 4.8) and then clicking between the two operations will insert a new operation. These clicks create a dialog box to query for the procedure name, as shown in Figure 4.15. If the

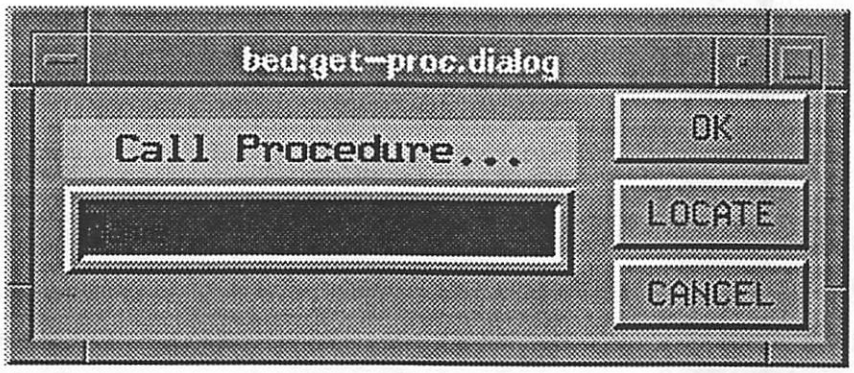

Figure 4.15: Initial selection box for a procedure call.

name is sufficiently unique, the user can type it. (The appropriate version for the procedure is worked out from which libraries are currently being used.) If the user wants to use the browser to locate the proper version, clicking on the `Locate` button will bring up the the procedure library window shown in Figures 4.11 and 4.12. As with all dialogs, clicking on `Cancel` ceases the operation requested. Selecting a procedure (e.g., `wet-oxidation`) will bring up another panel. This panel (Figure 4.16) describes the procedure at its top. Its bottom section gives the default and required arguments. Required arguments have a white background and must be supplied, default arguments have a gray background.

4.5 Selecting Libraries to Reference

When BPFL code is interpreted it needs to specify from which libraries to draw its procedures. The libraries are changed via the `Change Libraries Included`" menu entry. The dialog box used is shown in Figure 4.17. The dialog box will list all of the libraries currently being referenced along with their version numbers. The version use may be one of:

latest uses the most recent released version of the library.

query uses the most recent version of the library at the time of the creation of the process-flow specification. If a newer version is in existence at the time the flow is run, it generates an alarm to query the operator for which version to use.

static locks a particular library version (released or unreleased) down. It will use this version unless it is explicitly changed within the process-flow.

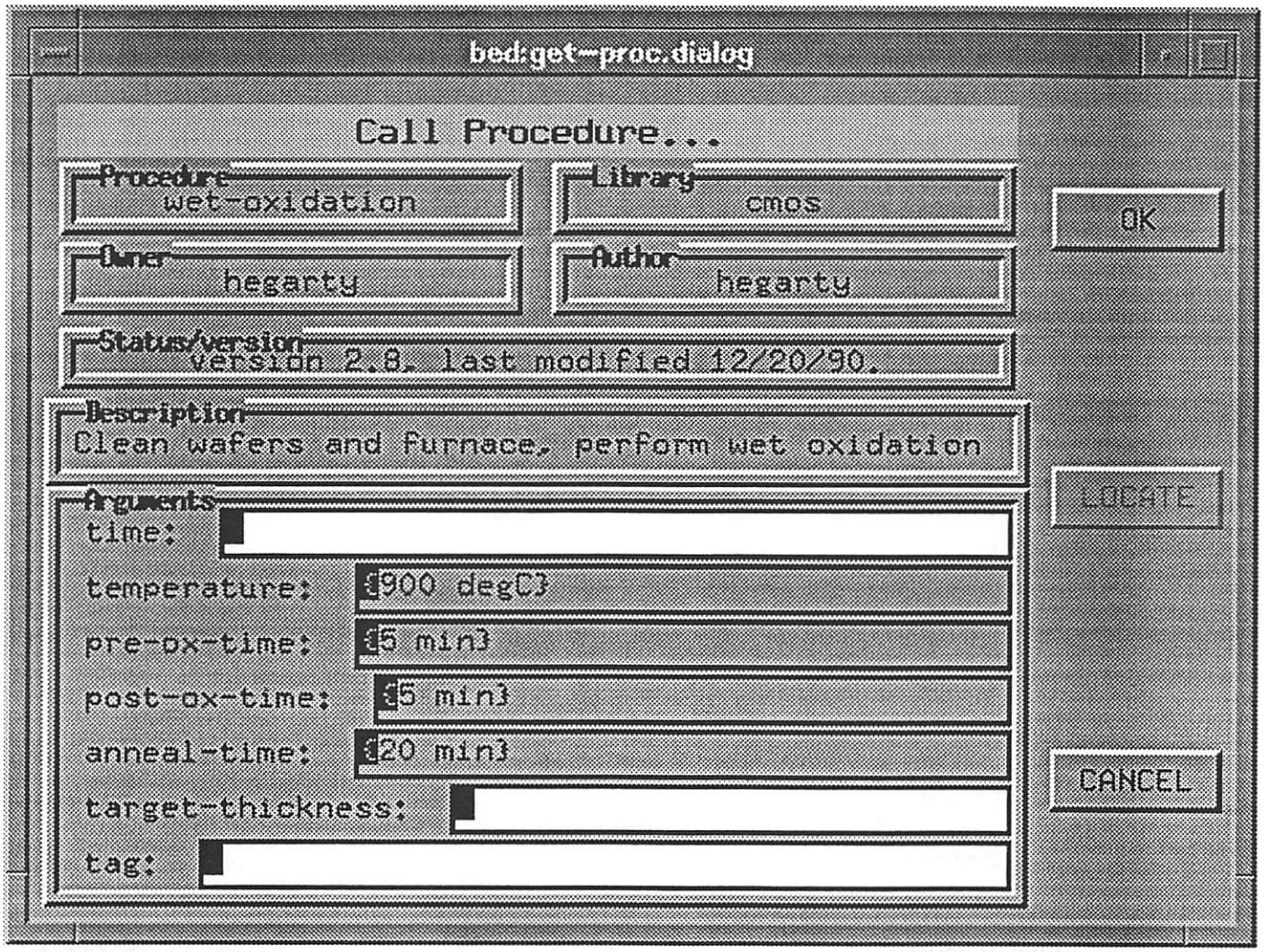

Figure 4.16: Adding a call to the `wet-oxidation` procedure.

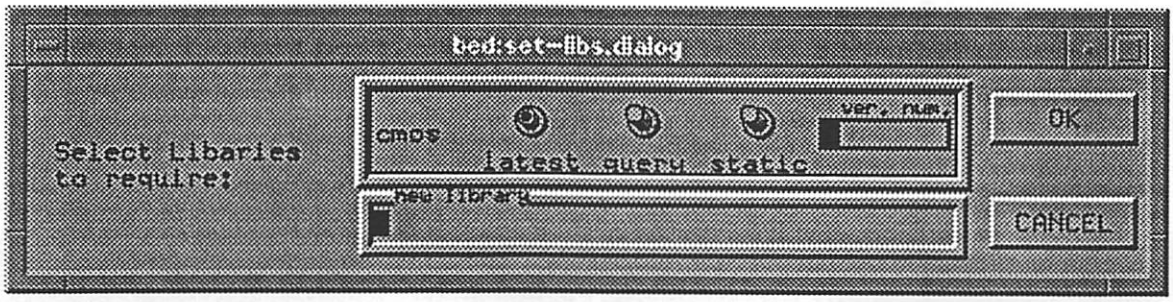

Figure 4.17: Dialog box to change libraries used by a process-flow.

The "new library" field allows the user to add a library dependency. Adding a new library will create a box for it (like the one for the cmos library shown in the Figure) and allow its version to be specified.

4.6 Using the Help system.

A sample help panel is shown in Figure 4.18. It may be examined by using the scrollbar, or via text searches. It is accessed from the `File` pull-down menu. The current version of help does not support hyperlinks or multi-part displays (i.e. a composite document with drawings, tables, images, etc.).

4.7 User Dialogs

From time to time a process specified with BPFL will prompt the operator with questions about test results or other information. These interactions are specified by executing a user-dialog command. When executed, this command displays a form for the operator to fill out. The form is printed on the operator's terminal. The forms for these dialogs are stored in the database and can be edited from BED. A sample for computing results from measuring oxide thickness with the nanospec is shown in Figure 4.19.

4.8 Hypermedia in BED

This Section describes the hypermedia facilities used in BED.

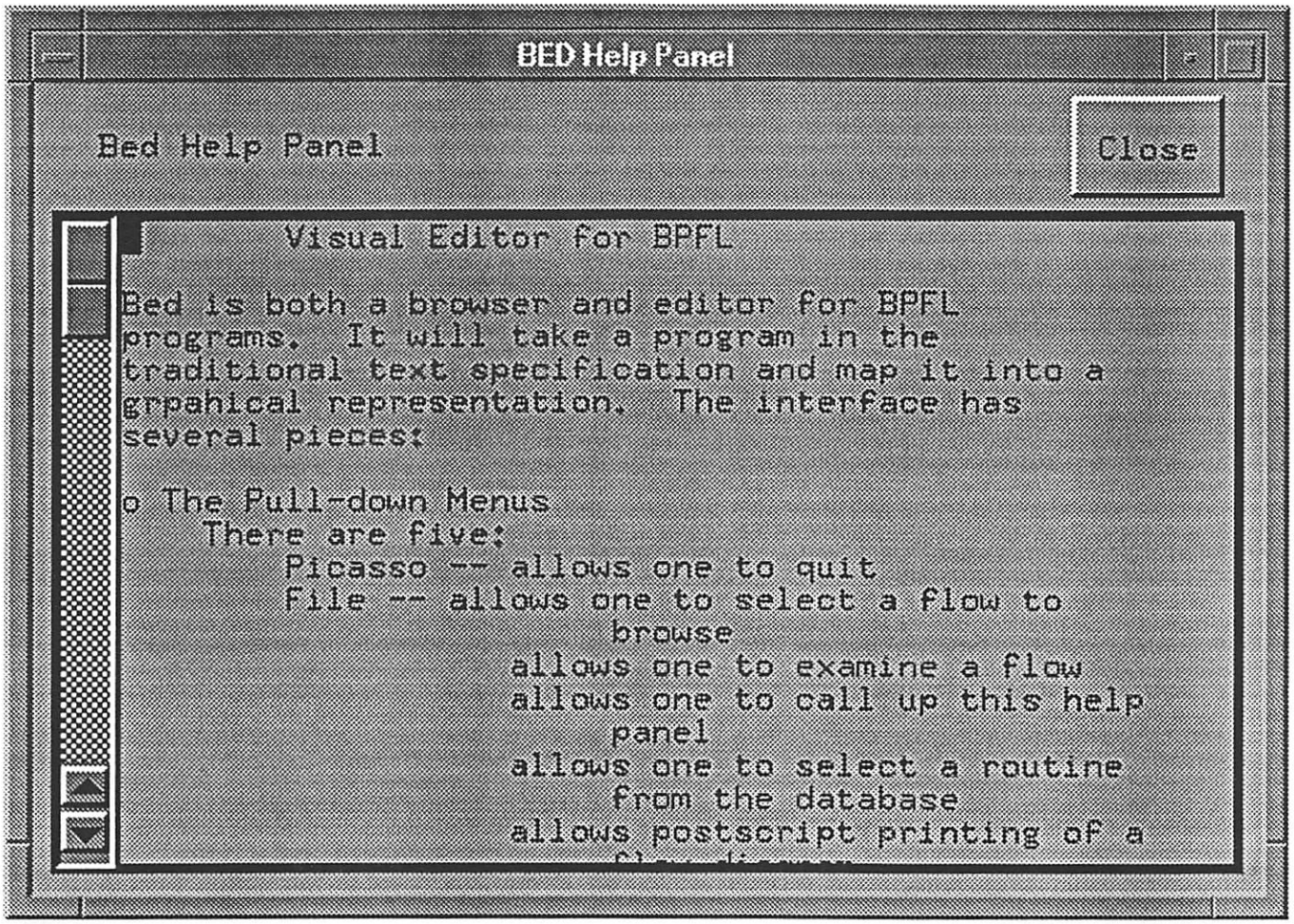

Figure 4.18: A sample help panel.

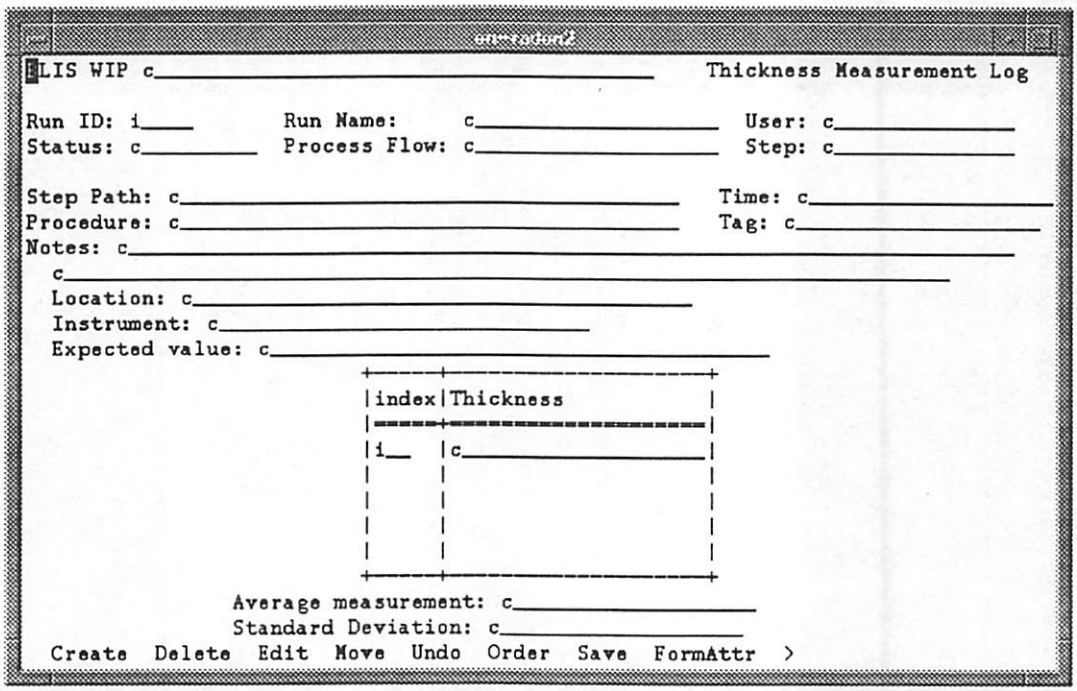

Figure 4.19: The forms for editing user dialogs.

4.8.1 Introduction

Hyperlinks are arbitrary connections between different parts of a document. They were initially discussed by Bush as "Non-sequential writing," see [8]. The term "hypertext" was invented by Ted Nelson [23]. Hyperlinks basically provide a non-linear form of reading where instead of reading from one page to the next in a set order, one may jump about the text. In most computer applications (which make hypertext practical), hypertext systems are defined in terms of *nodes* and *links*. Links connect the source section to the destination, which is called a node. Thus, one begins reading the start node, and may follow links from it to other nodes (which may in turn have further links). Links are arbitrary in the sense that they do not form a part of the regular structure of the computer interface. One trivial example is the help provided by many software packages, where an operator who is (linearly) entering data may click on a help button and read an informative description about the task (a trivially simple hypertext system). BED makes use of *hypermedia* which is a generalization of hypertext. The document can contain the usual sort of hypertext features, but in addition may contain non-text data forms, such as video sequences, audio recordings, dynamically updated graphs, etc. One place BED uses hypermedia is by integrating video clips into the process specification. Hypermedia opens up a new range of possibilities for many applications. For example, fabs might add digitized audio of operator comments, or integrate automatic comparisons of videotaped wafers with the wafers being manufactured.

4.8.2 Using Hypermedia in BED

Hypermedia is used extensively in BED. In fact, there is no comment command in the BED version of BPFL because commenting is subsumed by the hypermedia system. Instead of putting a comment next to a statement, one creates a hyperlink to the comment. Then the comment can have arbitrary text and other media about the statement, and perhaps even hyperlinks to other parts of the specification. For example the **i** button next to `measure-bulk-resistivity` in Figure 4.1 indicates a comment. Clicking on it displays the window in Figure 4.20, which is a hyperlink. The presence of these links in indicated by the **i**'s next to the name of an operation (See Figure 4.1 or 4.22). Clicking on the **i** will pop up a menu that lists the different hyperlinks attached to this operation. The menu contains a short name for the link and an indication of the type of the node at the link's end.

Hyperlinks are also used to integrate log entries with the specification. One

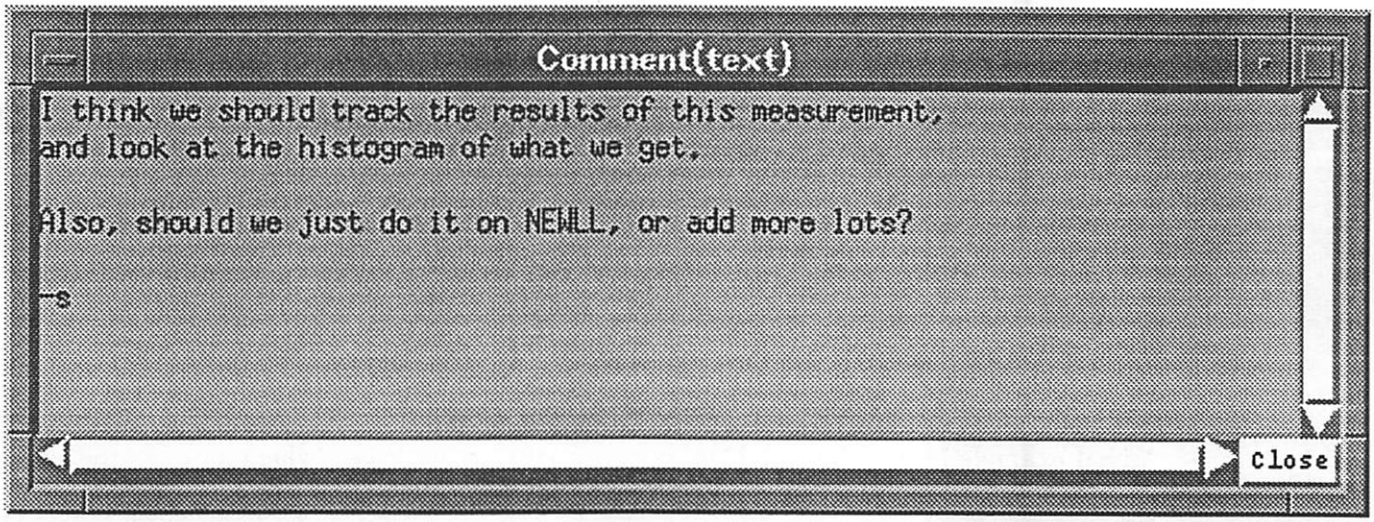

Figure 4.20: A typical comment hyperlink.

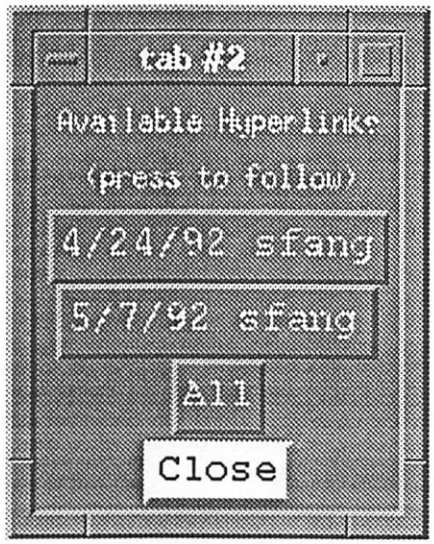

Figure 4.21: Selection of which previous run log to access.

BED mode displays the links to previous runs of the specification shown mixed in with the processing operations. This is done by selecting a menu entry (`Load previous flow performance`), and selecting which previous process runs to include links to. A sample window used to choose a previous run is shown in Figure 4.21. In this window, the date and operator of previous runs are given, along with the option to choose them all. If a link is selected, the buttons to access them (links) will show up on the edit screen (they appear as tiny **R**'s, see Figure 4.22).

Clicking on an **R** button will bring up the log entry for that operation, as illustrated in Figures 4.23 and 4.24. This appears as another window (usually in a corner of the screen). This new window can be referenced independently from the edit window. The log window has a scroll bar to examine all of its contents. The log entry may have hyperlinks or active data within it. Active data can take the form of video clips, audio, or graphs. A graph is the presentation in a new window of a table of data, such as in Figure 4.25 . The graph was summoned by clicking on the **graph data** button in Figure 4.24. It is also possible to browse a log via its table of contents, as displayed in Figure 4.26.

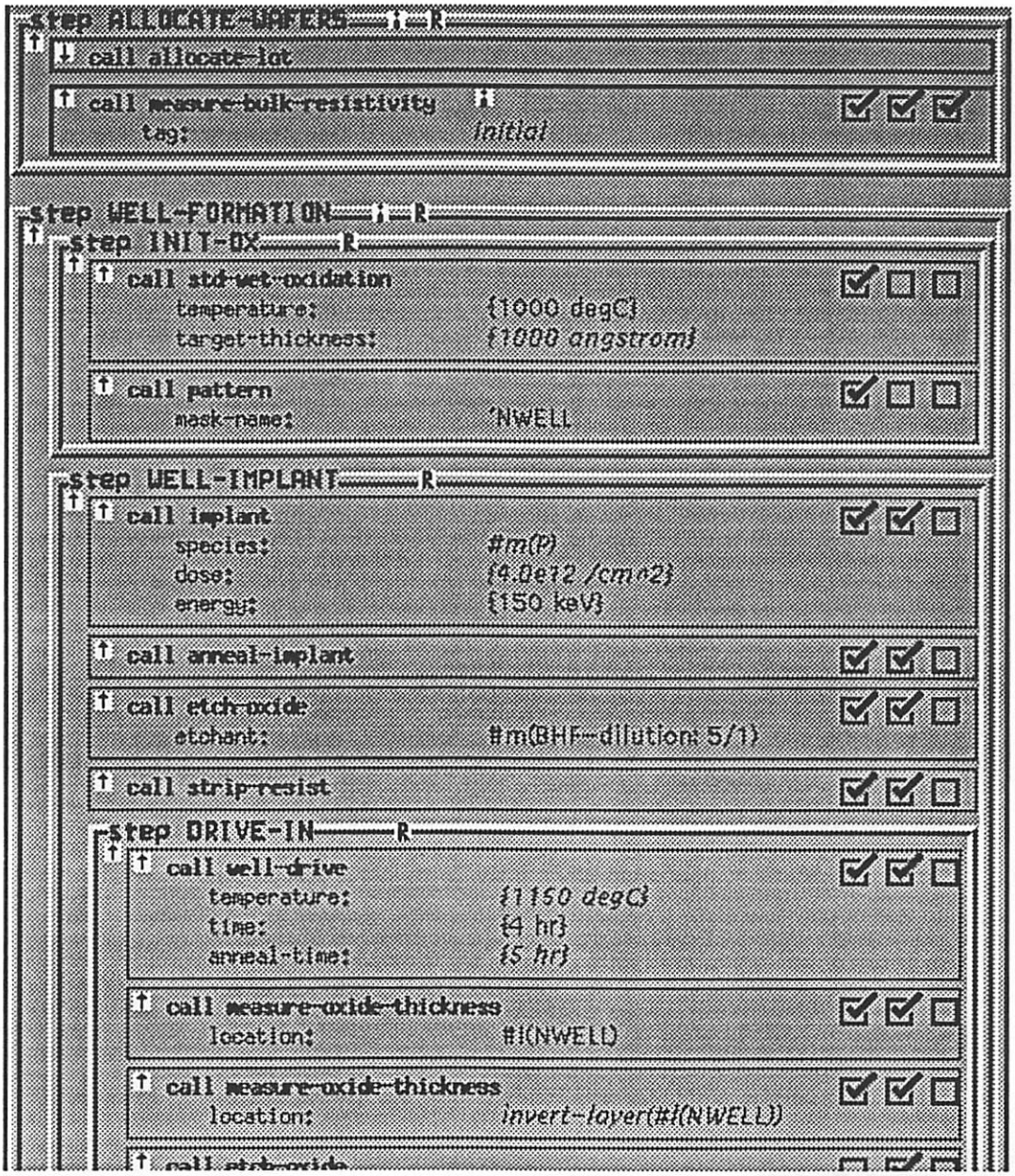

Figure 4.22: A BPFL code example with hyperlinks to previous performance nodes shown.

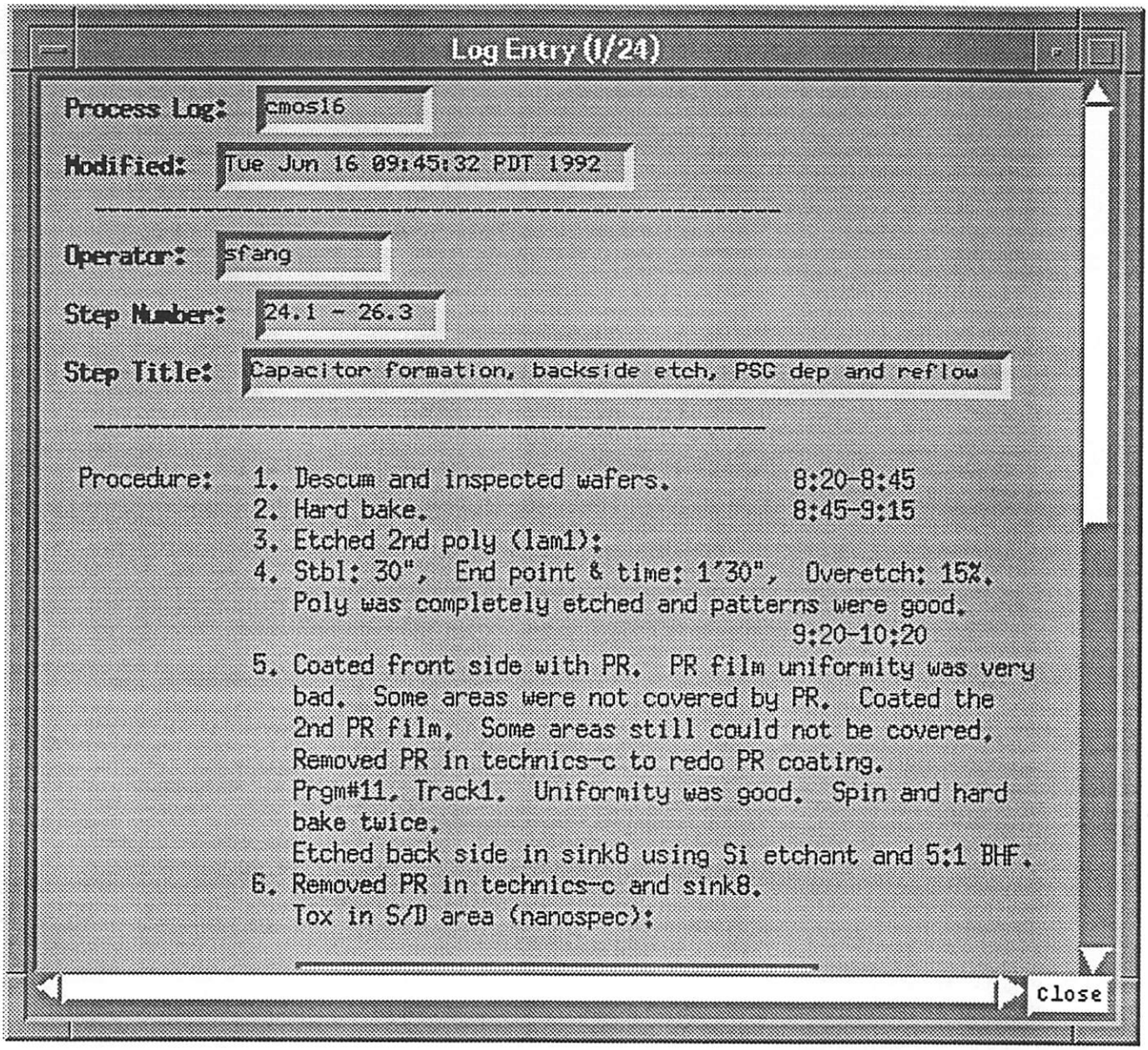

Figure 4.23: A log entry for the formation step of the CMOS-16 process.

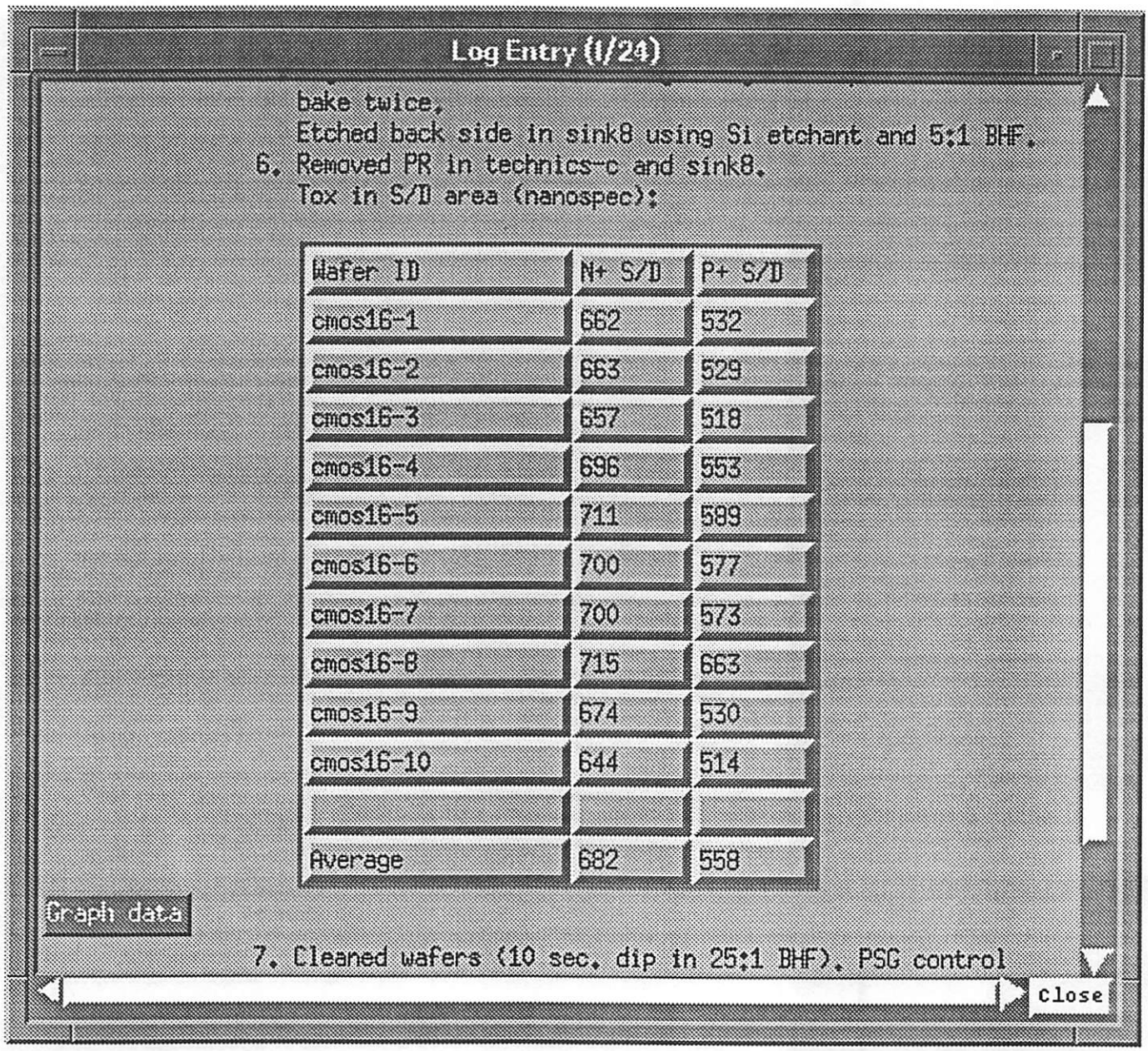

Figure 4.24: A log entry for the formation step of the CMOS-16 process (scrolled).

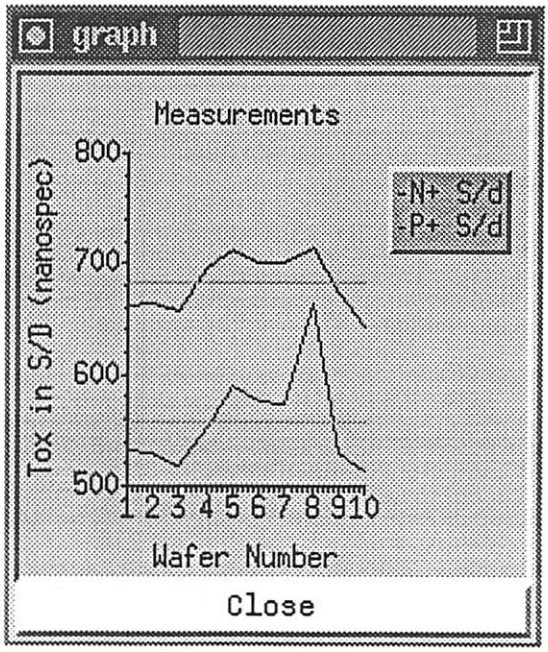

Figure 4.25: A graph of the S/D Tox data (the data from Figure 4.28).

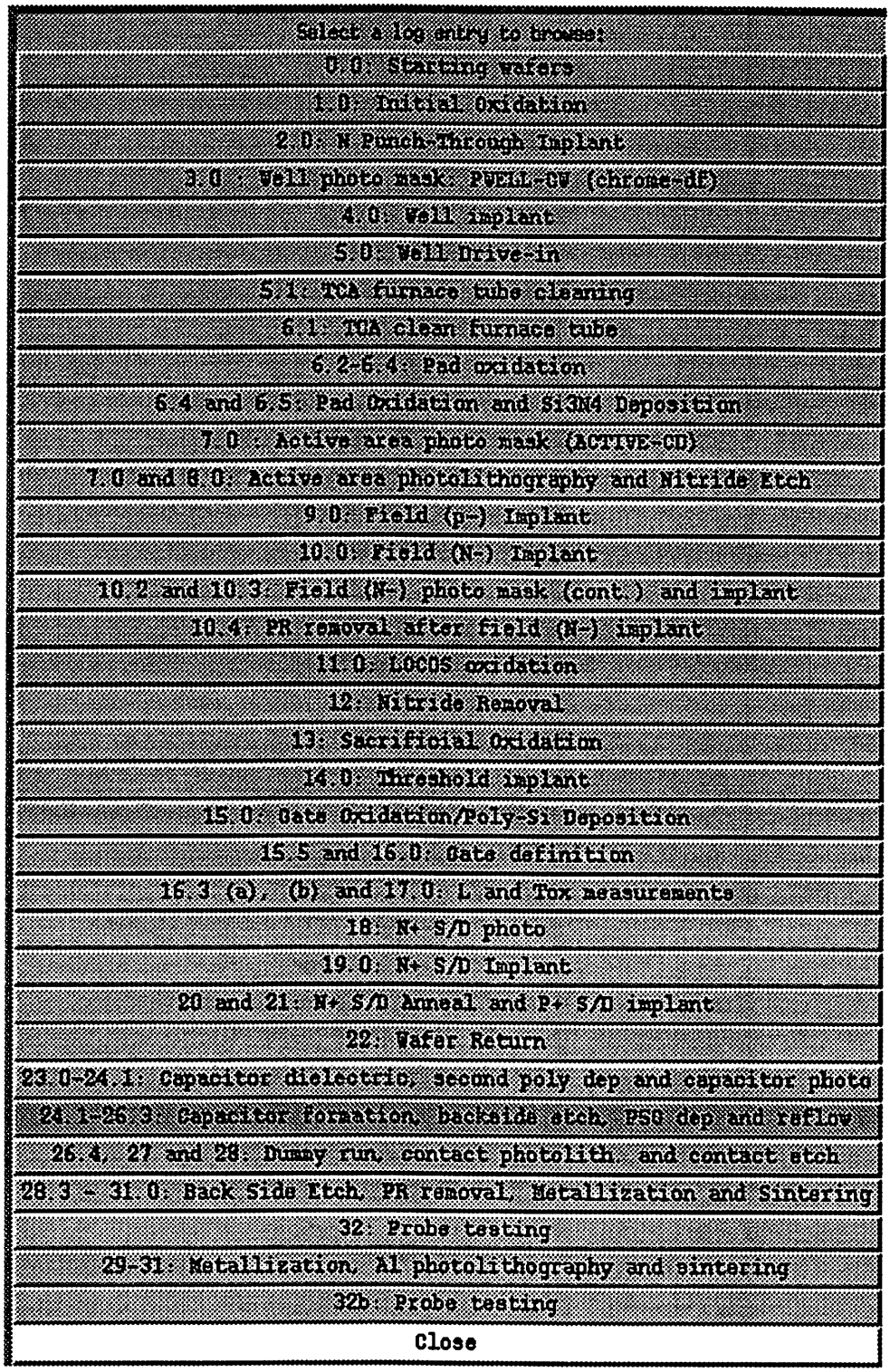

Figure 4.26: These are the log descriptions. Clicking on a line will summon that particular log entry into another window.

Chapter 5

Implementation

The BED system is composed of many different modules. This chapter describes the modules and their implementation. The motivation for each part of the design is given. Finally the problems with the implementation are discussed.

BED is primarily implemented in the Common Lisp Object System (CLOS) [36, 18]. It uses the Picasso Application Toolkit (version 2.1) [31, 29] for most of the interface. Picasso is an X Window System [32] toolkit in CLOS. It provides high-level abstractions for interface development. In total, BED contains seven thousand lines of CLOS and Picasso code. Approximately half of this code has to do with the specialized outline widget that implements the edit window's body. A custom widget was used as the Picasso implementation, using nested collection widgets, was very slow (e.g., it required half minute window repaints). The rest of the code implements the connections to the database, specialized translation routines, the wafer-set widget, the various window formats BED uses, and the editing functions. The help system is a collection of text files, loaded into Picasso browsers.

Some of the interface is implemented in Tcl/Tk [25, 24]. A Tk hypertext widget allows arbitrary Tk widgets to be embedded in a scrolling field [16]. BED capitalizes upon this widget to implement the hypermedia and hyperlink features. Only two hundred lines of Tcl code were written to provide this functionality. However, the log entries are technically in Tcl/Tk also, having been converted into mixed text and Tcl commands. As these can be automatically generated, they are not included in the total, but can represent a significant amount of "code;" the log entries for a single run of CMOS-16 make up three thousand lines.

5.1 Motivations

The combination of Picasso and CLOS was chosen for several reasons. First, Picasso provides high-level abstractions for user interface development, abstracting away many of the details of an X application. In particular, Picasso's framework frees the programmer from the details of window creation and destruction, hides many style decisions, automatically creates buttons and links them to functions, etc. Second, CLOS and Lisp provide high-level programming language tools. Lisp languages tend to be good at string manipulation, which simplifies the implementation of editors. Finally, the Picasso source was available should any changes be needed to support BED. While no new features were added, its availability made it possible to fix the toolkit rather than have to write application code to avoid bugs.

Tcl/Tk was chosen for the implementation of the hypermedia system because it provides much of the functionality very cheaply. A previous Picasso project [3] implemented a hypertext/hypermedia system, and was known to be slow. This motivated a search for another way to implement that part of BED. Tcl/Tk features a hypermedia widget as an extension [16]. In addition, Tcl/Tk programs tend to be very small, so they could easily be used alongside Picasso. Finally, other research at Berkeley [30] is being done integrating video with Tcl/Tk, allowing the simple addition of video to BED.

5.2 Problems

The fundamental problem with the implementation of BED is that it is too large and slow. The executing image is approximately 28 megabytes. In addition, X11 and INGRES must be running, although INGRES may be run remotely, so the actual local load may be small. Even running on a machine with 40 megabytes of real memory, BED is still too slow. Recent experiments to improve space/time performance of large Lisp programs have been disappointing [6].

An additional problem with using Lisp for an interface package is that periodic garbage collection causes annoying pauses in the display. This bothers some users even more than the general slowness. A better integration of graphics package and Lisp environment would ameliorate this problem. The bottom line is that Lisp is an excellent prototyping environment, but a poor environment for developing production applications.

The Picasso interface to Tcl/Tk is primitive. All non-interface computation is on the Lisp side, and it simply tells the Tcl code what to display. However the development of a function in Picasso similar to the Tcl *send* primitive would have allowed a more equal division of work and a more interactive system.

The primary difficulty experienced was with speed. While some optimizations increased the speed of some BED operations, BED still has longer delays than users like (such as its eight minute startup time). It is believed that the speed problems stem from using Picasso. Apparently, the generality Picasso provides comes with a high recalculation cost whenever the interface changes, a high start-up cost (though this was known before implementation), and a high window-creation cost. Part of this time is consumed by the slow creation time for CLOS objects and classes. A new implementation entirely in Tcl/tk has been begun; initial results show a dramatic speed increase.

An additional problem was with the database design. BPFL process-flows are stored either in files that are referenced by the database or in the database itself. When reading text files, the editor has to parse the files on every load to determine how they should be displayed. This parsing is somewhat slow and inefficient. Unfortunately, the database retrieval of the files is also slow. However, having the specifications in the database enables a broader range of queries to be done on their text.

A continuing problem is the use of screen real estate. Most X applications are developed to be one of many running at a time on a user's screen. However it is difficult to fit a reasonable amount of process specification along with the editing tools in a small window. Thus, the average editor window consumes half a screen. This problem has spurred several innovations in BED: the iconified procedures, the scrollbars, and the separation of hyperlinks from the text. No magic solution for this problem is evident. Possible solutions range from using a virtual desktop, to speeding up window creation/deletion times so much that it is natural for the user to do so frequently.

As mentioned in Chapter 3, the system is general. However, some of the flexibility is confusing to the user. For example, the different "look" of language elements can be changed. However, this requires one to understand a complex calling mechanism and write CLOS/Picasso code to display an element in a different way. This exposure is undesirable; a friendlier mechanism is needed. Unfortunately doing this is nearly identical to writing a user-interface generator, which is a large and complex tool. It is desirable to find some middle ground that could make this possible without all the complexities of a full-blown interface

generator. Similarly the different layout customizations cannot be easily set because they are hidden in the menu hierarchy. It would be better if BED recognized the changes made in its presentation style and automatically stored them (such customizations range from colors and fonts to automatic iconification).

Chapter 6

Conclusions

BED makes three important contributions to specification editing. First, it expands the definition of a specification to include diverse but useful related data such as video clips and dynamic tables. Second, BED presents the first use of outline processing tools for online specification editing. Third, BED uses a database to track specification changes and a deep library structure. An additional contribution is in the user interface realm: what is successful about BED is all of the little details of the interface. Such features include having different fonts for default and specified arguments, denoting hyperlinks in a small amount of space, and supporting iconification. Each of these features represents a small portion of the code, but makes the system much more pleasant to use. While this implementation of a specification editor has serious performance problems, the functionality it provides is both useful and novel.

Bibliography

[1] Donald G. Basile. *Flow Control for Object-Oriented Semiconductor Process Representations.* PhD thesis, Stanford University. Dept. of Electrical Engineering, 1992.

[2] Donald G. Basile, Paul Losleben, and Krishna Saraswat. FLEECE — Flow Control Extensions for Object-Oriented Semiconductor Process Representations. In *Seventh Annual SRC/DARPA CIM-IC Workshop*, August 1992.

[3] B. S. Becker and L. A. Rowe. HIP: A Hypermedia Extension of the PICASSO Application Framework. Master's thesis, University of California at Berkeley, Electronics Research Laboratory, Berkeley, CA 94720, 1990. Available as technical report number UCB/ERL M90/121.

[4] Bart J. Bombay. The BCAM Control and Monitoring Environment. Master's thesis, University of California at Berkeley, Electronics Research Laboratory, Berkeley, CA 94720, September 1992. Memorandum No. UCB/ERL M92/113.

[5] Duane S. Boning. *Semiconductor Process Design: Representations, Tools, and Methodologies.* PhD thesis, MIT, January 1991.

[6] John Boreczky and Lawrence A. Rowe. Building Common Lisp Applications with Reasonable Performance. Technical report, UCB, 1993. To be published.

[7] Harry Braverman. *Labor and Monopoly Capital: The Degradation of Work in the Twentieth Century.* Monthly Review Press, 1975.

[8] V. Bush. As we may think. *Atlantic Monthly*, pages 101–108, July 1945.

[9] Duane S. Boning (ed.). *Technology CAD Framework Architecture.* TCAD Framework Architecture Committee, 1.1 edition, May 1990.

[10] John B. Goodenough. Exception Handling: Issues and a Proposed Notation. *Communications of the ACM*, 18(12):683–696, December 1975.

[11] R. Hartzell, P. Griffin, J. Shott, E. Wood, and P. Losleben. Integrated Manufacturing/TCAD Specification System. In *Seventh Annual SRC/DARPA CIM-IC Workshop*, August 1992.

[12] Christopher J. Hegarty, Lawrence A. Rowe, and Christopher B. Williams. The Berkeley Process-Flow Language WIP System. In *SRC Techcon '90*, 1990. Also available

in techreport form as UCB/ERL-M90/77 from the Electronics Research Laboratory, College of Engineering, UC Berkeley, Berkeley, CA 94720.

[13] Christopher James Hegarty. *Process-Flow Specification and Dynamic Run Modification for Semiconductor Manufacturing.* PhD thesis, University of California at Berkeley, Electronics Research Laboratory, Berkeley, CA 94720, April 1991. Memorandum No. UCB/ERL M91/40.

[14] Michael Heytens and Rishiyur S. Nikhil. GESTALT: An Expressive Database Programming System. *ACM SIGMOD Record*, 18(1):54–67, March 1989.

[15] C. P. Ho, J. D Plummer, S. E. Hansen, and R. W. Dutton. VLSI process modeling — SUPREM-III. *IEEE Trans. Electron Devices*, ED-30(11):143–152, November 1983.

[16] George Howlett. Graph-1.0. Available by ftp from barkley.berkeley.edu in pub/tcl/extensions/graph-1.0.tar.Z., 1992.

[17] Doug Johnson, Texas Instruments. Presentation at UC Berkeley, 1991.

[18] Sonya E. Keene. *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS.* Addison-Wesley Publishing Company, 1989.

[19] K. Lee and A. R. Neureuther. SIMPL-2 (SIMulated Profiles from the Layout — version 2). In *Symposium on VLSI Technology*, May 1985.

[20] Michael B. McIlrath, Donald E. Troxel, Duane S. Boning, Michael L. Heytens, and Paul Penfield Jr. CAFE — The MIT Computer Aided Fabrication Environment. In *Proceedings of the Ninth IEEE/CHMT International Electronics manufacturing Technology Symposium*, 1990.

[21] David C. Mudie and Norman H. Chang. FAULTS: An Equipment maintenance and Repair System. In *IEEE/CHMT International Electronics Manufacturing Technology Symposium*, October 1990.

[22] David C. Mudie and Norman H. Chang. FAULTS: An Equipment Maintenece and Repair System Using a Relational Database. Technical Report UCB/ERL-M90/95 and M91/44, UC Berkeley, Electronics Research Laboratory, College of Engineering, UC Berkeley, Berkeley, CA 94720, October 1990.

[23] Theodor Holm Nelson. *Literary machines : the report on, and of, Project Xanadu concerning word processing, electronic publishing, hypertext, thinkertoys, tomorrow's intellectual revolution, and certain other topics including knowledge, education and freedom.* Ted Nelson, Swarthmore, PA, 1987.

[24] John K. Ousterhout. An X11 Toolkit Based on the TCL Language. In *Proceedings of the 1991 Winter USENIX Conference*, pages 105–115, 1990.

[25] John K. Ousterhout. TCL: An Embeddable Command Language. In *Proceedings of the 1990 Winter USENIX Conference*, pages 105–115, 1990.

[26] Jeff Yung-Choa Pan, Jay M. Tenenbaum, and Jay Glickson. A Framework for Knowledge-Based Computer-Integrated Manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 2(2):33–46, May 1989.

[27] M. R. Pinto, C. S. Rafferty, and R. W. Dutton. PISCES II: Poisson and Continuity Equation Solver. Technical report, Integrated Circuits Lab, Stanford University, Palo Alto, CA 94305, 1984.

[28] T. Reps and T. Teitelbaum. *The Synthesis Generator Reference Manual.* Department of Computer Science, Cornell University, Ithaca, NY, July 1987.

[29] Lawrence A. Rowe, Joe Konstan, Brian Smith, Steve Seitz, and Chung Liu. The Picasso Application Framework. Memorandum UCB/ERL M90/18, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, March 1990.

[30] Lawrence A. Rowe and Brian Smith. A Continuous Media Player. In *Third International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 334–344. IEEE Computer and Communication Societies with ACM SIGCOMM, SIGOIS, and SIGOPS, November 1992. See also Memorandum No. UCB/ERL M-M92/126 Electronics Research Laboratory, Berkeley, CA 94720.

[31] Patricia Schank, Joe Konstan, Chung Liu, Lawrence A. Rowe, Steve Seitz, and Brian Smith. Picasso Reference Manual. Memorandum UCB/ERL M90/79, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, September 1990.

[32] Robert W. Scheifler, James Gettys, and Ron Neuman. *X Window Systen: C Library and Protocol Reference.* Digital Press, 1988.

[33] Jeffrey C. Sedayko and Lawrence A. Rowe. A Structured Editor for Process Specification, May 1990. Masters report, available as Memorandum UCB/ERL M90/48.

[34] Brian C. Smith and Lawrence A. Rowe. An Ad Hoc Query Interface for ICCIM Databases. *IEEE Transactions on Semiconductor Manufacturing*, 5(4):281–289, November 1992.

[35] Stephen R. Smoot. VED: A Visual Editor for BPFL. Unpublished, August 1991.

[36] Guy L. Steele. *Common Lisp: The Language.* Digital Press, second edition, 1990.

[37] John S. Wenstrad, Hiroshi Iwai, and Robert W. Dutton. A Manufacturing-Oriented Environment for Synthesis of Fabrication Processes. In *IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 376–379. Dig. Tech. Papers, November 1989.

[38] John Stewart Wenstrad. *An Object-Oriented Model for Specification, Simulation, and Design of Semiconductor Fabrication Processes.* PhD thesis, Stanford, Integrated Circuits Laboratory, Stanford University, Palo Alto, CA 94305, March 1991. Technical report number ICL91-003.

[39] C. B. Williams. A Process-Flow Specification Language for Manufacturing Semiconductor Integrated Circuits. PhD Thesis, University of California at Berkeley, in preparation.

[40] C. B. Williams and L. A. Rowe. The Berkeley Process-Flow Language: Reference Document. Memo 87.73, Electronics Research Lab, U.C. Berkeley, October 1987.

[41] A. S. Wong. An Integrated Graphical Environment for Operating IC Process Simulators. Memo 89.67, Electronics Research Lab., UC Berkeley, May 1989.