CONTROL LOOPS AND DYNAMIC RUN MODIFICATION USING THE BERKELEY PROCESS-FLOW LANGUAGE

by

Christopher J. Hegarty and Lawrence A. Rowe

Memorandum No. UCB/ERL M91/87

25 September 1991

CONTROL LOOPS AND DYNAMIC RUN MODIFICATION USING THE BERKELEY PROCESS-FLOW LANGUAGE

by

Christopher J. Hegarty and Lawrence A. Rowe

Memorandum No. UCB/ERL M91/87

25 September 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CONTROL LOOPS AND DYNAMIC RUN MODIFICATION USING THE BERKELEY PROCESS-FLOW LANGUAGE

by

Christopher J. Hegarty and Lawrence A. Rowe

Memorandum No. UCB/ERL M91/87

25 September 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Control Loops and Dynamic Run Modification using the Berkeley Process-Flow Language

Christopher J. Hegarty and Lawrence A. Rowe

Abstract

This paper describes advanced features of a *work in progress* (WIP) system for use in semiconductor fabrication. The WIP system is part of a *computer-integrated manufacturing* (CIM) system which uses a distributed heterogenous database and is based on a special purpose programming language designed for manufacturing, the Berkeley Process-Flow Language (BPFL). Support for feedback control is demonstrated by the application of the BPFL WIP system to a photolithography workcell controller. The ability of the system to permit ad hoc changes to active runs is also described.

I. Introduction

The goal of *computer-integrated manufacturing* (CIM) is to use computer and information management technology to integrate and automatically execute manufacturing operations. Two key elements of a CIM system are:

1. a shared, integrated, distributed database and
2. an executable process-flow representation suitable for all manufacturing phases.

This paper describes a *work-in-progress* (WIP) system using a special purpose programming language designed for manufacturing and a relational database management system for semiconductor *integrated-circuit* (IC) manufacturing.

Traditional WIP systems in the semiconductor industry are based on run-sheet specifications. These systems use a process specification stored on a computer that describes the operations required at each processing step and indicates where wafers should be moved when the step is complete. Examples of commercial run-sheet systems are WORKSTREAM [1] and PROMIS [2]. The process representations used in these systems includes commands to communicate with an operator through a form displayed on a terminal and, in some cases, communicate with equipment. However, these representations do not have the power of a full-function programming language so data structures (e.g., arrays, records, etc.) and control structures (e.g., conditional statements, looping statements, exception handling, etc.) are not provided[1]. Data and control structures are needed in a process representation to specify conditional processing (e.g., if it has been a long time since preventative maintenance was done on a piece of equipment, tweak the recipe parameters to compensate for the change in equipment performance) and feedforward and feedback control. More importantly, exception handling mechanisms are needed to allow the CIM system to respond to unanticipated events (e.g., equipment failures).

Several research groups have been working on advanced WIP systems. Examples include the MIT CAFE system [3] and a WIP system developed at Siemens [4].

This paper describes the Berkeley Process-Flow Language (BPFL) and the design and implementation of a WIP system that uses it. The paper is organized as follows. Section II presents an overview of the CIM system being developed at the Microfabrication Laboratory at the University of California at Berkeley. Section III discusses basic approaches to the design of process-flow representation and introduces BPFL. Sec-

1. WORKSTREAM has a scripting language with some control structures that can be called from a run-sheet command. However, these commands cannot be executed directly in the run-sheet, which severely limits the flexibility of the system.

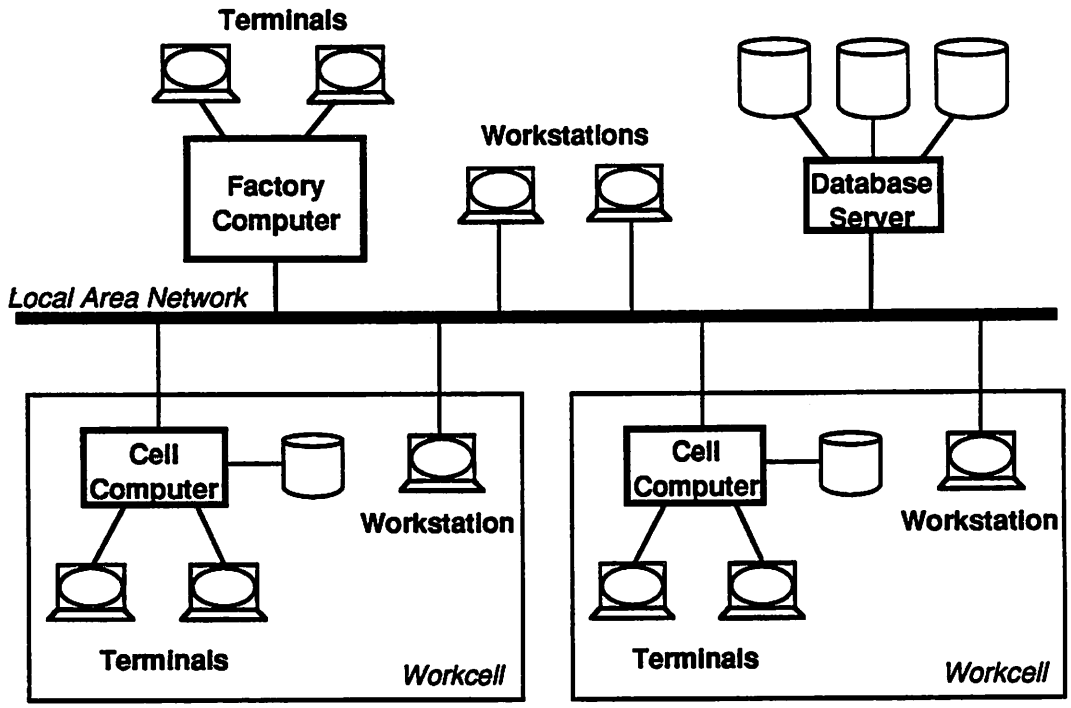

Figure 1. Typical IC-CIM fab computing system.

tion IV describes the architecture of the BPFL WIP system. Section V discusses the use of BPFL in an adaptive process control experiment, and Section VI describes the features available for ad hoc modification of the process flow used by an active run.

II. The Berkeley CIM System

The development of the system described in this paper has been influenced by our vision of a CIM system architecture. The system runs in a distributed heterogeneous computing environment composed of a variety of computers connected by a local-area network. A typical fab might use large microcomputers for cell controllers, a large mini- or mainframe computer for area and factory control, and a collection of workstations and terminals for user interactions. Figure 1 shows a typical system. Notice that cell controllers have local databases and that the fab has a large shared database server which motivates the need for a distributed database. Terminals and workstations are provided where appropriate. Equipment is connected to the cell controllers. Programs on any computer can access databases and programs running on any other computer using an interprocess communications protocol.

A key component of the system is a shared database that stores all information about the design and manufacture of semiconductors. This database contains information about the manufacturing facility, process-flow specifications, WIP, equipment, test data, product inventory, and orders. While the database is treated logically as a single centralized database, the architecture that we envision stores data in a distributed heterogeneous database (e.g., Gestalt [5] or INGRES/STAR [6]). Data is stored on the computer that optimizes the cost, reliability, and access constraints imposed by its use. A heterogeneous distributed DBMS is required because different applications in the fab have different data requirements and one DBMS cannot satisfy all these requirements. For example, the real-time performance and data volume required by some on-line monitoring applications can only be met today by file systems.

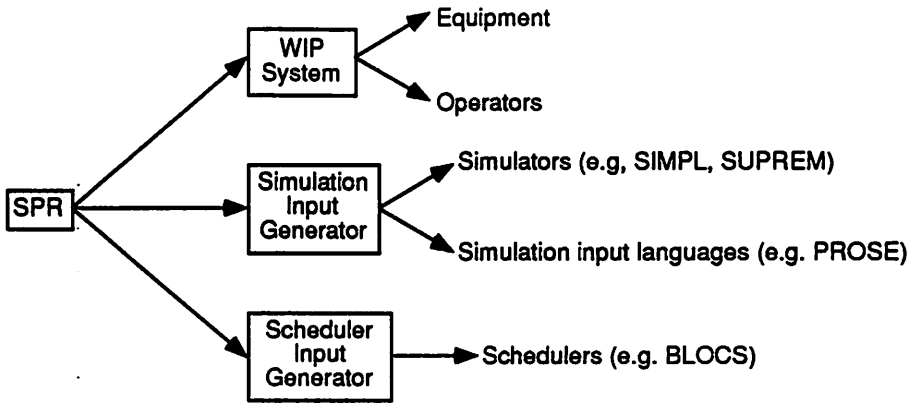

Figure 2. Information flow in SPR interpreters.

A third generation database system which supports relational data storage and access, an object-oriented data model (i.e., inheritance, user-defined data types, and methods), and a rules system [7] is required for many CIM applications. An example is the POSTGRES system being developed at Berkeley [8–9]. A third generation database system can store and access data that cannot be stored and accessed easily in a conventional relational database. For example, measurements collected during wafer processing are often represented by a sequence of values with units. A third generation database system can store arrays of user-defined data types (e.g., values with unit designations) in a table.

A *semiconductor process representation* (SPR) is a sequence of instructions that specifies how to manufacture a semiconductor product[1]. Another important part of a CIM system is an executable SPR. The goal of an SPR is that it be complete and facility independent. It must be executable to aid in process automation. Moreover, an SPR should be applicable to all stages of manufacturing (i.e., design, fabrication, and testing). An SPR interpreter *executes* a specification to accomplish a goal. Different interpreters accomplish different goals by performing different computations on the same specification as illustrated in Figure 2. For example, commands are issued to people and equipment when a WIP interpreter executes a process flow. Another interpreter will produce input commands for a process simulator (e.g., SIMPL [10], SUPREM [11]) or a simulation input language (e.g., PROSE [12]) when it executes the same process flow. A scheduling interpreter generates timing information for use by a scheduling system (e.g., BLOCS [13]).

The Berkeley Process-Flow Language (BPFL) [14] is a procedural SPR that is used in the Berkeley CIM system. The CIM database is used by BPFL and its associated interpreters in several ways. First, BPFL programs themselves are stored in the database. A software version-control system is implemented on top of the DBMS to manage libraries of BPFL procedures.

Second, BPFL interpreters use information in the CIM database. For example, the equipment in the fab and its current status is maintained in the database [15]. A scheduler uses this data to determine which piece of equipment should be allocated to a run. Another example is the WIP system, which stores the state of all active runs in the database so that the system can recover from a computer failure.

Third, BPFL programs store and access data in the CIM database. For example, an event log that records the start- and end-times of operations, in-process and in-situ measurements collected during processing, and other processing information is stored in the database. This log can be accessed by a BPFL procedure to change future processing based on previously recorded measurements (i.e., feedforward or feedback control).

1. Another term used for an SPR is a process flow.

```
 1  step INIT-OX do
 2    wet-oxidation(time: {11 min}, temperature: {1000 degC},
 3                  target-thickness: {1000 angstrom});
 4    pattern(mask-name: 'NWELL);
 5  end;
 6  step WELL-IMPLANT do
 7    with-lot '(cmos, nwell) do
 8      implant(species: #m(P), dose: {4.0e12 /cm^2},
 9              energy: {150 keV});
10      anneal-implant();
11      etch-oxide(etchant: #m(BHF, dilution: 5/1));
12      strip-resist();
13      step DRIVE-IN do
14        well-drive(temperature: {1150 degC}, time: {4 hr},
15                   anneal-time: {5 hr});
16        measure-oxide-thickness(location: #1(NWELL));
17        measure-oxide-thickness(location: invert-layer(#1(NWELL)));
18        with-lot 'nwell do
19          etch-oxide(etchant: #m(BHF, dilution: 5/1));
20          measure-sheet-resistance(location: #1(NWELL));
21        end;
22      end;
23    end;
24  end;
```

Figure 3. BPFL specification example.

III. The Berkeley Process-Flow Language

There are two basic approaches to the design of an SPR: knowledge-based and procedural. A knowledge-based approach uses a hierarchical, object-oriented data structure to represent a process flow. A procedural, or programming-language, approach represents a process flow by a program. Both approaches have essentially the same expressive power. Consequently, the kind of representation is less important than the particular constructs and abstractions that are provided.

BPFL is a procedural SPR so process flows are programs. Process flows consist of statements and procedure calls which are executed by a BPFL interpreter. The following goals influenced the design of BPFL:

1. Allow all manufacturing operations to be specified including lot splits and merges, conditional tests, feedforward and feedback control, rework loops, timing constraints, equipment and operator communication, and exception handling.
2. Separate the facility-specific information from the process specification to make it easier to change equipment in a fab or to move a process to another fab.
3. Allow a process specification to be used as input to other programs (e.g., process simulators and checkers and factory scheduling systems) to reduce the time required to design a process and manufacture product.

An example of BPFL code is shown in Figure 3. The code describes the fabrication of the isolation well in a standard CMOS process (named `cmos-16`) used in the Berkeley Microfabrication Laboratory. For illustration purposes, line numbers are shown to the left of each line of code and are referred to in the following discussion. The **`step`** statement in line 1 specifies a process step. The statement has a name (e.g., `INIT-OX`) and a body. The body contains the operations in the step. A **`step`** statement is used primarily for documentation purposes. For example, the step name is recorded whenever a measurement is recorded[1].

The body of a **step** statement includes procedure calls. In this example, the body is the code in lines 2–4 and consists of calls to the `wet-oxidation` and `pattern` procedures. In BPFL, arguments can be passed to procedures either by position or by name. Arguments passed by name can be passed in any order because the formal argument name precedes the value in the call. For example, the statement in lines 2–3 is a call to the BPFL procedure `wet-oxidation` with three named arguments (i.e., `time`, `temperature` and `target-thickness`). Notice the use of unit designators for constants. The `time` argument specifies an oxidation time of 11 minutes. Procedures may also have default values for arguments that are not supplied in a given procedure call. For example, the `wet-oxidation` procedure has an argument called `anneal-time` with a default value of 20 minutes. Because the `anneal-time` argument is not supplied in this example, the default value is used.

BPFL provides abstractions to manipulate wafers and lots, since they form the basic units on which processing is performed. A *lot* is a named set of wafers. Predefined lot names are supplied for wafers that are intended for production (`product`), wafers that are to be scrapped (`scrap`) and wafers that need rework (`rework`). A given wafer may be in several lots at the same time, with the exception that wafers in the `scrap` lot may appear only in that lot. A process flow specifies which lots of wafers will be processed using the **`with-lot`** statement shown in line 7.

BPFL programs maintain a model of wafer state that is used to check processes for correctness, to store measurements, and to support moving wafers between different runs. The model is based on the *Profile Interchange Format* (PIF) [16]. BPFL uses a subset of PIF, called *naive PIF*, to represent the adjacency relationships between materials on a wafer. BPFL interpreters maintain a data structure called a *snapshot* that describes the profile of a wafer. Operations are provided to change snapshots to simulate the effects of processing operations (e.g., depositing material, removing material, etc.). Any information recorded about the state of a wafer (e.g., resistivity, gate oxide thickness, device transconductance) may be recorded in the PIF model. Information about the masks used in a process is required to define the wafer profile. For example the procedure call in line 16 measures the oxide thickness at a location on the wafer called `NWELL`[1]. A location is defined as the logical intersection or union of masks.

BPFL also has a material class hierarchy that is used to specify the properties of a material and the names that simulators use for the material. For example, the procedure call in line 11 etches oxide using 5/1 buffered hydrofluoric acid[2]. A similar hierarchy is defined for equipment which describes equipment capabilities (e.g., a description of the SECS[3] interface to a furnace).

BPFL has many other features that are not discussed here. For example, there are statements to specify control flow, (e.g., `if-then`) and common abstractions encountered in processing (e.g., rework loops, timing constraints, and exception handling) [14,17].

IV. The BPFL Work In Progress System

The software architecture of the WIP system is shown in Figure 4. The system is composed of many processes that communicate with users, equipment, and the CIM database. The main process is the WIP interpreter that executes runs. A run corresponds to an execution of a BPFL process flow. Each run is represented by data structures that describe the run state (e.g., the next statement to execute, the names and values of local variables created by the program, and data retrieved from the database). The WIP interpreter executes many runs at the same time. In other words, it is a server process.

---

1. Steps can be nested, and the concatenation of all current step names is called a *step-path*. The step-path is recorded with all processing events.

1. Locations are denoted by an escape sequence ("#l") followed by a location name.

2. Materials are denoted by a "#m" escape sequence followed by a material name and optional attributes.

3. Semiconductor Equipment Manufacturers Institute (SEMI) Equipment Communications Standard protocol (SECS).

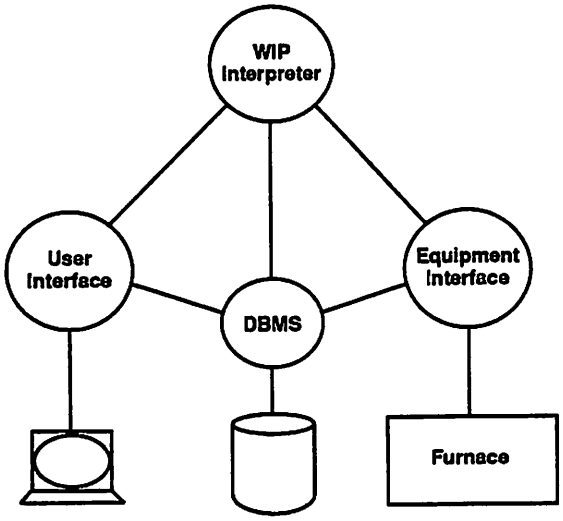

Figure 4. WIP system architecture.

The user interface (UI) process(es) support communication with operators and is the user-interface to runs. Operators at different locations in the fab can communicate with any run by connecting to the WIP interpreter through a UI process. A BPFL procedure called `user-dialog` is used to communicate with an operator. Commands are sent to the appropriate UI process.[1] The UI process is an INGRES Application-By-Forms (ABF) [18] program in the current prototype. The UI process reads information about the state of a run from the database and displays it to the user. Each active user has a UI process through which the user can respond to user-dialogs, examine run state, and browse the database. The UI process also writes events (e.g., user-dialog events) to the WIP-log in the database and is responsible for enforcing access control to runs.

The equipment interface (EI) process(es) support communication with equipment. Each EI process is an instance of Wood's SECS server [19–20]. An object-oriented SECS interface is defined within BPFL, and methods are defined for high-level equipment operations (e.g., run recipe, monitor run, fetch equipment status, etc.). These methods are implemented by remote procedure calls that invoke SECS commands implemented in the EI process.

These processes communicate either through interprocess communication channels (IPC) or through the shared CIM database. Internet-domain connections (TCP/IP) are used for real-time notification. Non real-time communication is implemented by the database. The WIP interpreter checkpoints the states of runs in the database so that other users and programs can access run information and run state can be recovered if a computer or network fails.

ABF applications use *frames* as the user-interface. A frame consists of two components: a *form* that displays information to the user and in which the user enters information, and a *menu* listing the available operations that the user can execute [21]. The main frame of the UI process is the `Run-Summary` frame shown in Figure 5. The top line of all frames in the UI process displays system information: the system version and the name of the current frame. Most of the screen area is taken up by a `Run-Information` table. This table displays a list of runs and information about them including their status (i.e., `running`, `waiting`, `stopped`, `aborted`, `finished`), the process flow, the current step, and the run owner. The bot-

1. In a low volume fab such as the Microfabrication Laboratory, a user moves to a different terminal and reconnects to the run. In a high volume fab, the WIP system sends the command to the user interface process at the appropriate workcell.

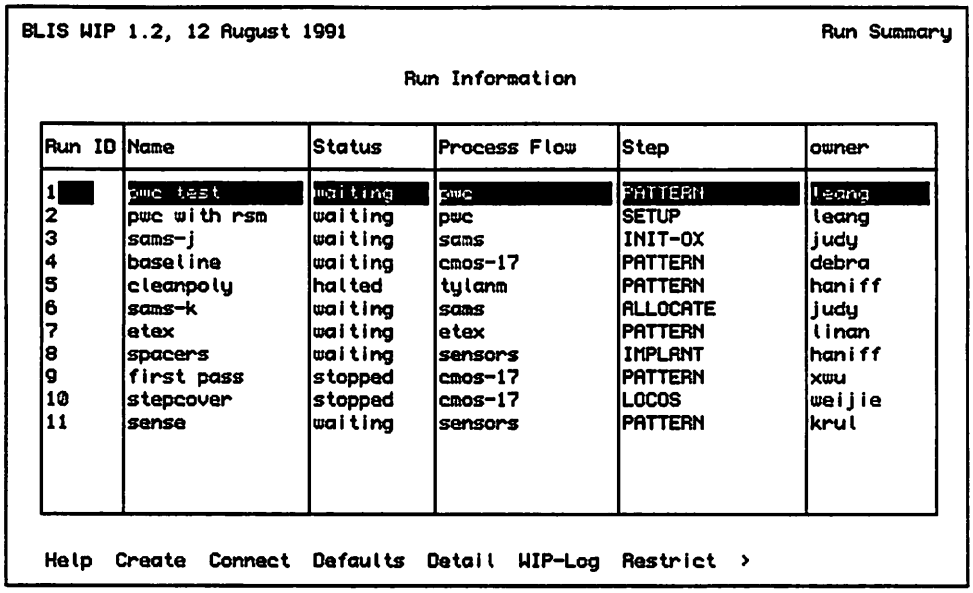

Figure 5. `Run-Summary` frame.

tom line in a frame lists the operation menu. The operations in the `Run-Summary` frame are listed in Table 1[1].

As an example of user interaction, consider a run using the BPFL code in Figure 3. When line 16 is interpreted, the `measure-oxide-thickness` procedure will be called and a BPFL `user-dialog` procedure will be executed. The user will be alerted to the fact that the run requires attention, and when the `Connect` operation in the `Run-Summary` frame is executed, the `Nanospec` frame shown in Figure 6 is displayed[2]. The top few lines of the form display information about the run. The middle portion of the form displays information about the required operation. The `measure-oxide-thickness` procedure has queried the wafer-state model for the `NWELL-1` wafer and extracted the anticipated oxide thickness on the

Operation	Description
`Help`	Displays help screen for the frame.
`Create`	Create a new run.
`Connect`	Connect to an existing run.
`Defaults`	Set up user defaults for the WIP system.
`Detail`	Provide more information about a run.
`WIP-Log`	Display the WIP-Log for a run.
`Restrict`	Enter criteria for runs to display (e.g., only runs owned by a particular user).
`Version`	Displays process-flow version information.
`Quit`	Leave the WIP system.

Table 1. `Run-Summary` frame operations.

1. The menu for the `Run-Summary` frame is too long to fit across the screen, and the `Version` and `Quit` operations do not appear in the menu in Figure 5. The `'>'` character after the `Restrict` operation is used to indicate that more operations are available, and ABF provides mechanisms for viewing them.
2. Oxide thickness measurements in our facility are carried out using a Nanometrics Nanospec thin film measurement instrument.

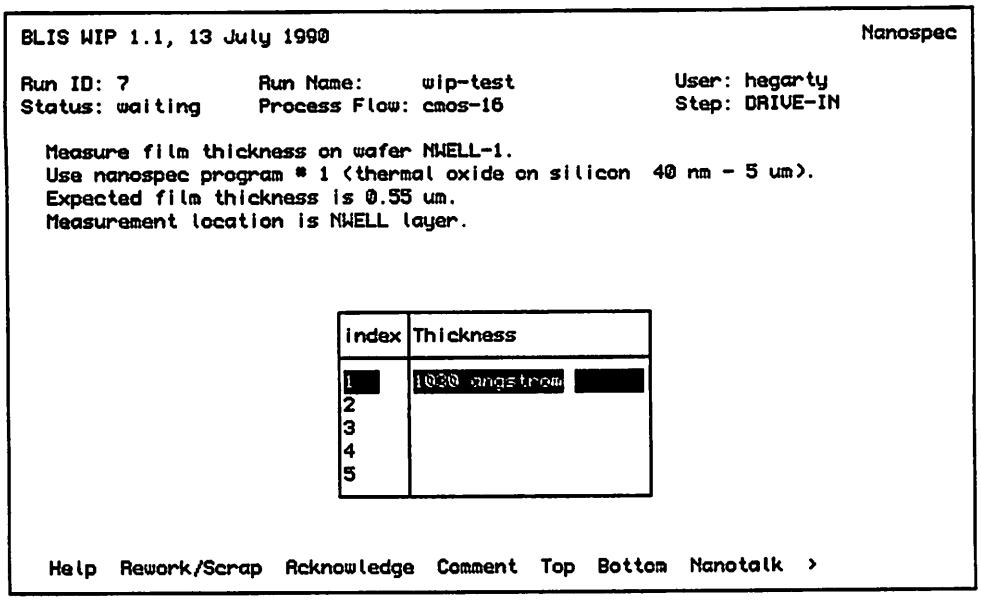

Figure 6. `Run-Summary` frame.

surface at the wafer within the well. The table in the lower part of the frame is where measured values are entered by the user. The user can either type values manually or use the `Nanotalk` operation which will use the SECS link to the nanospec to read the values.

The `Nanospec` frame is an example of a user-dialog frame. All user-dialog frames support the operations listed in Table 2, although some frames support additional operations. For example, the `Nanospec` frame supports three additional operations: `Nanotalk`, `Top`, and `Bottom`.

V. Process Control in BPFL

Semiconductor fabrication processes run for weeks or months, and it is frequently necessary to modify the parameters of a process during a run. Some modifications are planned and are accommodated in the design of the SPR and the support environment. For example, processing may be dynamically changed on the basis of data collected during prior processing of other lots on the same piece of equipment (i.e., feedback control) or on the basis of data collected during prior processing of the same lot (i.e., feedforward control). Systems that employ feedforward and feedback control are often referred to as *control-loops*. Control loops allow process designers to reduce manufacturing variability caused by equipment variations.

The BPFL WIP system implements control loops with parameters stored in the CIM database. A prototype photolithography workcell controller that uses a control loop to model the resist coating step has been implemented using BPFL [22]. The goal of the photolithography workcell controller is to control the thick-

Operation	Description
`Help`	Displays help screen for the frame.
`Rework/Scrap`	Force rework or scrap wafers.
`Acknowledge`	Respond to the dialog.
`Comment`	Attach a comment to the dialog.
`End`	Return without responding to the dialog.

Table 2. User-dialog frame operations.

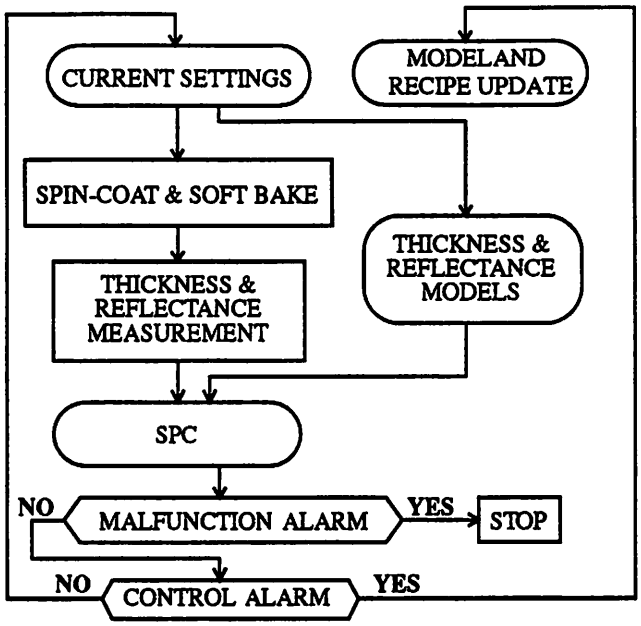

Figure 7. Schematic representation of the photolithography workcell controller feedback procedure [22].

ness and the photoactive compound (PAC) concentration of photoresist applied on oxidized silicon wafers. Statistical experiment design based on the response-surface method (RSM) is used to empirically model the step. The model is parameterized in terms of the measurable characteristics of the resist (i.e., film thickness and reflectance). The process is coded in BPFL using automatic measurement and download of recipe parameters to the equipment. An outline of the procedure is shown in Figure 7. First, the wafers are coated using the current settings for the resist coat parameters (i.e., spin speed, spin time, soft-bake time and soft-bake temperature). The resist thickness and reflectance are measured, and the statistical process control (SPC) procedure is called to determine if an alarm should be generated. The system generates two types of alarms, *malfunction alarms* and *control alarms*. Malfunction alarms signal increased process variability and cannot be corrected without human intervention (e.g., a bad temperature sensor on the hotplate causes incorrect bake temperatures). A malfunction alarm causes the process to be halted until the problem can be corrected. Control alarms result from occasional process disturbances that may be compensated for by an appropriate change in the recipe. In this case, the model and recipe are updated so that the disturbances will be compensated for in the next run. The recipe update procedure uses stepwise linear regression and is implemented in BLSS [23] which is called from BPFL.

The photolithography workcell controller is being extended for use in exposure and development, and a feedforward control methodology is being developed to further improve process control. These operations will be coded into BPFL.

VI. Ad Hoc Run Modification

This section describes the features available for ad hoc modification of a run. The WIP system allows runs to be modified while they are executing. A user can add or remove wafers, import wafers from another run, split a run into multiple runs, and modify the BPFL code used by a run.

Before describing how BPFL code may be modified, it is necessary to explain the structure of BPFL process flows. BPFL code consists of *flows*, which contain the top-level code for processing wafers (e.g.,

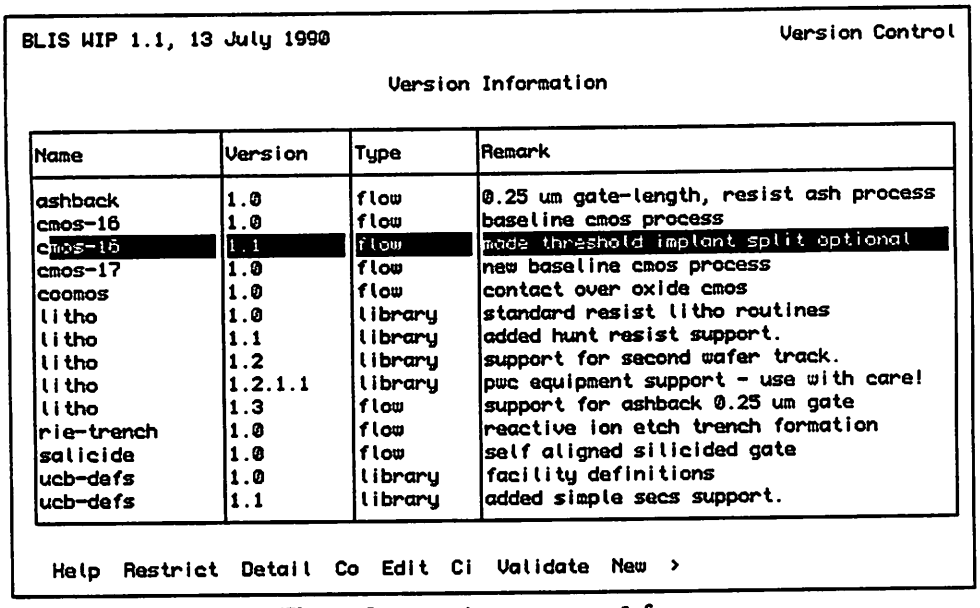

Figure 8. `Version-Control` frame.

`cmos-16`), and *libraries*, which contain standard procedures shared between flows (e.g., `wet-oxidation`). The term *module* is used to refer to both flows and libraries.

BPFL code may be created and edited using the `Version-Control` frame shown in Figure 8. The operations provided in the frame are described in Table 3. The WIP system uses the Revision Control System (RCS) to organize and maintain different version of BPFL code [24]. Whenever a user wishes to modify a module, he or she may *check out* the module if they are authorized to do so. Only one person may have a particular module checked out at any given time. The user can edit the module, run it, and debug it until satisfied that it is ready for use by others. The flow is then *checked in* and assigned a version number. RCS locks flows to prevent simultaneous modification of the same version, and stores the code modification tree in an efficient way. RCS permits the use of branch modifications so that several people can make modifications on the same version of a flow.

Wafers can be moved between lots and removed from a run by using the `Modify-Lots` frame shown in Figure 9. In this example, wafers from the `split-low` lot of the `baseline` run are to be moved to the

Operation	Description
`Help`	Displays help screen for the frame.
`Restrict`	Enter criteria restricting display.
`Detail`	Display further detail about the selected flow or library.
`Co`	Check out a flow or library.
`Edit`	Edit BPFL code
`Ci`	Check in a flow or library.
`Validate`	Parse BPFL code and check syntax.
`New`	Create a new flow or library.
`View`	Examine code without making changes.
`Update-Runs`	Update code used by runs.
`End`	Return to the Run-Summary frame.

Table 3. `Version-Control` frame operations.

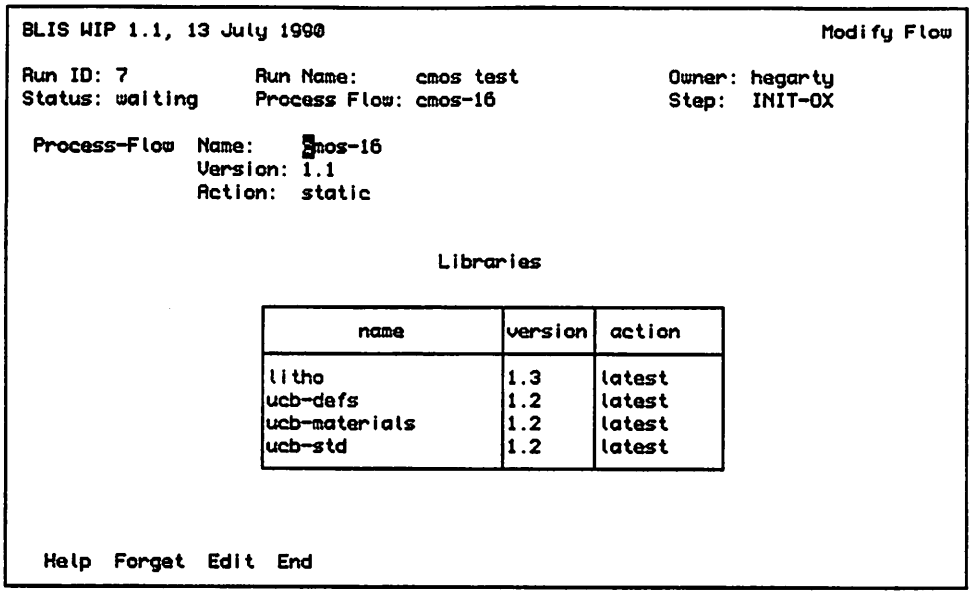

Figure 9. `Modify-Lots` frame.

`split-med` lot. A new lot can be created by typing in a `lot-name` that does not currently exist. The `New` operation allocates new wafers for a run.

Wafers can also be moved between runs. When a wafer is imported into a run, the wafer state description of the wafer is also imported and used to check for incorrect or undesirable processing. For example, to prevent cross-contamination caused by moving a wafer between runs that use incompatible processing.

A run may be split into multiple runs with the `Split-Run` frame shown in Figure 10. This operation allows a process engineer to experiment with different treatments on wafers that have undergone identical processing prior to the run split. In this example, the `baseline` run is being split into three runs, each of which will receive a different implant dose.

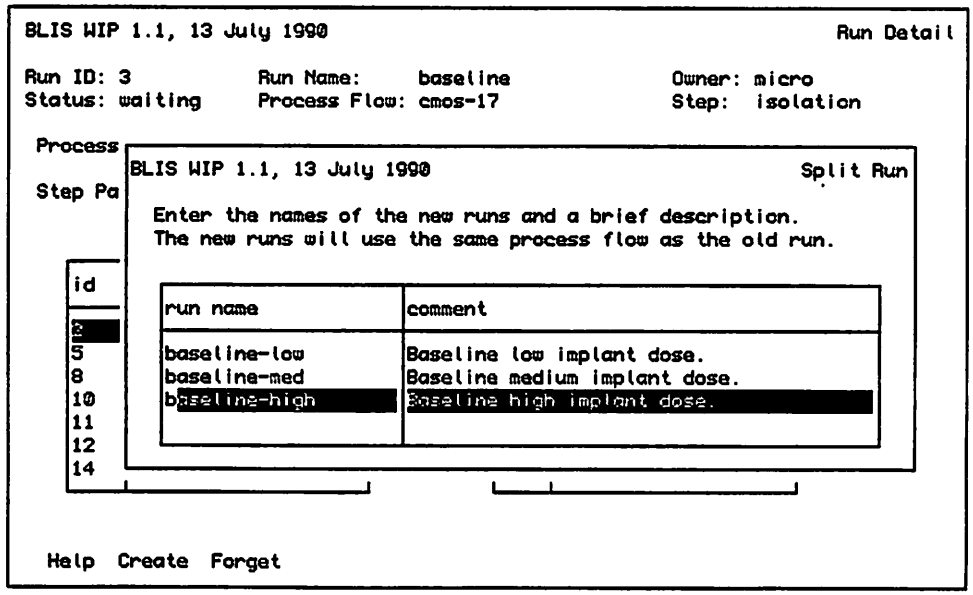

Figure 10. `Split-Run` frame.

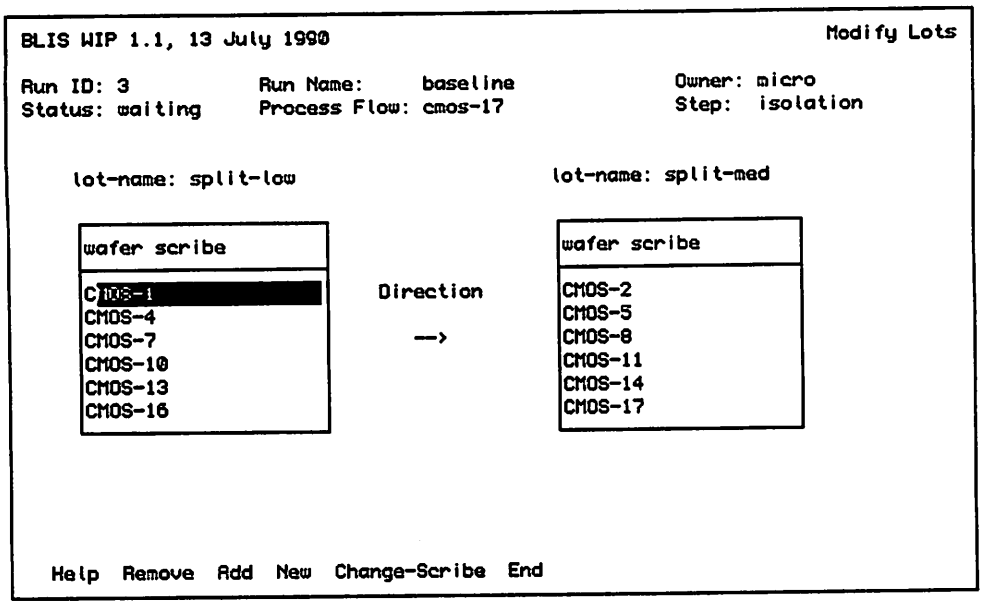

Figure 11. `Modify-Flow` frame.

BPFL process flows may be altered while a run is executing. Such changes are necessary to improve the process, correct it for errors, and to accommodate changes in facility policy. The desired response of the run to changes in its process-flow code are specified using the `Modify-Flow` frame shown in Figure 11. The `Process-Flow` in this case is version 1.1 of `cmos-16`. The `action` field specifies how the run responds to changes in code. This field can have one of three values:

1. `Static` – The process flow used by the run is never changed,
2. `Latest` – The process flow used by the run is always updated to the latest version available, or
3. `Query` - Whenever a new version of a process flow is created, the run owner is asked whether or not to use the new version.

In this example, the `action` field for the process flow has the value '`static.`' If later versions of `cmos-16` become available while the run is executing, they will not be used. The action field for the libraries (e.g., `litho`) has the value '`latest`' because library code is used to enforce facility policy and all active runs should use the latest versions of library code. If a run is set up to use the latest version of a flow, then all libraries used by the run are updated whenever the flow is updated. This is necessary to ensure that libraries which are compatible with the updated process flow are used.

VII. Conclusions and Future Work

In this paper we have described several capabilities of the BPFL WIP system. The ability to code control loops in BPFL aids the development of adaptive process control illustrated by the photolithography workcell controller. The version control system for process flows provides a simple mechanism to track modifications to flows and to control who can use and modify a process flow. Coupled with the ability of the system to dynamically modify the process-flow code used by an active run, the system overcomes a major shortcoming of procedural SPRs, the inability to easily modify the process used by a run once it has been started.

The UI process uses a terminal-based interface rather than a graphical user-interface (GUI) because the Berkeley Microfabrication Laboratory is equipped with ASCII terminals. Some operations that are cumbersome to perform with the current implementation (e.g., moving wafers between runs) are much easier to perform in a GUI. A GUI for the WIP system is under development and a sample screen is shown in Figure

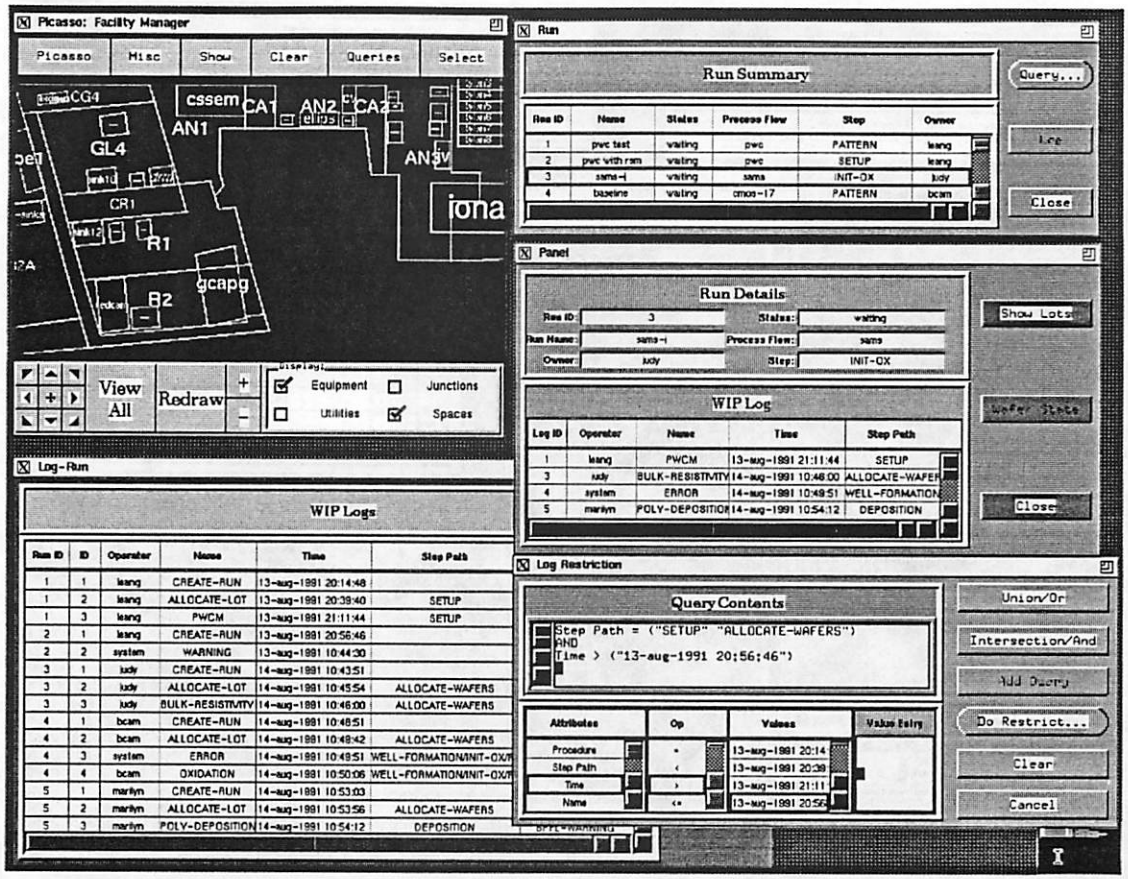

Figure 12. WIP system graphical interface.

12. This screen shows graphical prototypes for several of the frames used in the WIP system including the `Run-Summary` frame discussed in Section IV and the `WIP-Log` frame.

A graphical user-interface to BPFL is required because process engineers are not programmers and have no desire to learn programming. A graphical editor for BPFL is under development and a sample screen is shown in Figure 13. This figure shows the same code as Figure 3. The window on the left-hand side of the screen shows the graphical depiction of the top-level steps (i.e., `INIT-OX` and `WELL-IMPLANT`) within the code. The window in the center of the screen shows the graphical depiction of the `WELL-IMPLANT` step, and the window on the right of the screen shows the arguments to the `implant` procedure.

Acknowledgments

The authors wish to acknowledge the help of Sovarong Leang, Steve Smoot, Yan Or and the staff of the Berkeley Microfabrication Laboratory. Thanks also to David Hodges and Costas Spanos.

This research was supported by the National Science Foundation (Grant MIP-8715557), the Semiconductor Research Corporation, Philips/Signetics Corporation, Harris Corporation, Texas Instruments, National Semiconductor, Intel Corporation, Rockwell International, Motorola Inc., and Siemens Corporation with a matching grant from the State of California's MICRO program.

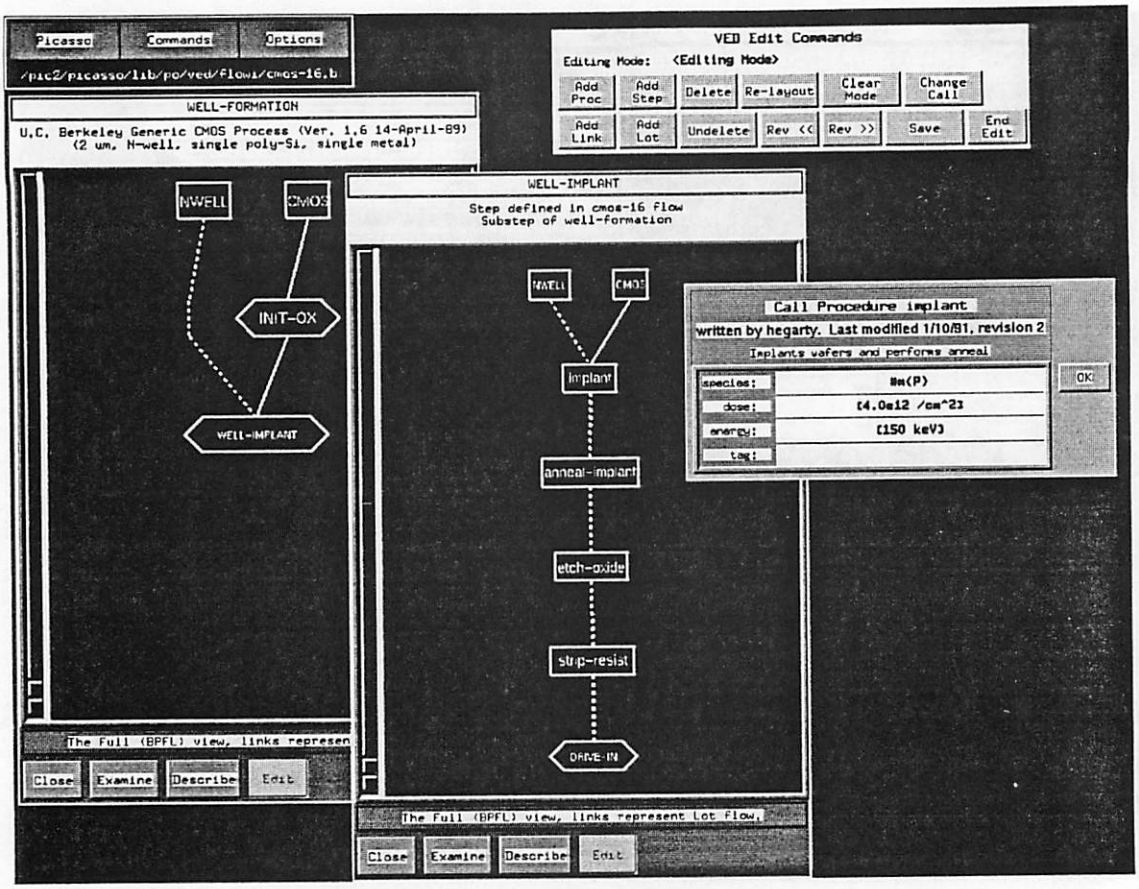

Figure 13. Graphical BPFL editor.

References

[1] *CAM Systems for Smart Shop Control*, Consilium, Mountain View, California, 1986.

[2] *The PROMIS System: Controlling the Journey to Factory Automation.* Promis Systems Corp., Toronto, Canada, 1987.

[3] D. Troxel, "The MIT CAFE System," *1989 DARPA/SRC Workshop on Integrated Factory Management for Integrated Circuits (IFM-IC)*, (College Station, TX, USA), Nov. 1989.

[4] Voorhees, E. M., "A Work-In-Progress Tracking System for Experimental Manufacturing," *Proc. Second Int. Conf. on Data and Knowledge Systems for Manufacturing and Engineering*, (Gaithersburg, MD, USA), pp 190–197, Oct. 1989.

[5] M. L. Heytens and R. S. Nikhil, "GESTALT: An expressive database programming system," *ACM SIGMOD Record*, vol. 18, no. 1, pp 54–67, Mar. 1989.

[6] *Distributed Ingres Manual*, Ingres Corp, Alameda, California, June 1989.

[7] M. R. Stonebraker, E. Hanson and C. H. Hong, "The Design of the POSTGRES Rules System," *IEEE Conf. Data Engineering*, (Los Angeles, CA, USA), Feb. 1987.

[8] M. R. Stonebraker and L. A. Rowe, "The Design of POSTGRES," *Proc. 1986 ACM-SIGMOD Conf. on Managment of Data*, (Washington, DC, USA), June 1986.

[9] M. R. Stonebraker, L. A. Rowe, and M. Hirohama, "The Implementation of POSTGRES," *IEEE Trans. on Knowledge and Data Engineering*, vol. 2, no. 1, March 1990.

[10] K. Lee and A. R. Neureuther, "SIMPL-2 (SIMulated Profiles from the Layout - version 2)," in *1985 Symposium on VLSI Technology*, (Kobe, Japan), pp. 64–65, May 1985.
[11] C. P. Ho, J. D Plummer, S. E. Hansen, and R. W. Dutton, "VLSI process modeling – SUPREM-III", *IEEE Trans. Electron Devices*, vol ED-30, no. 11, pp 1438–1452, Nov. 1983.
[12] A. S. Wong, "An Integrated Graphical Enviroment for Operating IC Process Simulators," Electronics Research Lab. Memo 89.67, University of California, Berkeley, May 1989.
[13] R. Glassey, "An Overview of BLOCS/M: The Berkeley Library of Objects for Control and Simulation of Manufacturing," *1989 DARPA/SRC Workshop on Integrated Factory Management for Integrated Circuits (IFM-IC)*, (College Station, TX, USA), Nov. 1989.
[14] L. A. Rowe, C. B. Williams and C. J. Hegarty, "The Design of the Berkeley Process-Flow Language," to appear in *IEEE Trans. Semiconductor Manufacturing*. Also available as ERL Report M90/62, University of California, Berkeley, July 1990.
[15] D. C. Mudie and N. H. Chang, "FAULTS: An Equipment maintenance and Repair System," *Proc. 1990 IEEE/CHMT International Electronics Manufacturing Technology Symposium*, (Washington, DC, USA), Oct. 1990.
[16] S. G. Duvall, "An Interchange Format for Process and Device Simulation," *IEEE Trans. on CAD*, vol. 7, no. 7, pp 741–754, Jul. 1988.
[17] C. J. Hegarty, "Process-Flow Specification and Dynamic Run Modification for Semiconductor Manufacturing," Electronics Research Lab. Memo M91/40, University of California, Berkeley, April 1991.
[18] *Ingres ABF/4GL Reference Manual*, Ingres Corp, Alameda, California, June 1989.
[19] J. L. Mohammed, *Common Lisp Implementation of SECS II Protocol*, Schlumberger Technologies, July 1990.
[20] E. J. Wood, H. Schenck and J. Wijaya, "Networking and Object-Oriented Coding for SECS Communication," *Proc. Automated IC Manufacturing Symp.*, Fall Electrochemical Society Meeting, Oct. 1987.
[21] L. A. Rowe, "*Fill-in-the-Form* Programming," *Proc. 11th Int. Conf. on Very Large Data Bases*, Aug. 1985.
[22] S. Leang and C. J. Spanos, "Statistically Based Feedback Control of Photoresist Application," to appear in *Proc. Advanced Semiconductor Manufacturing Conference '91*, (Boston, MA, USA), October 1991.
[23] D. M. Abrahams and F. Rizzardi, *BLSS – The Berkeley Interactive Statistical System*, (New York, NY, USA), W. W. Norton and Company, 1988.
[24] W. F. Tichy, "RCS – A System for Version Control," *Software – Practice and Experience*, vol. 15, no. 7, pp. 637–654, July 1985.