

Copyright © 1995, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**BCIMS: THE BERKELEY COMPUTER
INTEGRATED MANUFACTURING SYSTEM**

by

L. J. Massa-Lochridge

Memorandum No. UCB/ERL M95/46

27 June 1995

(Revised 18 August 1995)

**BCIMS: THE BERKELEY COMPUTER
INTEGRATED MANUFACTURING SYSTEM**

by

L. J. Massa-Lochridge

Memorandum No. UCB/ERL M95/46

27 June 1995

(Revised 18 August 1995)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

**BCIMS:
THE BERKELEY COMPUTER INTEGRATED
MANUFACTURING SYSTEM**

L.J. Massa-Lochridge

ABSTRACT

This document describes the Berkeley Computer Integrated Manufacturing System (BCIMS) being developed by the Computer Integrated Manufacturing (CIM) research group. This system implements process control, equipment management, access control, facility management, and administrative subsystems. A novel architecture for system integration is presented. A framework for developing integrated systems computer-human interaction layers, which hide specifics of heterogeneous platform clusters is described. BCIMS has been validated in operation at EECS/ERL's Microfabrication Laboratory.

Table of Contents

1.0	Introduction	4
2.0	Major features	6
2.1	Microlab access	6
2.2	Computer-Human interaction	6
2.3	Access control for equipment use	8
2.4	Reserving equipment for use	9
2.5	Data collection, equipment monitoring, and process control	10
2.6	An example of a typical BCIMS client at work	13
3.0	Tools for the Microlab's maintenance and research staff, and CIM researchers	15
3.1	Tools for process development and associated research staff	15
3.1.1	BLIMP	16
3.1.2	HotPotato	17
3.1.3	WIP	18
3.1.4	Statistical Process Control	18
3.2	Equipment maintenance tools	19
3.2.1	Gases	20
3.2.2	Techjob	21
3.2.3	Faults	21
3.3	Facility Management Tools: CIMtool	22
3.4	The Staff system	26
3.5	Administrative tools	26
4.0	System Design, Architecture, and Implementation	27
4.1	BCIMS system architecture	27
4.2	Software system design	28
4.2.1	The Wand HCI architecture and implementation	34
4.2.2	The BCIMS Relational Database System	42
4.2.3	BCIMS Equipment Control subsystems	46
4.2.4	Process Monitoring	49
4.2.5	Faults Equipment Maintenance and Diagnosis Tool	55
4.2.6	BCIMS Facility management subsystems	57
4.2.7	Administrative applications	59
4.3	BCIMS Hardware Architecture	62
5.0	History and organizational facts	65
5.1	History	65
5.2	BCIMS facts	66
5.3	Organizational Facts	67
6.0	Future research and development Directions	68
7.0	Appendix 1: Database Entity Relationship Diagrams	70
8.0	Appendix 2: Technical description of the Wand login system and related processes ..	80
9.0	Appendix 3: Program Listing	85
10.0	Bibliography and References	91

11.0	Acknowledgments	95
12.0	Author Acknowledgment	97

List of Figures

FIGURE 1	The Login Terminal	7
FIGURE 2	Wand main menu	8
FIGURE 3	The Reserve subsystems's initial form	10
FIGURE 4	The Equipment submenu	12
FIGURE 5	Tystar furnace status	13
FIGURE 6	Labuser activity and charges review	14
FIGURE 7	The Staff system	16
FIGURE 8	BLIMP Sensor Status output	17
FIGURE 9	SPC output	19
FIGURE 10	The Equipment status board, epstat	20
FIGURE 11	Techjob	22
FIGURE 12	A typical Faults report	23
FIGURE 13	CIMtool initial view, a floor plan of the lab	24
FIGURE 14	The CIMtool interface to BLIMP sensor data	25
FIGURE 15	Two-level software/hardware system architecture	29
FIGURE 16	Relationships between wand, BRDS, and subsystems	30
FIGURE 17	Subsystem layers and corresponding code libraries	32
FIGURE 18	Relational Database System Access	35
FIGURE 19	Database software subsystem integration	36
FIGURE 20	Wand login system: Process intercommunication	40
FIGURE 21	CIM Database: Top level entity-relationship diagram	43
FIGURE 22	Software and hardware data acquisition systems	47
FIGURE 23	Tycom Communication Subsystem	51
FIGURE 24	BLIMP sensor status output	53
FIGURE 25	The BCIMS multi-cluster network	63
FIGURE 26	CIM Database: Primary top-level entity-relationship diagram	71
FIGURE 27	CIM Database: Accounting tables entity-relationship diagram	72
FIGURE 28	CIM Database: Labuser tables entity-relationship diagram	73
FIGURE 29	CIM Database: Gases tables entity-relationship diagram	74
FIGURE 30	CIM Database: Inventory tables entity-relationship diagram	75
FIGURE 31	CIM Database: Purchase tables entity-relationship diagram	76
FIGURE 32	CIM Database: Qualify tables entity-relationship diagram	77
FIGURE 33	CIM Database: Reserve tables entity-relationship diagram	78
FIGURE 34	CIM Database: Resource tables entity-relationship diagram	79

1.0 Introduction

An integrated circuit computer-integrated manufacturing (CIM) system has been developed and used to control the Berkeley Microfabrication Laboratory (Microlab). The system, called the Berkeley CIM System (BCIMS) includes subsystems that implement the following functions:

- process control (e.g., work-in-progress (WIP) and real-time process monitoring and control);
- equipment management (e.g., equipment status logging and maintenance and repair);
- facility management (physical resource management, real-time monitoring of utilities, lab use scheduling); and
- administration (e.g., access control, accounting, inventory control, and purchasing).

The Berkeley CIM project was formed in the early 1980's to develop software to conduct CIM research and manage the Microlab. The Microlab is a class 100 clean room fabrication facility for semiconductor research.

The initial computing environment was a centralized time-shared computer with alphanumeric terminals connected to it. In the late 1980's, the environment changed to a distributed computing system composed of two servers: one for production and one for development. Workstations, X terminals, and alphanumeric terminals are connected to the servers over local area network. In addition there are several specialized hardware subsystems that are used for data acquisition, equipment control, and direct connection to processing or analytical equipment.

The principal design goal for BCIMS is that it be a portable, open system based on a shared, integrated database. It runs on the UNIX operating system and incorporates the Ingres Relational Database Management System (RDBMS). BCIMS is comprised of approximately

200 distinct programs written in a variety of programming languages including C, C++, Common Lisp, ABF/4GL, and UNIX shell scripts, and it is integrated through the database.

Our philosophy has been to develop prototype applications and then refine them into production applications in response to user feedback. Applications are developed by one or more students or staff members under the direction of a faculty advisor. The integration of subsystems, and particularly the logical database design, are maintained and enhanced by one or more student and staff software designers and engineers under the direction of one or more faculty and staff advisors. The modular software architecture and the shared database produced a flexible open system that can easily be extended to accommodate new applications and concurrent development efforts.

The remainder of this paper describes the operation, design, and implementation of BCIMS. Section 2.0 presents several examples that illustrate how BCIMS applications are used. Section 3.0 describes tools that have been developed to support the lab's maintenance, processing, administrative, and associated research staff. Section 4.0 describes the architecture, design and implementation of the system. Section 5.0 discusses the history and organizational details of the facility. Lastly, Section 6.0 describes our future plans and presents our conclusions.

2.0 Major features

This section presents some examples that will give the reader an understanding of how the system works from the lab client's point of view. The reader will be taken, through the typical operations that most lab members require, beginning with entry into the laboratory.

2.1 Microlab access

The gateway to the clean room portion of the lab is the "login terminal" located just inside the Microlab lobby doors. On the way into the clean room area, before gowning up, you sign-in at the login terminal. A sign-in is a simple login procedure, followed by a choice of either continuing immediately into the lab or accessing the BCIMS menu-driven interface and login system, called the Wand. The software checks to see if there are others in the lab; if there are not, the user is warned that he or she may not work alone. The sign-in procedure also begins the accounting of the user's time in the lab. After the sign-in, the user may gown up and proceed into the clean room area of the lab. FIGURE 1 shows the login terminal screen. Once inside the clean room, the user may log on to any workstation or terminal located at the various work cells in the lab.

2.2 Computer-Human interaction

The computer-human interaction subsystem, called the Wand, provides a consistent menu-based interface to more than 200 separate programs. The goal was to provide seamless computer-human interaction across disparate subsystems. It was designed for the naive user, to hide the complexity of the UNIX command shell language. In addition, the Wand offers an integrated view of the heterogenous multi-cluster of hardware platforms which are part of BCIMS in the Microlab, and can do so for a generic multi-cluster. The Wand completely hides system specifics from the user, including the existence of separate machines. From any terminal or workstation the user perception is that there is one computer and a

```

+argon
>>> Welcome to the MICROFABRICATION LABORATORY [argon:ttyp8]
>>> Enter your Account Name please: labwho
13 users @ Fri Aug 4 12:28:58 1995
Name                               Entry time      Idle      Total
Patrick Wehrly                     (wehrly) S Fri Aug 4 06:23 1:12      6:05
Marcelle Stagno                     (marcelle) S Fri Aug 4 07:30 2:02      4:58
Kim Allen                           (kallen) Fri Aug 4 07:43 1:33      4:45
Katalin Voros                       (voros) S Fri Aug 4 08:03 4:24      4:25
Phill Guillory                      (phillip) S Fri Aug 4 08:30 3:55      3:58
Zhenlei Bao                         (baoz) Fri Aug 4 08:33 2:57      3:55
Tariq Haniff                        (haniff) Fri Aug 4 09:07 :36*      3:21
Xiaofan Meng                        (mengxf) Fri Aug 4 09:14 1:35      3:14
+recharge: repair                  Fri Aug 4 09:14 ( 3:14)
Norton Mitchell                     (nortonm) S Fri Aug 4 11:18 1:07      1:10
Marilyn Kushner                     (marilyn) S Fri Aug 4 11:22 :48      1:06
Seung Hyuk Kang                     (shkang) Fri Aug 4 11:26 :31      1:02
Wen-Hwa Chu                         (whchu) Fri Aug 4 11:35 :43      0:53
Dongkyun Kim                        (dkim) Fri Aug 4 11:43 :21      0:45
>>> Welcome to the MICROFABRICATION LABORATORY [argon:ttyp8]
>>> Enter your Account Name please: █

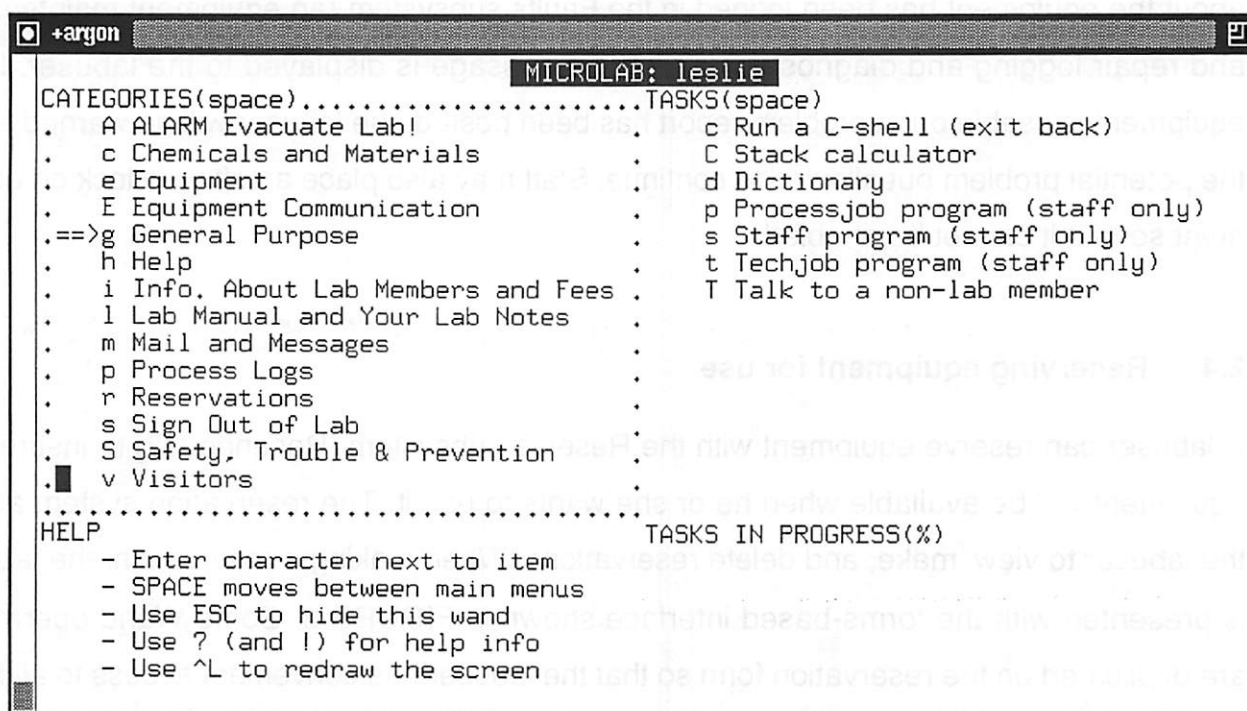
```

FIGURE 1 The Login Terminal

single login session, regardless of user movement from one terminal or workstation to another. The primary reason for this perception is that as a user logs into a machine at a work cell, her existing login session is transferred to the new location and displayed on the new screen.

A library of menu functions is available for use in developing each new software subsystem. Each application developed using the Wand libraries presents a uniform look and feel. FIGURE 2 shows a typical Wand text-based menu screen. The screen is divided into four areas. The upper left area, labelled **CATEGORIES**, is the main menu within the current context. The upper right area, labelled **TASKS** is the immediate submenu of the current selection (marked by an arrow) in the main menu. Because this interface was designed for the naive user, help operations are available and persistently displayed in the lower left portion of the screen labelled **HELP**. Lastly, the lower right area, labeled **TASKS IN PROGRESS**, shows a listing of subprocesses that the labuser (a term used in BCIMS to refer to a person who has access to the laboratory) has running. The labuser views the system as running on one

computer even though more than one program may be running at the same time on different machines. When a labuser logs out of one terminal in the lab, the state of the Wand and any running processes are saved. Existing processes are either suspended or kept running as appropriate. For example, a data collection process would continue to run, but an editor would be suspended. The labuser is reconnected to the Wand and the programs which were left running (if any) when he or she logs into the system again at any terminal on the local network. The Wand is accessible from anywhere on the Internet by logging into your Micro-lab account and typing "wand" on the command line.

A screenshot of a terminal window titled "+argon" with a user prompt "MICROLAB: leslie". The terminal displays a menu with two columns: "CATEGORIES(space)" and "TASKS(space)". The categories list includes items like "A ALARM Evacuate Lab!", "c Chemicals and Materials", "e Equipment", "E Equipment Communication", "g General Purpose", "h Help", "i Info. About Lab Members and Fees", "l Lab Manual and Your Lab Notes", "m Mail and Messages", "p Process Logs", "r Reservations", "s Sign Out of Lab", "S Safety, Trouble & Prevention", and "v Visitors". The tasks list includes "c Run a C-shell (exit back)", "C Stack calculator", "d Dictionary", "p Processjob program (staff only)", "s Staff program (staff only)", "t Techjob program (staff only)", and "T Talk to a non-lab member". Below the categories is a "HELP" section with instructions: "Enter character next to item", "SPACE moves between main menus", "Use ESC to hide this wand", "Use ? (and !) for help info", and "Use ^L to redraw the screen". At the bottom right, it says "TASKS IN PROGRESS(%)" with a percentage sign.

```
+argon MICROLAB: leslie
CATEGORIES(space).....TASKS(space)
.  A ALARM Evacuate Lab!      .  c Run a C-shell (exit back)
.  c Chemicals and Materials  .  C Stack calculator
.  e Equipment                .  d Dictionary
.  E Equipment Communication  .  p Processjob program (staff only)
.==>g General Purpose        .  s Staff program (staff only)
.  h Help                    .  t Techjob program (staff only)
.  i Info. About Lab Members .  T Talk to a non-lab member
.  l Lab Manual and Your Lab .
.  m Mail and Messages       .
.  p Process Logs            .
.  r Reservations             .
.  s Sign Out of Lab         .
.  S Safety, Trouble & Prev. .
.  v Visitors                 .
.....TASKS IN PROGRESS(%)
HELP
- Enter character next to item
- SPACE moves between main menus
- Use ESC to hide this wand
- Use ? (and !) for help info
- Use ^L to redraw the screen
```

FIGURE 2 Wand main menu

2.3 Access control for equipment use

Special training is required before most equipment in the Microlab can be used. The exact requirements are different for each piece of equipment. BCIMS controls access to the equipment so that only qualified labusers can access it. Data describing labuser qualifications are

stored in the database and checked each time a labuser executes an operation that accesses a piece of equipment. Qualifications expire after a time interval that the lab manager may set (every six months in the Microlab) to insure that a labuser who has not used equipment recently must requalify before using it again.

Located at each piece of equipment is a hardware interlock that physically disables a key function of the machine. When a currently qualified labuser selects an equipment operation, the hardware interlock is unlocked and the equipment is "enabled" for use. If a message about the equipment has been logged in the Faults subsystem (an equipment maintenance and repair logging and diagnosis system) the message is displayed to the labuser. If the equipment is usable but a problem report has been posted, the labuser will be warned about the potential problem but allowed to continue. Staff may also place a software lock on equipment so that it cannot be enabled.

2.4 Reserving equipment for use

A labuser can reserve equipment with the Reserve subsystem [Resende '87] to insure that equipment will be available when he or she wants to use it. The reservation system allows the labuser to view, make, and delete reservations. When making a reservation, the labuser is presented with the forms-based interface shown in FIGURE 3. Some Wand operations are duplicated on the reservation form so that the labuser has convenient access to all functionality that may be required. The **status** operator is mapped to a procedure which accesses the Faults system and retrieves the current status of equipment. The labuser may also make a new reservation, delete a reservation or look up existing reservations. In the screen shown, a listing of all reservations in the database for the current date has been selected. Alternately, one can ask for only those reservations made by a specific labuser or during a particular time period. Once the labuser has reserved the equipment, he or she has priority over all other labusers during that time.

08/04/95 12:50
mm/dd/yy hh:mm (24 hr clock)

Equipment name:
User name:
Process:

From:
To:

Equipment	User	From	To	Process
lgcapg	lmarilyn	131-jul-1995	08:00 04-aug-1995	17:00 masks
lmrc	lkallen	103-aug-1995	17:00 04-aug-1995	17:00 ZnO stress tests
lv401	lkallen	104-aug-1995	08:00 04-aug-1995	13:00 Ti/Au/Al
lgcaws	lmarilyn	104-aug-1995	09:00 04-aug-1995	11:00 focus patt
ltylan13	lhaniff	104-aug-1995	09:00 04-aug-1995	12:00
liv	lhaniff	104-aug-1995	09:00 04-aug-1995	12:00
ldisco	lhaniff	104-aug-1995	09:00 04-aug-1995	12:00
llam4	ltewodros	104-aug-1995	09:30 04-aug-1995	10:30 Etch
lbonder	lhaniff	104-aug-1995	12:30 04-aug-1995	15:00
ltylan20	ldhans	104-aug-1995	13:00 04-aug-1995	14:00 qualification

Use <TAB> and <RETURN> to move cursor. Use <ESC> or <CTRL> to choose a command.

Find Reserve Delete Equipment Status Quit

FIGURE 3 The Reserve subsystems's initial form

2.5 Data collection, equipment monitoring, and process control

During processing, labusers collect data from analytical, processing, and environmental monitoring equipment. Traditionally, data collected in semiconductor fabrication facilities has been stored on paper and on separate computers under incompatible software packages. This situation made it extremely difficult to share and use data from the disparate sources in any meaningful way (such as for work in progress, statistical process control, scheduling, or computer-aided design). Therefore, automated data collection, equipment monitoring, and process control have been implemented in BCIMS. In addition, using paper as a primary method of transferring information between people and for storage of data can be error-prone and inconvenient. Paper can't be continuously updated. In the clean room environment, any paper that is to be used must be a special type that will not contribute particulate matter to the air. Therefore, the use of paper under BCIMS has been minimized by providing for user document management and on-line manuals and help. FIGURE 4 displays the

Equipment submenu of the Wand; it offers options related to the status of equipment. Virtually all equipment supports some type of data collection from the equipment console. Some pieces of equipment have an RS232 interface that can be used to monitor the equipment and collect data. For a subset of equipment, capabilities available at the equipment console are duplicated by BCIMS software subsystems. These subsystems offer an improvement over alternatives that require the use of a specific console or terminal because with BCIMS the labuser is no longer tied to a particular location in the lab to collect data, monitor equipment status, or control processing. The equipment interface can be accessed from any machine on the network including a terminal in the labuser's office. Equipment that may be remotely operated through BCIMS includes but is not limited to:

- Lam Research etchers (various types)
- Tylan and Tystar furnaces (various types)
- Tencor's Alphastep 200 Profiler and flatness gauge
- Nanometric's film thickness measurement systems
- Prometrix Resistivity Mapping System

New equipment may be easily interfaced because the modularity of the equipment software requires only the specification of characteristics particular to the equipment.

A separate subsystem, the Berkeley Laboratory Infrastructure Monitoring Program (BLIMP), implements communication to a hardware/software system that monitors sensors in the lab. An off-the-shelf hardware data acquisition system (manufactured by Acurex) is used to deliver data from sensors to a host computer. BLIMP uses the input data collected by the Acurex data acquisition hardware to monitor environmental factors and resources in the lab such as gas pressures, fluid levels, etc. The features available through the equipment com-

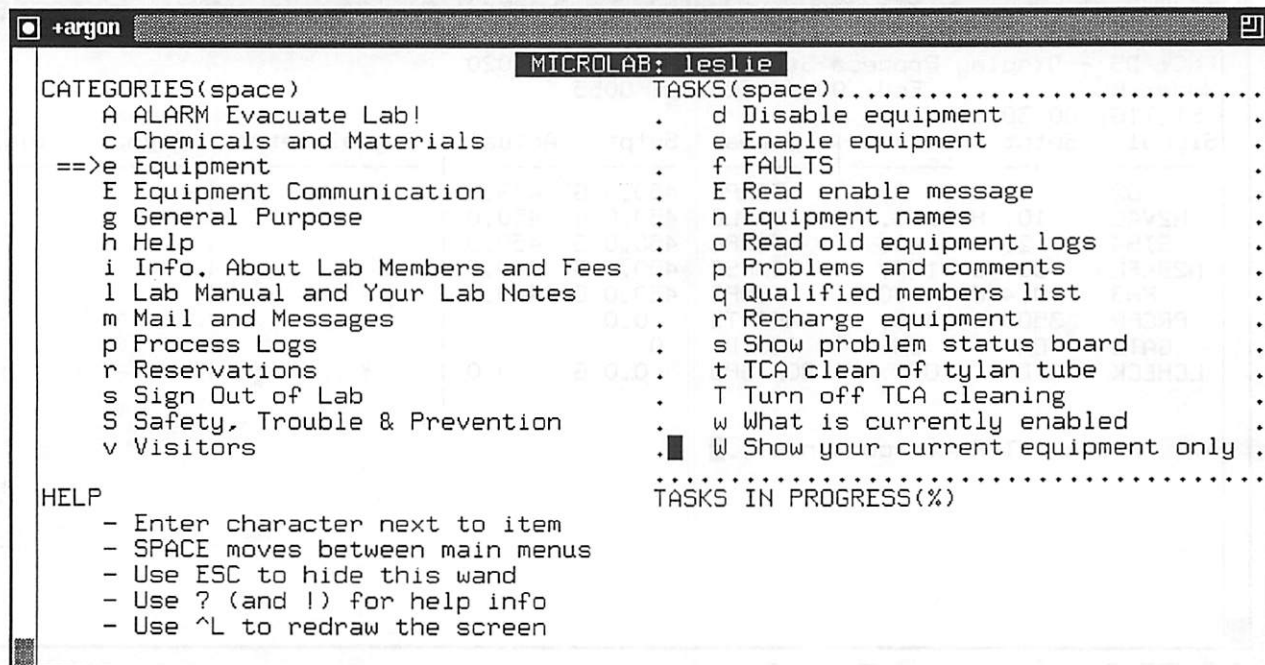


FIGURE 4 The Equipment submenu

munication subsystems vary with the different equipment but typical operations are:

- single status or continual status monitoring
- recipe viewing and/or recipe directory viewing
- recipe loading and/or downloading
- recipe editing
- single set or continuous data collection
- operation parameters setting

The Tystar Communication subsystem is representative of the equipment communication subsystems. FIGURE 5 shows the status returned by a Tystar furnace in response to a request through the communication interface.

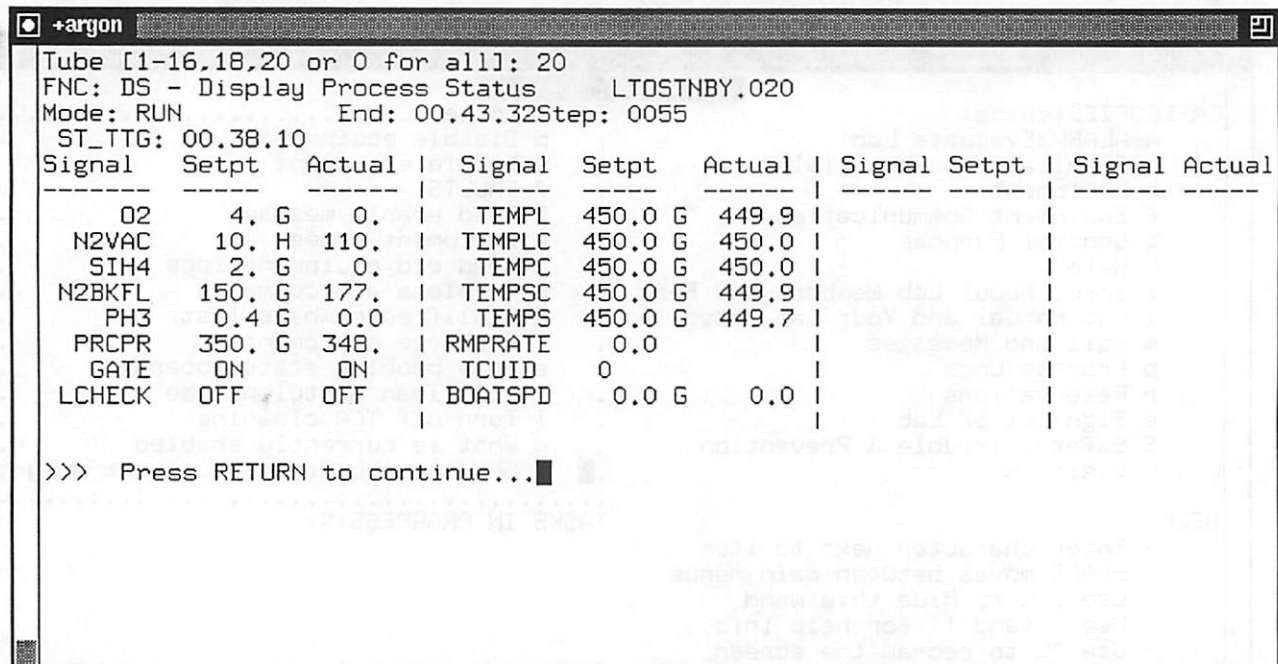


FIGURE 5 Tystar furnace status

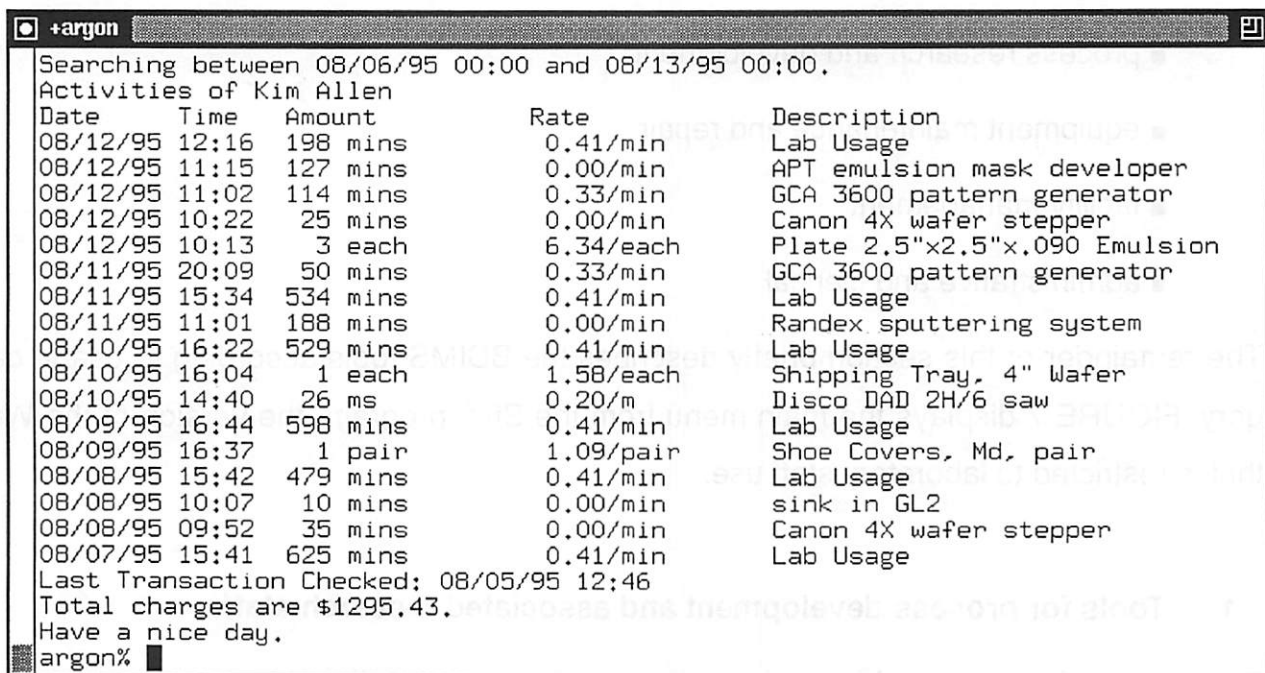
2.6 An example of a typical BCIMS client at work

This section describes how a typical individual conducting research in the lab (named Leslie) interacts with BCIMS. She will complete a furnace run, do some analytical data collection, and use a CAD program to produce a mask for photolithography.

Leslie logs into the lab, then proceeds to a terminal located near the Tylan furnace. She enables Tylan12 (one of the furnaces). Since Leslie is qualified to use the equipment, the hardware is unlocked for her use. A message is displayed from a previous labuser indicating that the boat loader for Tylan12 is sticking. Leslie initiates the furnace run and a process to collect data from the furnace into files for later analysis. Leslie plans to use the flatgag next, but did not make a reservation. Using an equipment control menu selection and the reservation subsystem she determines that the equipment is not in use and that there are no reservations during the time slot she requires. The furnace run will take hours so Leslie "hides" her Wand login at this terminal, which frees the terminal for use by others. She proceeds to

the area in the lab where the flatgag is located and logs in to the nearest terminal. Her Wand is re-displayed and the **In Process** portion of the Wand display lists the furnace data collection process that Leslie left running.

Leslie enables the flatgag, starts a data collection process for it, scans the wafer and then stops the data collection process and disables the flatgag. Now she leaves the clean area to go to a CAD station and finish her mask. Throughout the day she can check on the progress of her furnace run from the CAD area, her office, or from anywhere on the network. During the time that Leslie was in the clean room portion of the lab BCIMS has kept a log of her activities. A time-stamped equipment usage record is used to automatically produce accounting records so that the appropriate account may be billed for equipment use. A Wand menu option allows Leslie to review her charges. FIGURE 6 shows a typical user's activities and charges for the month.



Date	Time	Amount	Rate	Description
Searching between 08/06/95 00:00 and 08/13/95 00:00.				
Activities of Kim Allen				
08/12/95	12:16	198 mins	0.41/min	Lab Usage
08/12/95	11:15	127 mins	0.00/min	APT emulsion mask developer
08/12/95	11:02	114 mins	0.33/min	GCA 3600 pattern generator
08/12/95	10:22	25 mins	0.00/min	Canon 4X wafer stepper
08/12/95	10:13	3 each	6.34/each	Plate 2.5"x2.5"x.090 Emulsion
08/11/95	20:09	50 mins	0.33/min	GCA 3600 pattern generator
08/11/95	15:34	534 mins	0.41/min	Lab Usage
08/11/95	11:01	188 mins	0.00/min	Randex sputtering system
08/10/95	16:22	529 mins	0.41/min	Lab Usage
08/10/95	16:04	1 each	1.58/each	Shipping Tray, 4" Wafer
08/10/95	14:40	26 ms	0.20/m	Disco DAD 2H/6 saw
08/09/95	16:44	598 mins	0.41/min	Lab Usage
08/09/95	16:37	1 pair	1.09/pair	Shoe Covers, Md, pair
08/08/95	15:42	479 mins	0.41/min	Lab Usage
08/08/95	10:07	10 mins	0.00/min	sink in GL2
08/08/95	09:52	35 mins	0.00/min	Canon 4X wafer stepper
08/07/95	15:41	625 mins	0.41/min	Lab Usage
Last Transaction Checked: 08/05/95 12:46				
Total charges are \$1295.43.				
Have a nice day.				
argon%				

FIGURE 6 Labuser activity and charges review

3.0 Tools for the Microlab's maintenance and research staff, and CIM researchers

This section briefly describes the applications that are used primarily to maintain and run the Microlab. Some of the tools are also used by research staff working on process development and CIM researchers working in various areas such as Work In Progress (WIP), production process management, and statistical process control. The tasks involved in running and maintaining the lab, process development, and CIM research require an extended set of operations. Many of the tasks require special privileges such as access to subsystems or features not generally accessible. The Staff subsystem provides privileged access operations to staff members. Staff has the same uniform look and feel as the other text-based menu driven BCIMS software. The tasks can be grouped into four areas of responsibility, which are addressed by BCIMS:

- process research and development
- equipment maintenance and repair
- facility management
- administrative and clerical

The remainder of this section briefly describes the BCIMS tools according to usage category. FIGURE 7 displays the main menu from the Staff program, the version of the Wand that is restricted to laboratory staff use.

3.1 Tools for process development and associated research staff

This section describes tools used by process development staff and researchers doing process development. Process developers require feedback about conditions in the lab, resource levels, and the status of equipment and processing runs. In addition to the capability to collect such data, tools are required to aid in extracting different types of information, or


```

+argon LAB MANAGEMENT : mudie
CATEGORIES(space).....TASKS(space)
. a Accounting                . c Maintenance calendar (qbf)
. c Computers                 . e Edit maintenance manual
. C Equipment Comments       . f FAULTS (abf)
. e Equipment                 . F FAULTS summary reports
. E Equipment Communication   . g Gases database (qbf)
. g General                   . h Hepa database (qbf)
. h Help                      . m Maintenance status board
. i Inventory                 . M Maintenance manual
. I Information About Lab Members
.==>m Maintenance            . p Pumps
. p Power Outage              . P Generate a printout
. q Qualify                   . u Display utilities dependencies
. r Reserve Equipment         . U Utilities database (qbf)
. v Vendors and Purchasing    . x Maintenance Matrix
                              . X Xfig of layouts
.....
HELP                          IN PROCESS(%)
- Enter character next to item
- SPACE moves between main menus
- Use ESC to suspend
- Use Control-C to quit
- Use ? to get help information

```

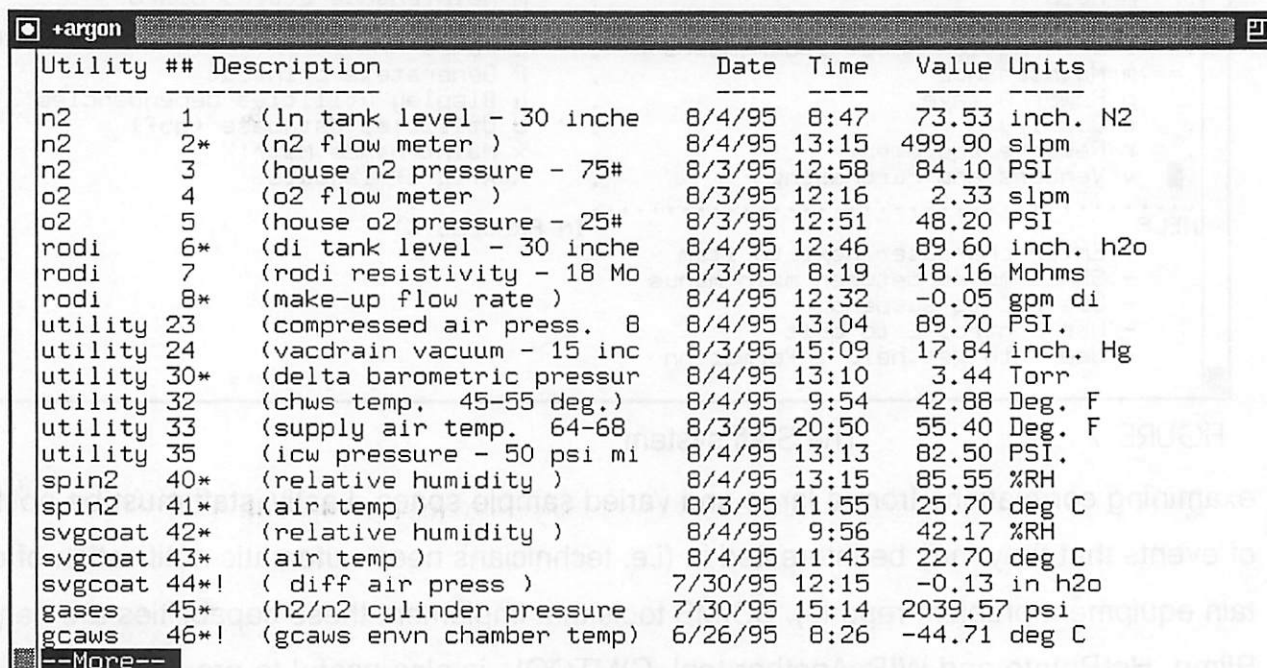
FIGURE 7 The Staff system

examining correlations from a large and varied sample space. Lastly, staff must be notified of events that they may be interested in (i.e. technicians need automatic notification of certain equipment problem reports). BCIMS tools that implement these capabilities are named Blimp, HotPotato and WIP. Another tool, CIMTOOL, is also useful to processing staff but because it is primarily a facility management tool it is described in Section 3.3 in Administrative and facility management tools.

3.1.1 BLIMP

BLIMP is a software/hardware subsystem that consists of sensors, data acquisition hardware, and software [Sharma '88]. Its purpose is to collect data from the sensors and monitor the sensor readings, sending out alarm email when specified conditions are met. The sensors are located at equipment and resource distribution points inside the laboratory. Sensors provide analog or digital input to BLIMP representing temperatures, pressures, flow rates, etc. BLIMP has a GUI (graphical user interface) based on X and VEM, and also an alphanumeric (ASCII) terminal interface. The GUI displays data by means of virtual instrumentation

and the ASCII version displays time-stamped data per sensor along with current settings for alarm conditions. An enhancement of BLIMP, called BLIMPIII, has been developed (by the author and an undergraduate student, Eric Ng) and written in C++. BLIMPIII augments the capabilities of the first version and also incorporates improved performance and methods for handling noisy data. FIGURE 8 shows the output of a BLIMP sensor status display.



Utility	##	Description	Date	Time	Value	Units
n2	1	(ln tank level - 30 inche	8/4/95	8:47	73.53	inch. N2
n2	2*	(n2 flow meter)	8/4/95	13:15	499.90	slpm
n2	3	(house n2 pressure - 75#	8/3/95	12:58	84.10	PSI
o2	4	(o2 flow meter)	8/3/95	12:16	2.33	slpm
o2	5	(house o2 pressure - 25#	8/3/95	12:51	48.20	PSI
rodi	6*	(di tank level - 30 inche	8/4/95	12:46	89.60	inch. h2o
rodi	7	(rodi resistivity - 18 Mo	8/3/95	8:19	18.16	Mohms
rodi	8*	(make-up flow rate)	8/4/95	12:32	-0.05	gpm di
utility	23	(compressed air press. 8	8/4/95	13:04	89.70	PSI.
utility	24	(vacdrain vacuum - 15 inc	8/3/95	15:09	17.84	inch. Hg
utility	30*	(delta barometric pressur	8/4/95	13:10	3.44	Torr
utility	32	(chws temp. 45-55 deg.)	8/4/95	9:54	42.88	Deg. F
utility	33	(supply air temp. 64-68	8/3/95	20:50	55.40	Deg. F
utility	35	(icw pressure - 50 psi mi	8/4/95	13:13	82.50	PSI.
spin2	40*	(relative humidity)	8/4/95	13:15	85.55	%RH
spin2	41*	(air temp)	8/3/95	11:55	25.70	deg C
svgcoat	42*	(relative humidity)	8/4/95	9:56	42.17	%RH
svgcoat	43*	(air temp)	8/4/95	11:43	22.70	deg C
svgcoat	44*!	(diff air press)	7/30/95	12:15	-0.13	in h2o
gases	45*	(h2/n2 cylinder pressure	7/30/95	15:14	2039.57	psi
gcaws	46*!	(gcaws envn chamber temp)	6/26/95	8:26	-44.71	deg C

FIGURE 8 BLIMP Sensor Status output

3.1.2 HotPotato

HotPotato is one of the original applications and is used by process development staff to maintain files of process recipes, record who is working on a process, log status and modifications for recipes, and interface to the UNIX editor, vi, to edit the process recipe files. All files that are opened for editing are locked. All files opened for viewing only are opened using the 'lookat' program, which does not allow any modifications to the file, therefore the file is not locked when being viewed. HotPotato is available from Wand menus under "Process Logs".

3.1.3 WIP

WIP (Work In Progress) is a more recently developed subsystem utilizing the Ingres database and BPFL (Berkeley Process Flow Language), which can be used to specify the information required to design, test, and execute processes to manufacture integrated circuits. [Rowe, Williams, Hegarty '90]. The WIP system can track the state of a process run, and it may monitor status and interact with operators or equipment in the lab concerning the state of processing runs or actions to be carried out at different steps in processing [Hegarty '91]. WIP may also provide the functionality of HotPotato (storing and editing recipes) but operates on recipes stored in the database rather than in flat files. The WIP system is described in detail in a paper entitled "The Berkeley Process-Flow Language WIP System" [Hegarty, Rowe, Williams '90].

3.1.4 Statistical Process Control

Statistical Process Control (SPC) is the most recent area of research applied in the Microlab. The SPC subsystem provides the processing staff with the ability to track and analyze equipment calibration data on-line. A "baseline" process is a standard manufacturing process which can be used in determining whether equipment is operating abnormally. The baseline process is run through the semiconductor fabrication facility periodically. Following each step in the process, analytical equipment is used to collect data describing the wafers in the baseline lot. If the data indicates that the equipment calibration has drifted beyond a normal range, the equipment may be scheduled for calibration or maintenance. Statistical methods are used to determine what is normal for the equipment, based on historical data.

The SPC subsystem is available on the Wand from the Process Logs menu item. The SPC submenus include options to initiate the baseline process, data collection, and analysis as well as options to display performance graphs and print out recorded data. When each baseline process is run if the equipment is found to be operating outside of control limits, automatic email is sent to the equipment technician for that equipment.

The data that is collected at each periodically run baseline process is stored in an equipment history database. The historic data is used to track equipment drift. Statistical methods can be used to predict and prevent future failures by predicting when maintenance and calibration may be required. FIGURE 9 depicts a graphed SPC sample set.

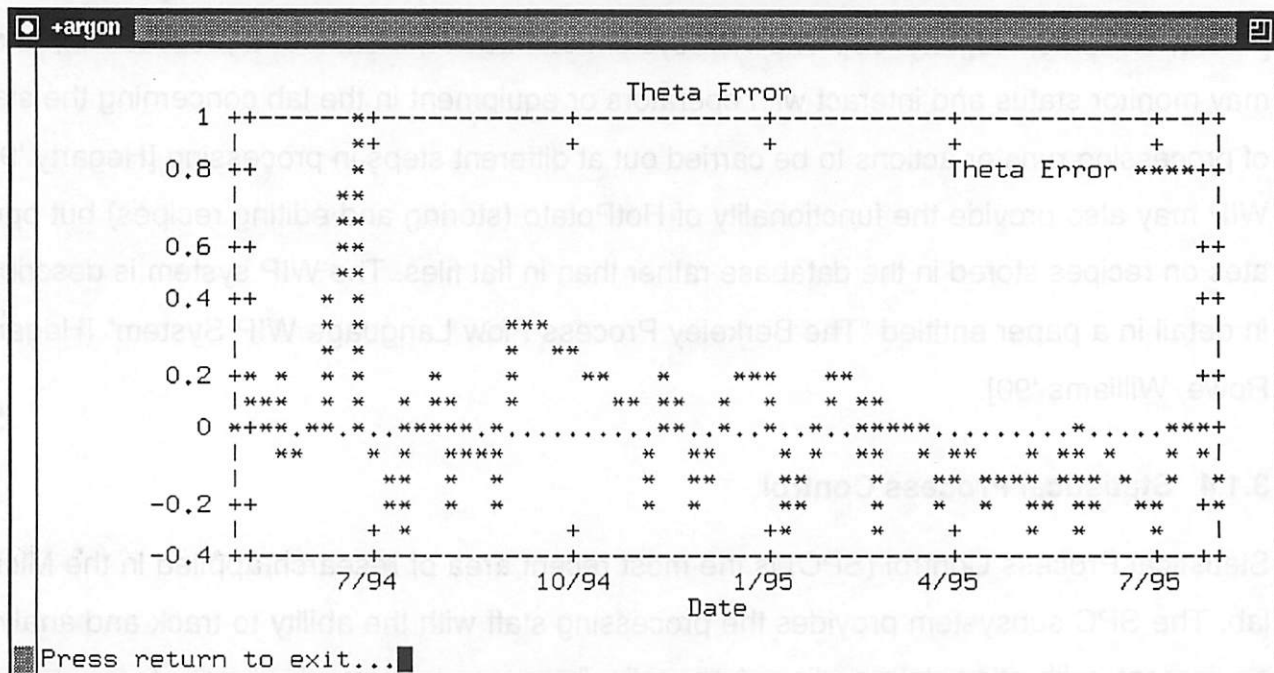


FIGURE 9 SPC output

3.2 Equipment maintenance tools

Several tools have been implemented that aid in equipment maintenance and repair. Many of the tools that were developed earlier on in the project were oriented towards administrative management of the repair and maintenance procedures, and typically supported text-based displays. For example, an earlier project produced **epstat** which focused on the recording of equipment problems or status. Another example is the **comments** program. Comments provides a repository for unstructured text comments or log notes, and display of the past comments. Epstat and comments can be used to manage information about equipment. When problem reports are entered, a copy of the report may be emailed to a specified

technician or equipment operator. After the problem has been cleared, the technician's comments are prompted for and stored. The stored equipment histories are a valuable source of knowledge which characterizes equipment for the technicians. A current-status display is available and the labuser may also review old comments. Earlier projects served to provide experiential data useful in developing the next generation of related tools. A more recent project (**Faults**) focused on recording equipment failures, repairs and maintenance and utilizing failure and symptom data for diagnosis and fault prediction. The experience and labuser feedback gained in developing and working with **epstat** and **comments** was useful in the development of **Faults**. Summary Faults data is still optionally available from the original **epstat** status board display for alphanumeric terminals. FIGURE 10 depicts a typical **epstat** display.

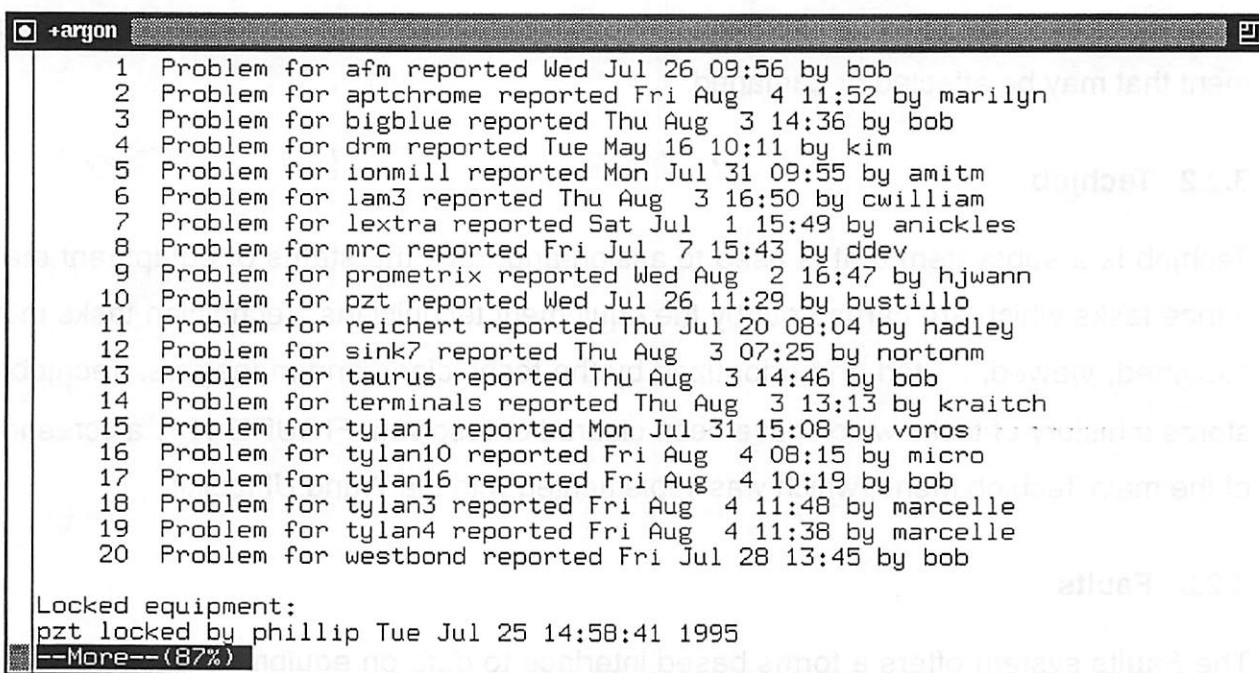


FIGURE 10 The Equipment status board, **epstat**

3.2.1 Gases

Many pieces of equipment use chemical gases in processing. The **gases** subsystem is a forms-based database application that maintenance staff use to track the status, repair,

maintenance and inventory of cylinders of gases. Standard information management operations are provided, (e.g., Append Retrieve, Update, etc.), as well as operations required to track inventory and the location of installed units and their status. For example, Gases has a 'PresUpDate' operation to update the current gas pressure in a cylinder. Other resource specific applications include: **quartz**, **pumps**, **cryopumps**, and **utility**. These are used to manage quartzware for the furnaces, pumps and pump parts, cryogenic pumps, and dependencies among utilities respectively. While each of the other programs manages a specific resource, the utility program (and database tables) not only tracks a single resource (utility lines such as water, vacuum lines and gas feeds) but also stores facility wide data. Dependencies among equipment and utilities are stored in the database. This information can be used to determine which equipment may be affected by utilities. In the event of a utility failure, technicians can use this information to quickly shut down or otherwise attend to equipment that may be affected or damaged.

3.2.2 Techjob

Techjob is a subsystem that is used to assign and track the status of equipment maintenance tasks which are carried out by the equipment technicians. Technician tasks may be assigned, viewed, edited and prioritized by the technicians and managers. Techjob also stores a history of tasks which have been cleared or resolved. FIGURE 11 is a screendump of the main Techjob menu, which was implemented with the Wand UI toolkit.

3.2.3 Faults

The Faults system offers a forms based interface to data on equipment faults and symptoms. The Faults tables are a part of the facility-wide CIM relational database. The Faults tables in the CIM database constitute a formalization of the semantics of preventive maintenance and repair events. Accumulated information is automatically indexed to aid diagnosis of failures as they occur. Utility programs may then produce summaries such as preventive maintenance intervals, mean time between failures, predicted downtimes, performance

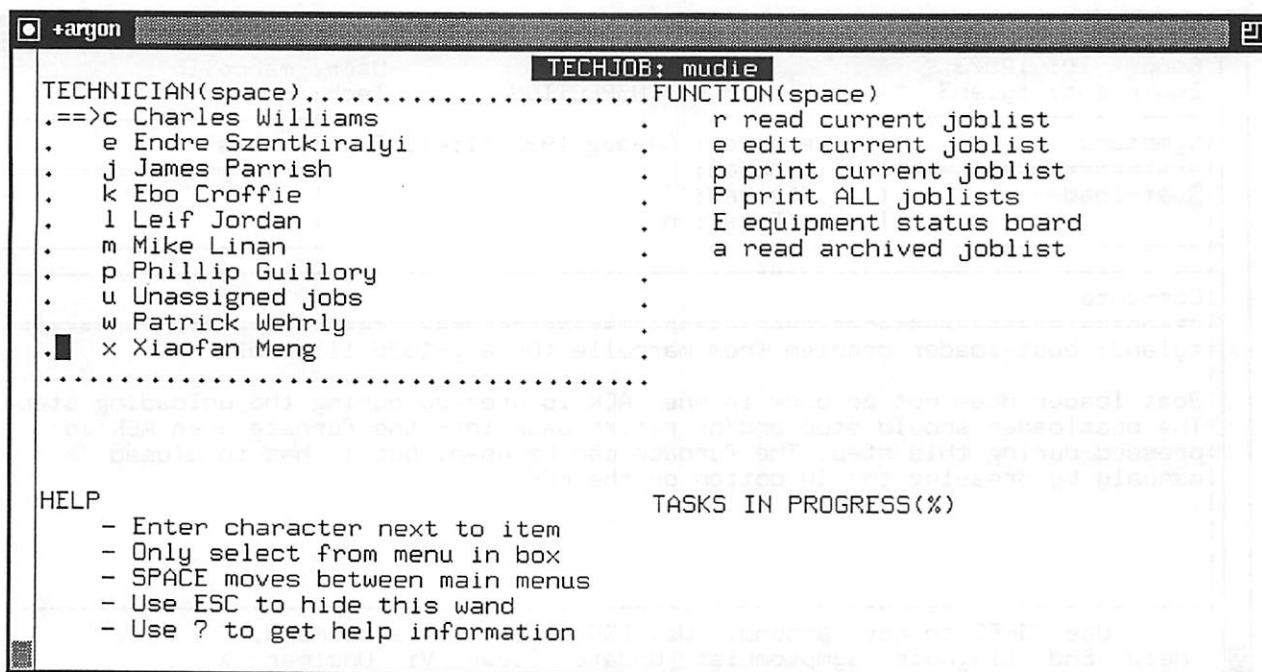


FIGURE 11 Techjob

trends, etc. [Mudie '91]. FIGURE 12 shows a typical problem report form displayed by Faults. The faults application includes a calendar system to track all types of maintenance activities, including scheduled preventive maintenance (PM). Equipment maintenance tasks that are performed periodically, such as changing pump filters and purging gas lines, may be entered into the calendar by technicians. When a scheduled PM task becomes due, an email notification is sent to a specified technician and recorded in the equipment log.

3.3 Facility Management Tools: CIMtool

This section describes tools that support facility management tasks in the Microlab. The tools in this category are designed to provide feedback for the labuser about conditions in the lab. They provide a broad view of laboratory operations and status of the physical resources.

+argon														
Report ID: 12573	User: marcelle													
Equipment: tylan3	Tech:													
REPORT INSPECTION														
<table border="1"> <tr> <td>Symptoms</td> <td>Reported: 04-aug-1995 11:41:57</td> <td>Faults</td> </tr> <tr> <td>boat-loader</td> <td>Diagnosed:</td> <td></td> </tr> <tr> <td></td> <td>Cleared:</td> <td></td> </tr> <tr> <td></td> <td>Fatal: n</td> <td></td> </tr> </table>	Symptoms	Reported: 04-aug-1995 11:41:57	Faults	boat-loader	Diagnosed:			Cleared:			Fatal: n			
Symptoms	Reported: 04-aug-1995 11:41:57	Faults												
boat-loader	Diagnosed:													
	Cleared:													
	Fatal: n													
Comments														
tylan3: boat-loader problem from marcelle (04-aug-1995 11:41:57) Boat loader does not go back in when ACK is pressed during the unloading step. The boatloader should stop and/or return back into the furnace when ACK is pressed during this step. The furnace can be used, but it has to closed manually by pressing the IN button on the ROP.														
Use ^JKFG to move around. Use ESC to execute a command. Help End Diagnose SymptomList Update Clear Vi Unclear >														

FIGURE 12 A typical Faults report

CIMtool is primarily a facility management and administrative tool which incorporates functions that overlap into other areas such as equipment communication, process monitoring, and education [Smith, Rowe '91]. For staff or researchers working in the laboratory it supplies easy access to high level information concerning the state of the lab as a whole (e.g. equipment status, layout of utilities, equipment utility dependency, location of moveable wall partitions). One can also access a hypermedia educational subsystem (IC-HIP) that utilizes recorded video and still images to teach material on semiconductor processing [Schank, Rowe '92].

The initial CIMtool display (FIGURE 13) is a two-dimensional schematic floor plan view of the Microlab. This display, and other views (obtained by zooming in on parts of the map) constitute a visual interface to data describing the laboratory. Status of equipment is encoded using colors that indicate whether the equipment is up for processing, down for short term maintenance, or in some other state. Routing of utilities is also displayed using

color coding and highlighting of the utilities that the labuser may select for display. The informational content of the display is controlled by clicking on the buttons at the lower right of the screen.

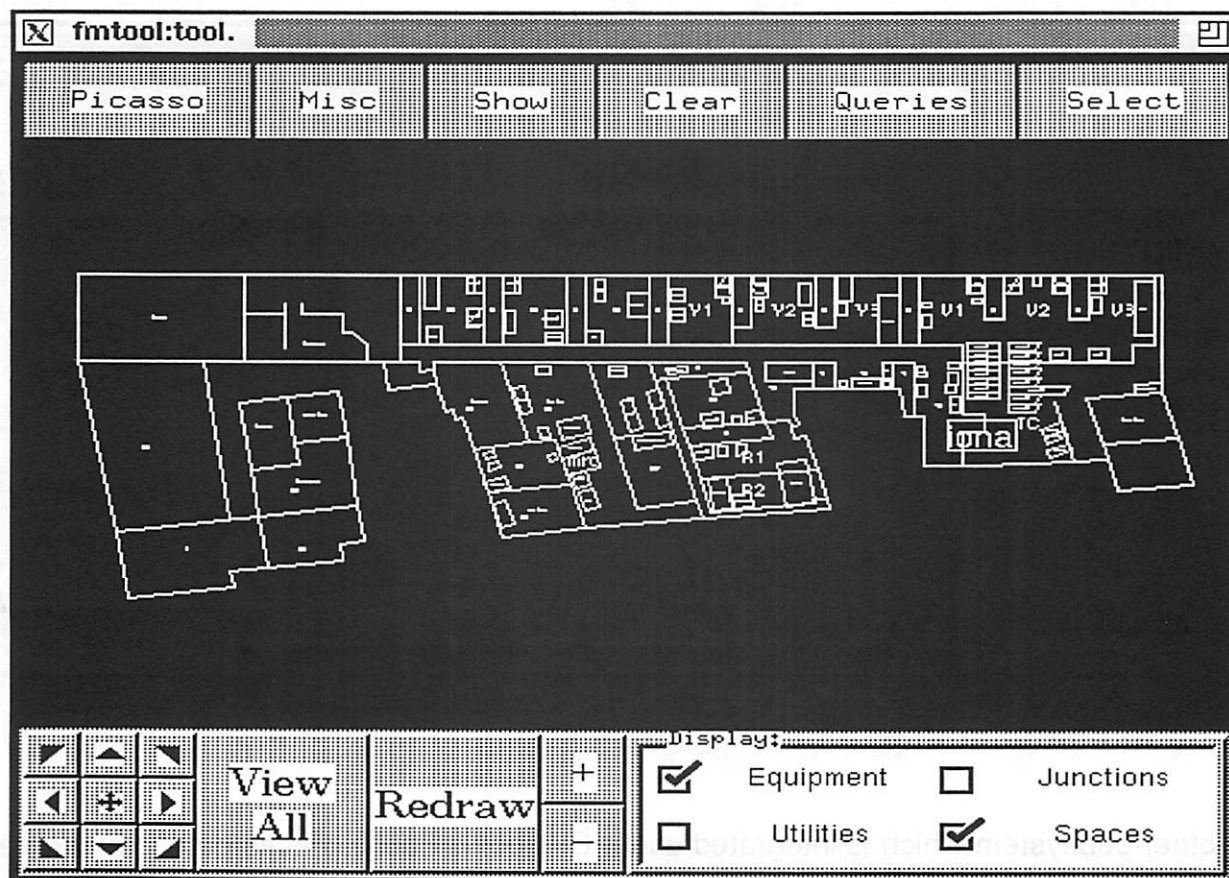


FIGURE 13 CIMtool initial view, a floor plan of the lab

Queries and constraints on queries may also be defined through this GUI. For example, to obtain information on utilities that are connected to a particular piece of equipment one can select the equipment or groups of equipment by pointing and clicking with a mouse on the equipment shown in the map. Any further queries, such as one for a listing of utilities, will select information concerning the selected equipment only.

CIMtool utilizes data from several other subsystems to represent laboratory state. One of these is BLIMP III, described above, which monitors data collected from sensors in the labo-

ratory. CIMtool provides an interactive window based interface to the sensor data that BLIMP III acquires, and supports graphing of one or more sets of sensor data at one time. FIGURE 14 shows the CIMtool interface to the BLIMP III data and a graph of sensor data.

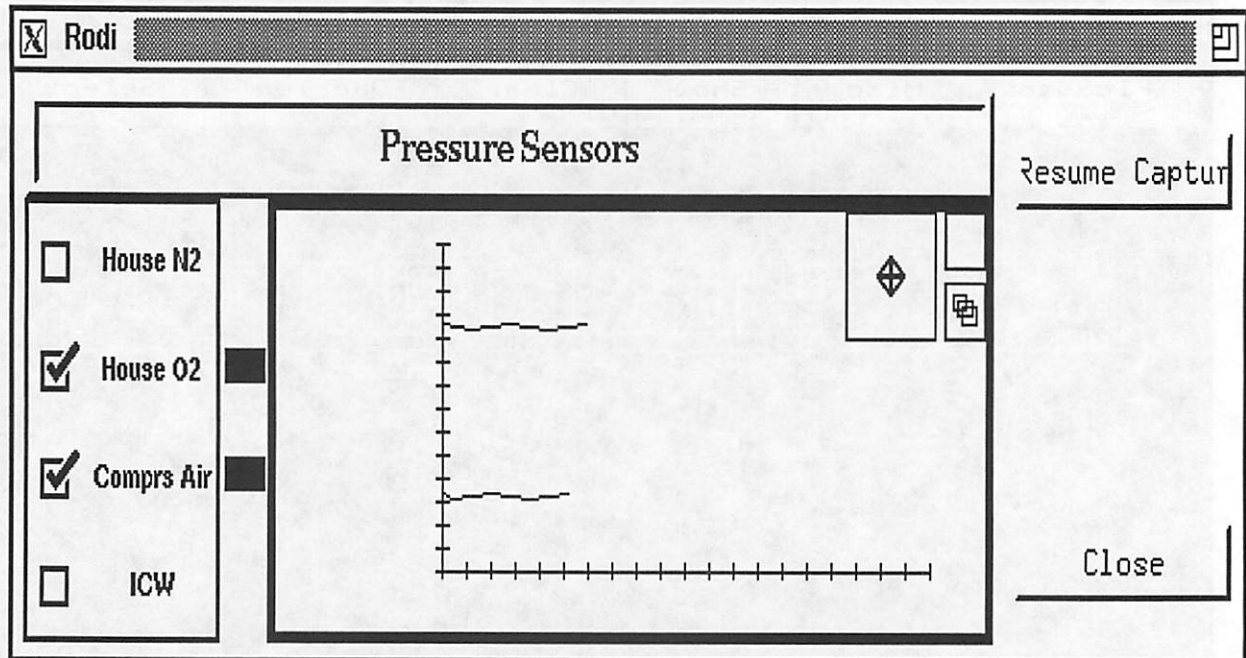


FIGURE 14 The CIMtool interface to BLIMP sensor data

Another subsystem which is integrated under CIMtool is Faults. Faults data is presented to the labuser through a CIMtool GUI. The interface supports an ad hoc query capability enabling a labuser to ask questions about facility, equipment and, processing history data. It is designed so the labuser may develop a query through the use of virtual switches or settings, and does not require any knowledge about the underlying RDBMS or databases in general. Another set of views that can be displayed from the ad hoc query GUI, as applied to Faults data are the equipment "status board" views, which provide a summaries of the facility-wide equipment status. A one-line description of each outstanding maintenance task of equipment problem is displayed. The status board offers overall status at a glance, mirroring more detailed information that is available through the regular Faults problem database interface shown in FIGURE 12.

3.4 The Staff system

A subsystem named the "Staff" includes Wand operations and an extended set of operations dedicated to staff tasks. In general, the staff tasks are more complex and numerous than lab client tasks. For example, the Faults equipment status and maintenance logging tool is most often used by lab clients to report problems. In contrast, the lab staff use the full complement of diagnosis and maintenance tracking capabilities of Faults. The equipment communications subsystem for the Tycom furnaces provides another example. Most labusers use the reduced set of "safe" operations that are made available on the Wand. Qualified staff and researchers are able to use an alternate console mode of the software which allows direct connection and unchecked SECS message passing to the furnace controller.

3.5 Administrative tools

The administrative tools are divided into three main categories, accounting, inventory/purchasing, and clerical. Accounting applications support tasks like keeping track of who our labusers are (**labusers**), how the labusers should be charged (**contracts**), and generation of billing documents (**acct**). Inventory/purchasing support tasks such as keeping track of inventory (**inven**), creating and monitoring purchase orders (**purchase**), and maintaining a vendors database (**vendors**). Clerical applications include operations to create and remove computer accounts (**killacct**, **makacct**, **newacct**), allow labusers to access their billing records on-line, redirect charges, and checkin/checkout inventory items (**chkin**, **chkout**). All of these applications provide report generation, and in some cases automatic notification of changes to one or more administrative staff members (by email).

4.0 System Design, Architecture, and Implementation

This section describes the overall system design and architecture of BCIMS. Additionally, the software and hardware subsystems are described. Section 4.1 describes the overall software/hardware system design and architecture. In Section 4.2, the software design and implementation is described. An exhaustive description of engineering and implementation details for all software subsystems is outside the scope of this paper. Instead, this section will cover at least one representative example from each class of laboratory operations (listed in the Introduction section). In each case the most significant design issues or goals are described, followed by a discussion of the resulting architecture, engineering and implementation of representative subsystem(s). Section 4.3 presents a discussion of the hardware architecture. Entity relationship diagrams for the database objects described in this section may be found in Appendix 1. A listing of the constituent executables of BCIMS may be found in Appendix 3.

4.1 BCIMS system architecture

Two-Level Architecture

Existing manufacturing systems are difficult to analyze because it is difficult to correlate data from different manufacturing areas. Even closely related data sources such as lot, process and test data are often stored in incompatible, separate databases. A primary goal of BCIMS is to use a logically integrated, physically distributed database so that data from all sources in the system may be fused and correlated including design, marketing and sales, scheduling, WIP, equipment control, transport control, process monitoring and testing.

The system architecture of BCIMS was designed to allow integration of dissimilar sources of data while minimizing the number of layers, in order to reduce complexity [Hodges, Rowe, Spanos '89]. BCIMS has a two-level architecture made possible because rather than designing the system so that all subsystems which need to exchange data communicate

directly, each subsystem instead communicates through a mediating subsystem. In this way asynchronous communication and independence of all subsystems in the communication network is achieved. A DBMS is used as the mediator or primary integration vehicle for the system. One reason for the added complexity of older systems is that individual subsystems were communicating directly, and therefore required facilities for knowing how to communicate to several disparate subsystems. Without asynchronous communication and independence, this architecture was not feasible and systems were typically 4 to 6 layers. FIGURE 15 shows a schematic of the BCIMS software/hardware system architecture. The lower level of BCIMS is composed of the embedded software and hardware subsystems that provide the data acquisition, labuser I/O and other local resources for a particular node. In this context, the term node indicates a stand-alone software or hardware system that constitutes a single unit on a network insofar as the capability for independent operation is concerned (i.e., the unit is not a client or completely dependent on any other unit). A node may or may not have its own network address. The upper layer is composed of the distributed network of nodes that are integrated through the relational DBMS. In our system the Ingres DBMS is used on all platforms. However with the addition of Ingres Star a distributed relational DBMS with gateways to other database systems (or some other RDBMS gateways system), databases from other vendors can also be integrated quite easily. Thus one of the roles of the RDBMS can be thought of as that of a bus or a network through which all other subsystems communicate. FIGURE 16 depicts the integration of BCIMS through the UI, for the labuser, and the RDBMS for internal operation of the system.

4.2 Software system design

Modular software design

The concept of modularity is central to the design of BCIMS software. The software system is a collection of integrated subsystems. Each subsystem is in turn composed of modules of code archived into code libraries. Each subsystem provides facilities for a class of tasks iso-

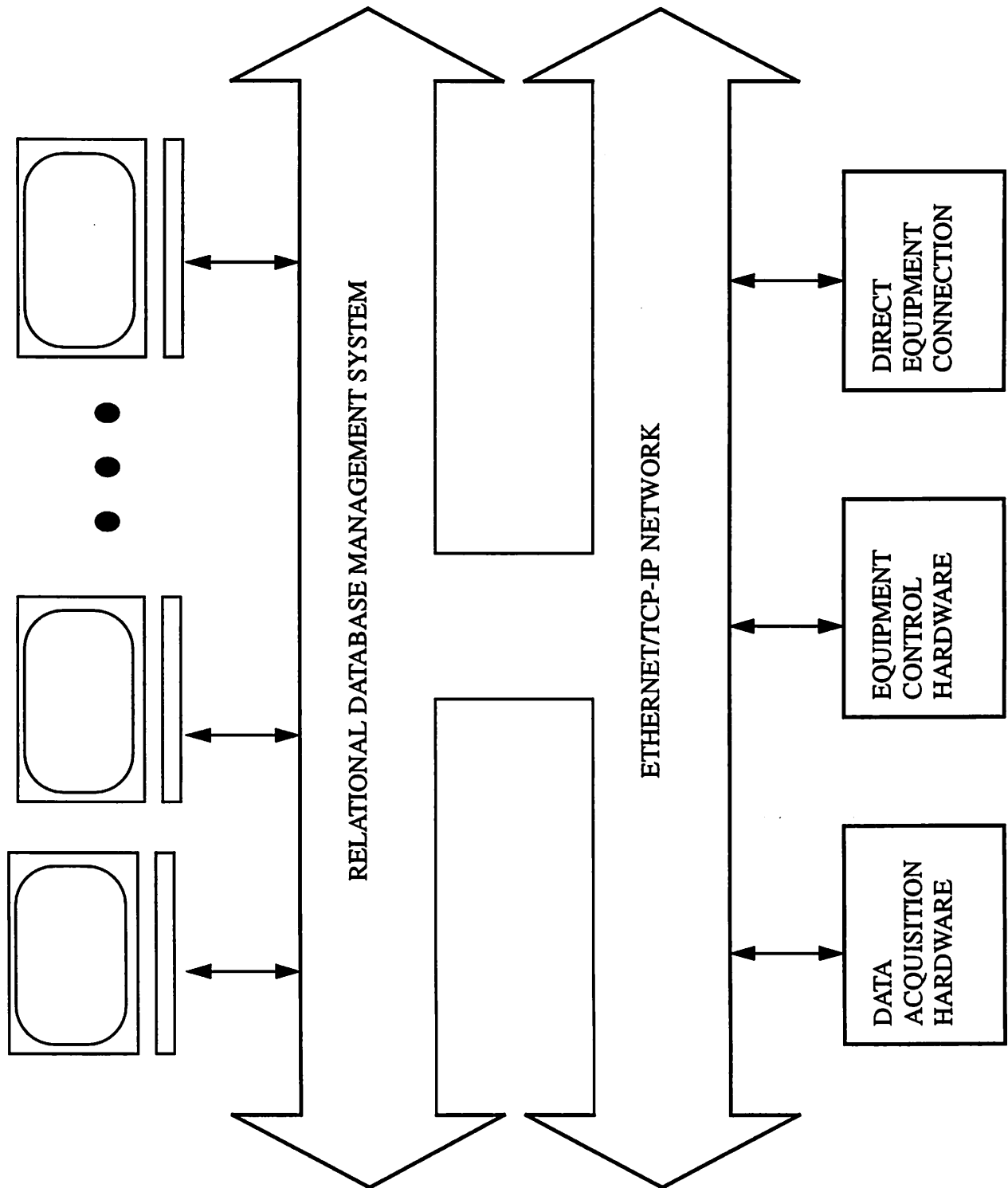


FIGURE 15

Two-level software/hardware system architecture

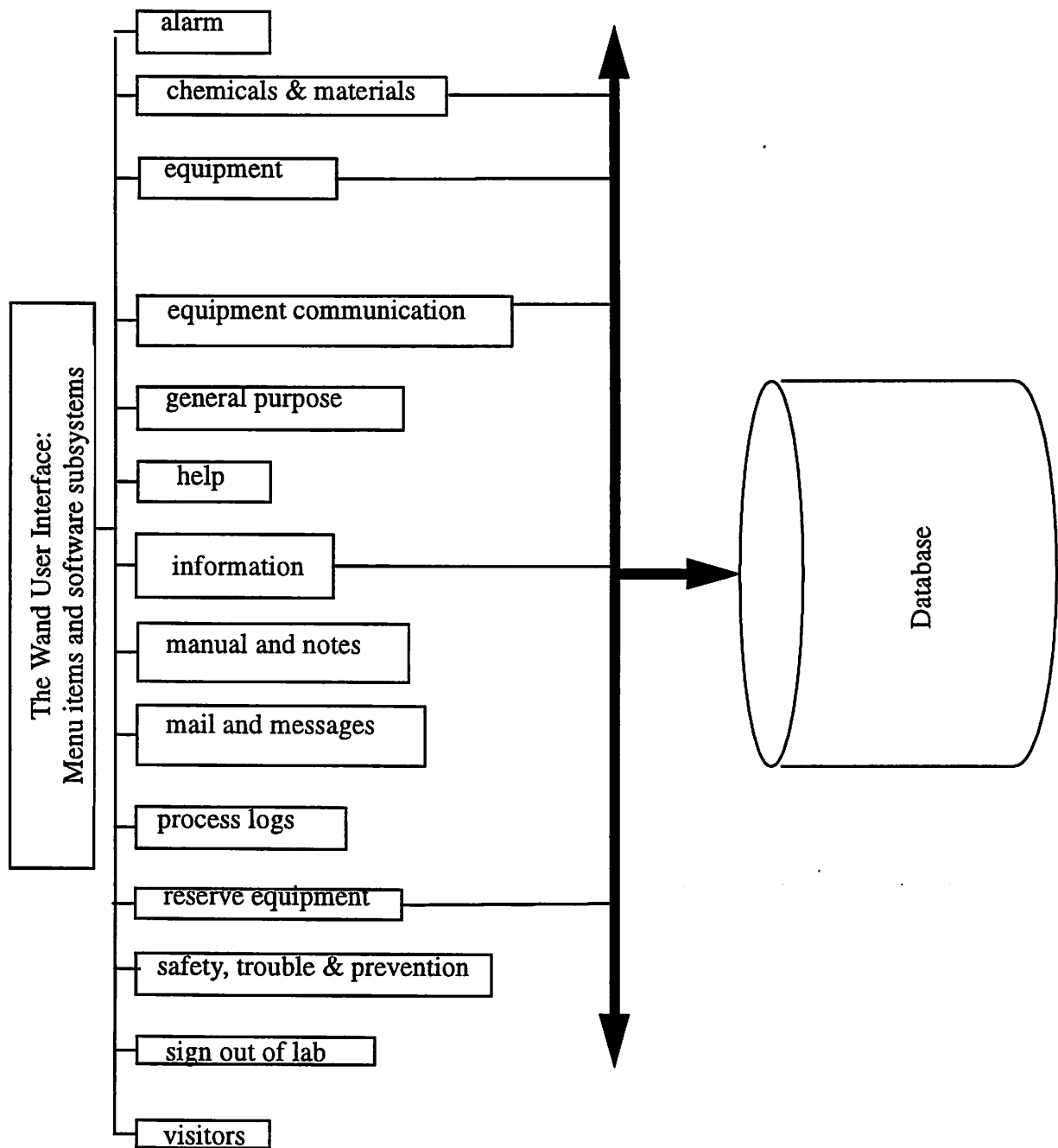


FIGURE 16

Relationships between wand, BRDS, and subsystems

morphic to a class of lab operation. As previously mentioned, the classes of lab operations are:

- process control
- equipment management
- facility management
- administration

We may also speak of functions that are classified according to the operations internal to the system. In BCIMS, these classes are:

- user interface
- application specifics
- data management and subsystem integration
- specialized protocol and or hardware interface

The architecture of the subsystems can be viewed as layered, with each layer implementing functions belonging to one of the classes of internal software operations. FIGURE 17 depicts subsystem architecture. The UI Layer is at the top, followed by application specific code, which in turn calls functions in a data management and subsystem integration layer. Lastly, if specialized protocol or hardware is a component to the subsystem, it is implemented in the lowest layer. One or more code libraries are associated with each type of layer, with the possible exception of the application specific layer. Grouped according to the layers just described, the following libraries have been developed.

- UI layer: **menulib**, **keymaplib** (menu and key mapping functions)
- application specific layer: **faultlib**, **blimplib** (for the Faults and BLIMP subsystems)
- data management and integration layer: **objlib** (database object methods), **wizlib**

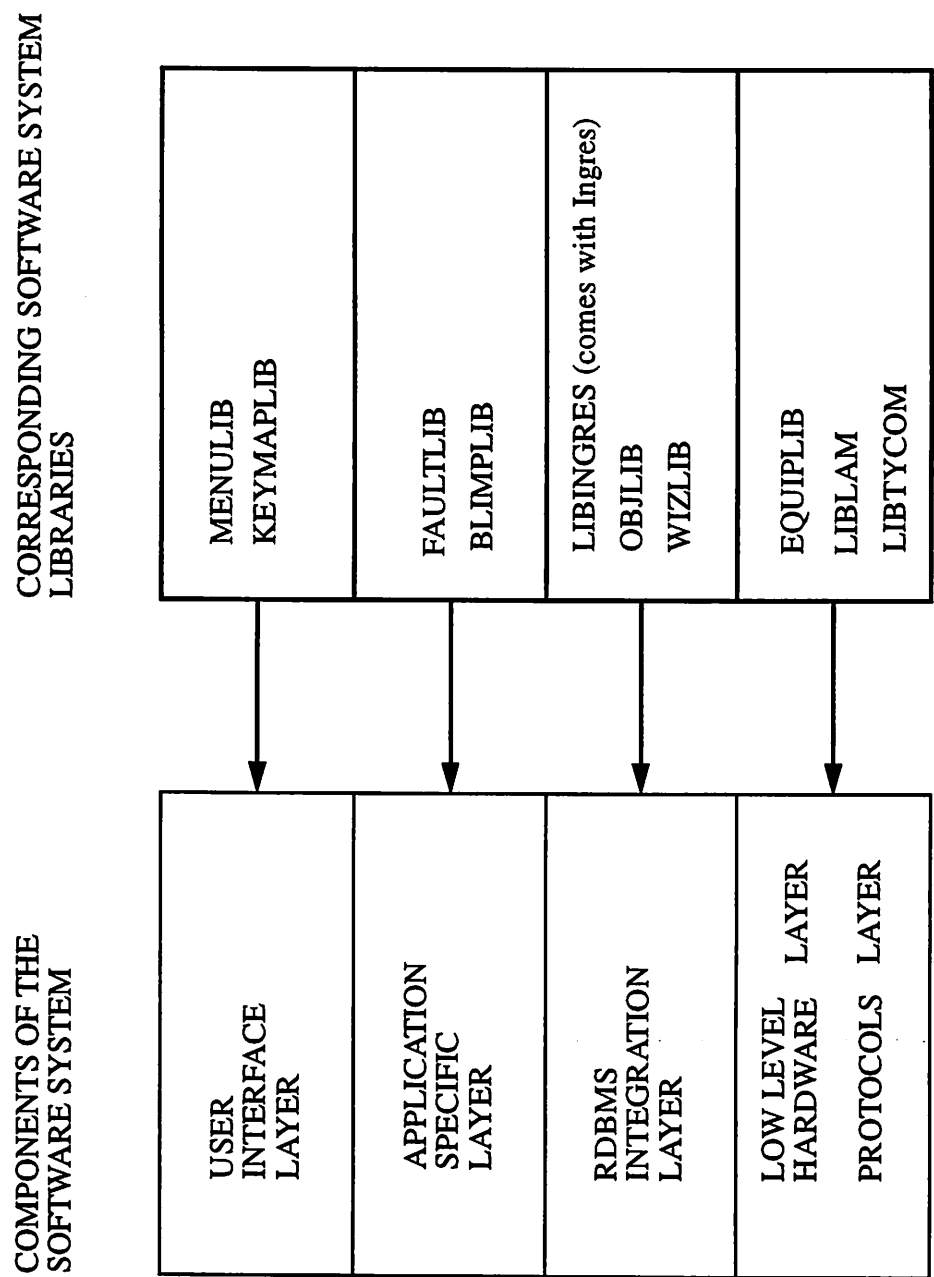


FIGURE 17 Subsystem layers and corresponding code libraries

(integration functions and utilities, interface functions to **libingres** distributed with Ingres)

- equipment protocol and hardware interface layer: **equilib** (generic functions for equipment interface), **tycomlib**, **lamlib** (SECS protocol equipment interfaces)

In the case of the SECS protocol libraries, a generic prototype library has been developed. The prototype library is cloned to create each new equipment specific interface.

When viewed as a dynamic collection of communicating processes the system software architecture is a collection of distributed agents, most of which act asynchronously. Each agent operates within a small scope of responsibility and is designed under the assumption that the run-time environment is deterministic. Some of the subsystems are implemented as client-servers. The client-server model is one in which two or more processes communicate; at least one server process provides one or more resources and one or more clients consume the resource(s) [Leffler, McKusick, Karels, Quarterman '89]. A client-server unit may be composed of one or more servers, one or more clients and one or more agents which are not strictly servers or clients but perform some service(s) such as communicating with hardware or downloading a program into a remote subsystem, usually asynchronously. From the User Interface (UI) perspective, it is the Wand that integrates all of the running subsystems for the labuser. However, the Wand does not perform any real, operational or internal integration of subsystems.

The BCIMS Relational Database Subsystem (BRDS) is composed of the Ingres relational database management system (RDBMS) and the BCIMS interfaces to the RDBMS. It is the primary vehicle for the internal operational integration within BCIMS, and it also helps to make the system more modular. FIGURE 18 depicts the major BCIMS subsystems and their relationship to one another. This architecture is designed to allow all of the subsystems to operate asynchronously and without knowing details about other subsystems. All of the subsystems create and/or use database data. Although a subsystem or database object may have a dependency upon the existence of some data produced by another subsystem, there

is no direct contact required between subsystems. The subsystems are effectively independent. The graph and network theoretic properties of the star shaped topology for communication are well known. The reader may refer to any elementary text on graph theory. Essentially, such topologies may be more efficient than those which are connected graphs, as long as the mediator(s) are not a resource bottleneck. FIGURE 19 depicts the graph or network topology of BCIMS. The RDBMS is the central node or mediator for all of the nodes or subsystems. Each node need only have the capability to communicate with the mediator node (the RDBMS). Therefore each node may be designed and implemented as a completely separate module. The mediator is not a bottleneck in this case because it has the capability to communicate simultaneously and asynchronously with multiple nodes. The number of connections that may be open to the RDBMS at one time limits the bandwidth of communication through the database. Currently the BCIMS RDBMS may have hundreds of connections open at any one time - far more than is typically required. Capacity can be increased through addition of distributed database nodes. Bandwidth may be improved through simple RDBMS configuration, and/or the addition of memory, semaphores, and OS parameters. In addition, the database design and database application design may have an effect on communication bandwidth. However, a more complete discussion of the factors affecting performance tuning is beyond the scope of this document. BCIMS subsystem communication performance has been satisfactory based on a fairly standard Sun Server based cluster which has not required an unusual amount of memory or other platform resources.

In addition to the performance and complexity related benefits this architecture has afforded, the RDBMS also impacts robustness and security. The transaction processing, security, backup and historical data discovery, and event trigger facilities built into the RDBMS have greatly enhanced BCIMS.

4.2.1 The Wand HCI architecture and implementation

The Wand's user interface is its most visible feature, but the Wand actually has several different faces. The Wand is a user interface that represents on the screen the separate sub-

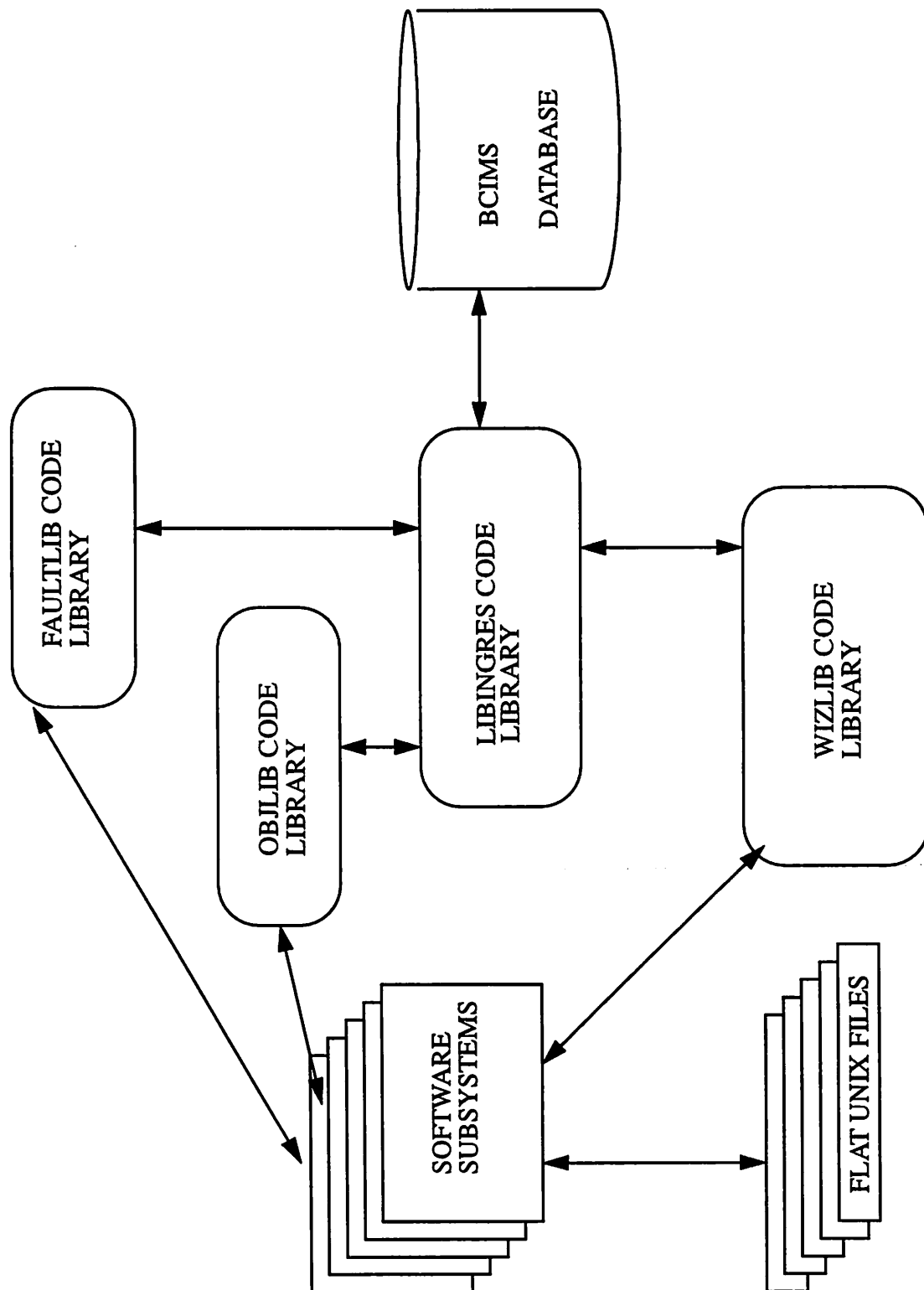


FIGURE 18

Relational Database System Access

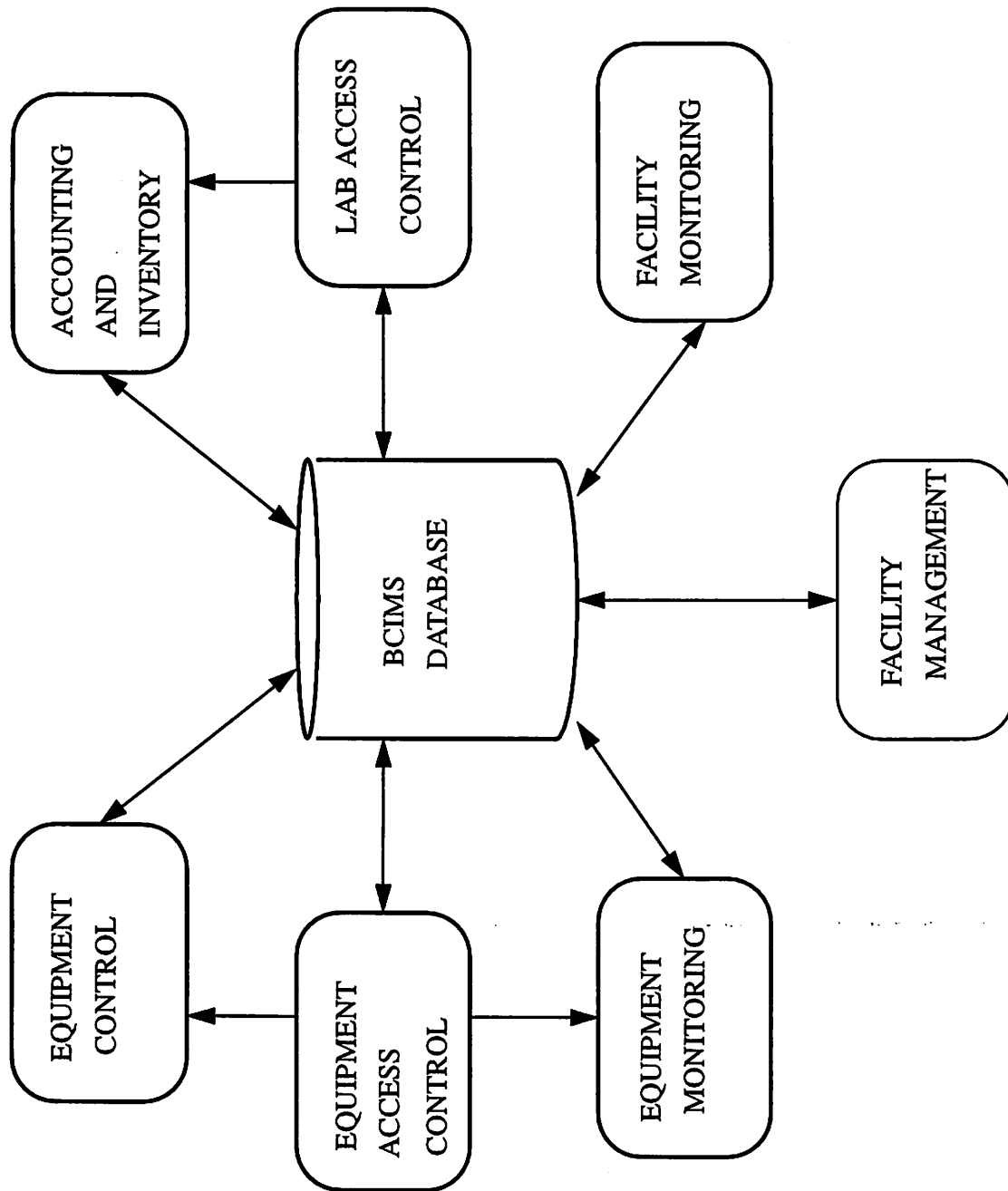


FIGURE 19

Database software subsystem integration

systems of BCIMS in a cohesive manner. The Wand is also a toolkit for engineering, for building menu driven UI's that offer a single highly structured "look and feel". Lastly, the Wand is a group of intercommunicating processes and ancillary ASCII text files which completely hide operating system and hardware topology from the user. It is more than a user interface because it defines the entire computer-human interaction for the system. This section will discuss the design, architecture, and implementation of the Wand in terms of these three facets. The discussion of the third facet of the Wand does not assume special knowledge about UNIX. For a more complete technical detail which does assume a knowledge of UNIX at the system level see Appendix 2. In the following text, all first occurrences of UNIX or BCIMS system objects and process names are indicated by bold underlined type, and may be looked up in the UNIX or BCIMS manual pages. Entity relationship diagrams for the database objects described in this section (and others) may be found in Appendix 1.

The Wand user interface

The Wand is a user interface that represents functions of BCIMS to the user in a menu driven command format. Design parameters for the Wand UI were based on a profile of the average user and the particular needs of a CIM environment.

The average IC fabrication facility or laboratory computer user is at the naive level. Therefore, the most singular requirements for representation of operations in a CIM environment are that the representations be designed for easy use with a minimum learning curve, and that the system be flexible and easily modified while on line. The first requirement is particularly necessary in a manufacturing environment where mistakes are costly or dangerous. The second requirement is driven by the fact that downtime in a manufacturing environment is very expensive, and the environment changes frequently.

The design and implementation of the Wand reflects these requirements. Appropriate to an interface for naive users, memorization of commands is not required. The display of all necessary information needed to select commands is persistent. The current menu and sub-

menu are always visible whenever the screen is not displaying output data. The most basic help commands are always visible whenever menus are being displayed. A list of processes that the user may have running is also persistently displayed, as a reminder. Lastly, all system specifics are hidden from the user. This last implementation feature required an innovative design and some very skilled engineering because it has been executed in an unusually complete manner. Typically, systems which hide system specifics conceal the operating system's command language from the user. The Wand UI goes further and additionally hides the entire system from the user, including the concept of one or more host computers. To a Wand user, there is a single login session regardless of which machine(s) the user may actually be using. Furthermore, this single continuing session is available at any terminal or workstation in the system.

The design goal to create a flexible system that can be modified without system down-time has been realized through the use of external configuration files. Two plain UNIX configuration files are required for each application based on the Wand UI. One of the files contains strings that specify the appearance of each menu selection (**Menudefs**), and the commands and optional prompts for arguments that will be executed when the particular menu item is selected. The other file contains strings that specify optional special key bindings (**Keydefs**). The Wand based program does not have to be restarted when changes to either of these two files are made, therefore changes may be made while on-line.

The Wand UI toolkit

The Wand UI may be used as a toolkit in the development of any application program. In addition to the Wand program, it has been applied to several other BCIMS subsystems including **Staff**, **Techjob** and **Equipment Communications**. Each of these is a BCIMS subsystem and a menu selection on the Wand program's menus, as well as a stand-alone system that may be run from the UNIX command line. The toolkit may be used by any software requiring the facility for mapping keyboard input to specific execution statements. One of the

fields in the menu configuration file (**Menudefs**) is used to indicate what type of execution is required. For example, an “x” in this field indicates a binary executable and an “s” indicates a bourne shell executable. Execution of binary files and several types of shell and script files are supported. The toolkit code is implemented as two collections of short modular functions, one for menuing (**menulib**) and one for key mapping (**keylib**).

The Wand computer-human interaction system

In this subsection the Wand computer-human interaction system (CHI), architecture and implementation is described. While basic knowledge of UNIX or some other multiple process operating system is assumed, we do not assume special knowledge about UNIX. For a detailed technical description, assuming a knowledge of UNIX at the system level, see Appendix 2.

The Wand CHI (WCHI) system is implemented as a system of intercommunicating processes and ancillary ASCII text files. Each process performs a different task required in support of each and every WCHI login shell. FIGURE 20 depicts the WCHI processes and their relationship to one another. The system of cooperating processes depicted is required to allow a continuous login session to be transferred from one terminal or workstation to another. The unaugmented UNIX operating system does not support moving a login session between terminals, or suspension of a login shell. The only way to free a terminal for a new user login shell is to discontinue the session by logging the user out i.e. terminate the login shell process. The user may log in at another terminal, but it will be a new login - not a continuous session. The benefits of a continuous session are that all work in progress or processes that the user has running may either continue to run, or be suspended to be awakened later. An additional benefit is that the multi-cluster network topology is hidden from the user.

How has the appearance of a single continuous session across multiple platforms been achieved? A method was developed which disassociates a login session from the terminal it

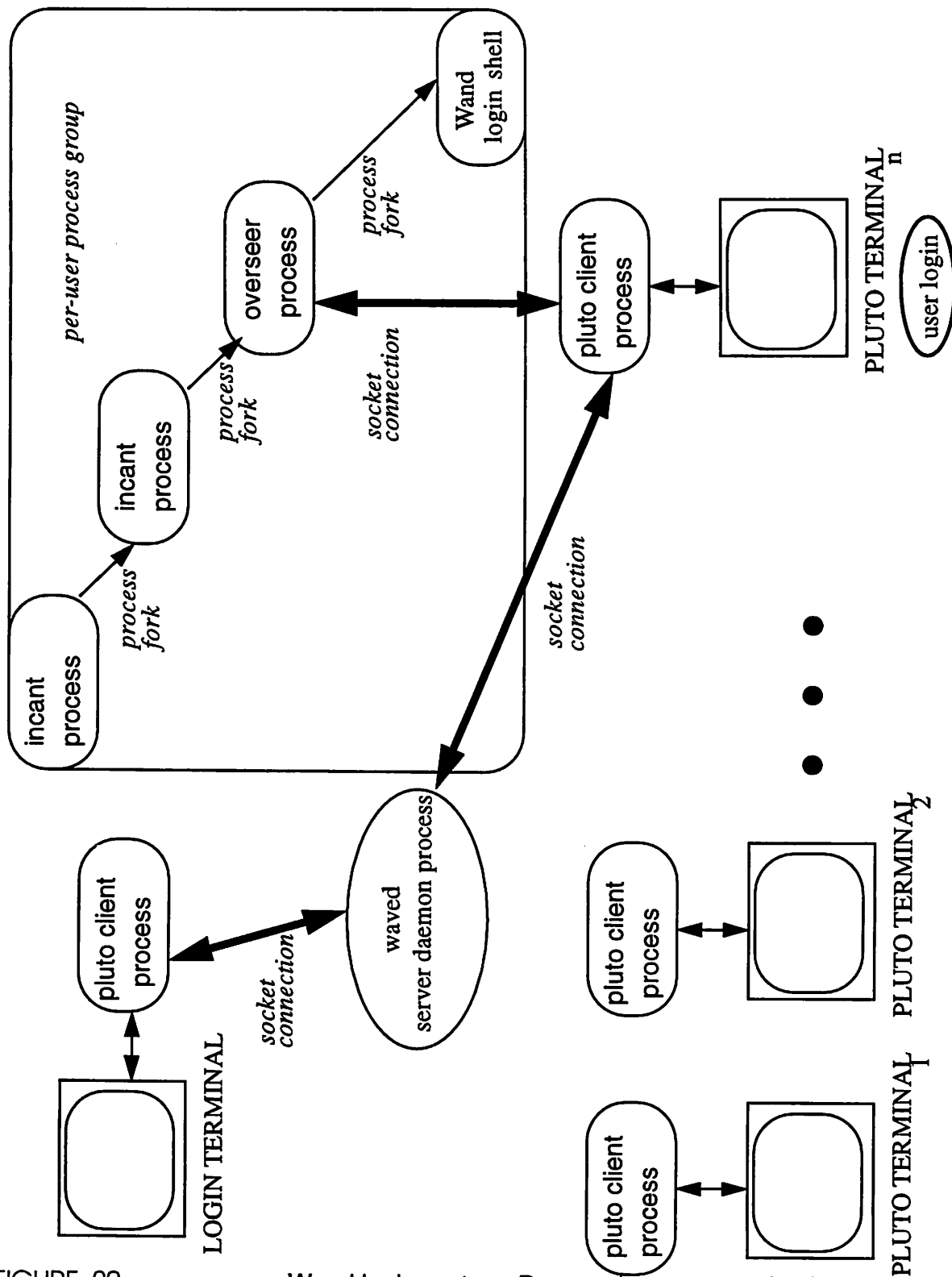


FIGURE 20

Wand login system: Process intercommunication

was initiated on. At host boot time one process named **pluto** is started for each terminal in the system that has been configured to be a Wand system terminal. Additionally a single process which runs continuously named **waved** is started. These processes continue to run as long as the host machine is operating in multi-user mode. A plain ASCII text file is used to configure the system - the file is read by a start-up script at boot time that starts all BCIMS processes. The pluto processes take the place of a UNIX process which is normally run (one for each terminal) (called a **getty**). One pluto process is dedicated to each new login at a terminal. When the user logs in to a terminal, the pluto process **connects** to the waved process to request information about the computer account name that the user has entered at the login prompt. The waved process is a special kind of process called a server daemon. Typically a server daemon provides one or more system resources or information to other processes called clients, at the client's request. The waved process provides information about who is logged in, whether a user is actively using a particular terminal, and other information required to manage the user logins. If the user does not have an existing login session then waved creates a login session and then returns the needed information about the new session to the pluto process. If the user already has a continuing login session then the waved process returns information about the existing login to the pluto client process. The waved server is able to do this because it stores information describing the status of a user login session.

To create a login session, waved starts a new process called **incant** which is at the root of the hierarchy of processes depicted in FIGURE 20 as the "per-user process group". This group of processes performs initialization and then starts a login shell for the user. Thereafter most of the group of processes continues to run as long as the user has an ongoing login session. The per-user process group is a dynamic structure of processes designed to support the two features listed earlier, which are not supported by UNIX, i.e. the ability to move the user login shell between terminals, and suspend and awaken the login shell. These two features are required to allow the user's login to follow him/her to different stations in the lab.

When a user moves to a new terminal and enters his computer account name at the prompt, the pluto associated with that terminal connects to waved and asks for information about the user. The pluto receives the Process Group Identifier (PGID), (a number used to identify the user's login) from waved. Then pluto uses the PGID to connect to one for the processes in the per-user-process-group, named **overseer**. The overseer process is the direct parent of the user's login shell process. Once overseer has **accepted** a connection from a pluto the user's login shell can be started or restarted at the terminal that the pluto is associated with.

If the user had left a login shell active on a different terminal, then that same login shell is disconnected from the old terminal and transferred to the new terminal. This frees the user from having to remember to disconnect from the terminal. Alternately, the user may explicitly temporarily disconnect (or "hide") the login shell, suspending the login process and freeing the terminal for later use by the same user or others. Whenever a user's login shell is suspended, all processes for which it is appropriate are kept running and others are suspended. Thus the user may move freely about the work area unhindered by any system specifics.

4.2.2 The BCIMS Relational Database System

This section briefly describes the BCIMS database design and implementation. No knowledge of database design or relational theory is assumed.

The BCIMS Relational Database System (BRDS) consists of the INGRES relational database management system (RDBMS) [INGRES '91], the BCIMS software interfaces to the RDBMS, and various plain ASCII text files. The text files are used as intermediate data storage areas for data to be up/downloaded to/from the database. The relationship of the BRDS components to one another is depicted in FIGURE 21. Entity relationship diagrams for the database objects described in this section (and others) may be found in Appendix 1.

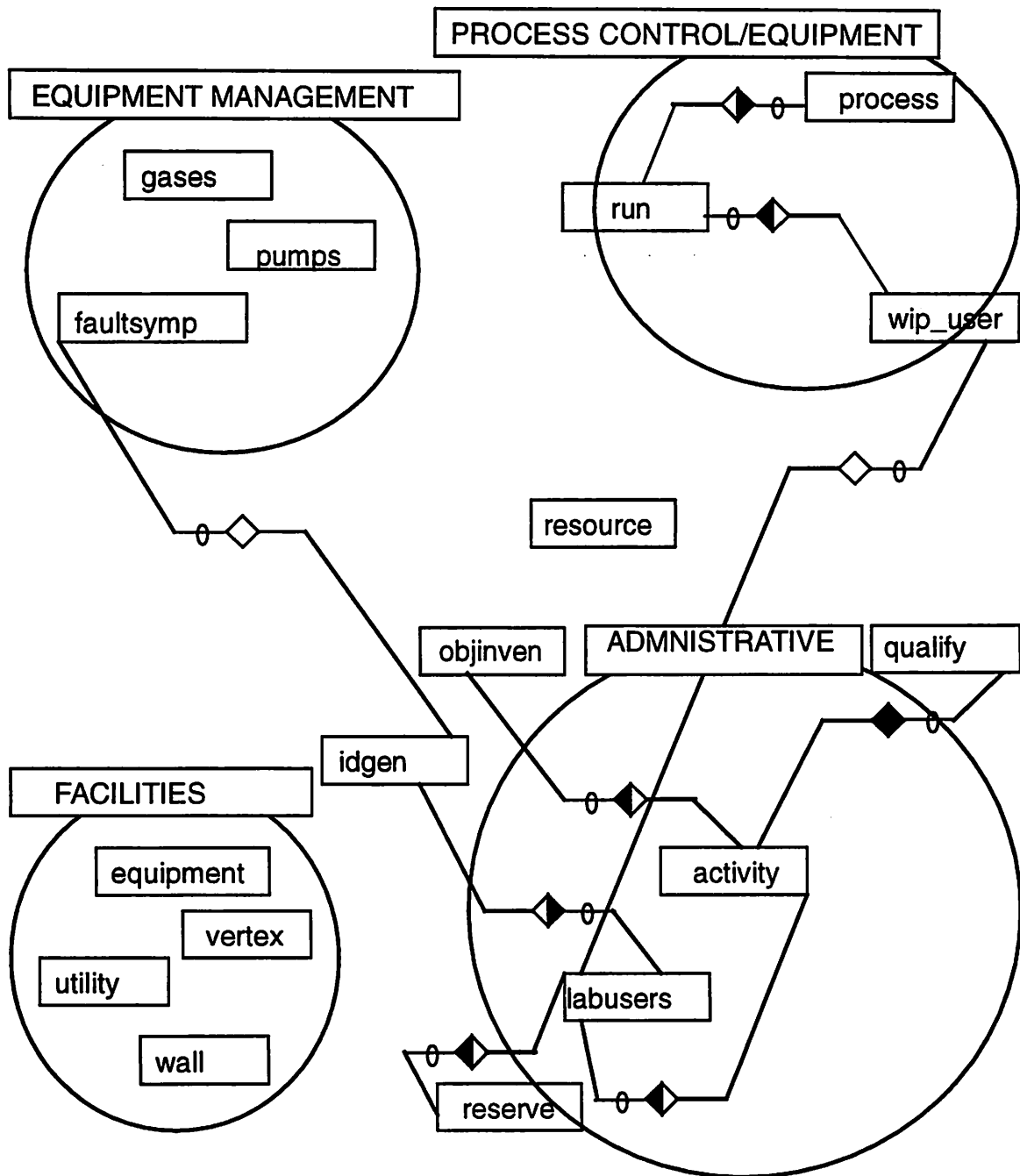


FIGURE 21

CIM Database: Top level entity-relationship diagram

Database design

In addition to the typical issues of all database design (which we will not expand upon here), the design of BRDS required addressing the following major issues:

- What kind of data structures are appropriate for a CIM environment?
- What sort of data types are required to represent CIM information?
- How can integration of heterogenous subsystems be integrated through an RDBMS?

The design for information representation in BRDS is object oriented in that classes of objects in the database correspond to physical or conceptual objects which exist in the laboratory. However, the object classes are not strictly hierarchically structured, they are represented in a structure which is isomorphic to the relationships between real world objects. Hierarchical structures are used only if a corresponding hierarchical relationship existed in the "real world".

Management and control of an IC-CIM manufacturing operation requires database facilities for the conventional business data (e.g. accounting and inventory) and also engineering and scientific data (e.g. geometric positional data and sensor measurement data collected during processing [Rowe, Williams '87]. Provision for the engineering and scientific data types was a difficult design issue to address. Most database systems do not support the engineering and scientific data types, a few commercial RDBMS's are beginning to emerge, only now, that do provide the facility for defining your own data types. As an interim measure, BRDS utilizes the facilities offered by a conventional database (the only available option at the time) to simulate geometric positional data types. This solution allowed some applications to be prototyped and developed to a certain extent, and provided the best interim solution possible. An example is the Facility Layout Information Program (FLIP). The portion of the CIM database developed for FLIP required several tables to store positional data that could be more optimally represented if geometrical positional data types were available.

Performance improvements and a reduction in complexity of queries could be obtained with a database supporting the needed data types. However, the interim solution devised for FLIP was optimal at the time, and has held up rather well. The FLIP database tables were reused without change (except to update data) as the source of data for the laboratory floor plan and equipment and utilities maps in CIMTOOL.

The FLIP tables contain the dimension data for objects. An object is either a single unit, such as a door, or a component to a single unit such as the back panel on a piece of equipment. Another separate table holds data representing vertices where objects meet. Since different heuristics apply to how a window, for example, and a piece of equipment may be displayed in a map, separate tables are used for these different classes of objects. This allows specific methods to be associated with specific geometric positional data types. The RDBMS data type issue may have to be addressed in the design of new subsystems or new IC-CIM system projects. Some examples of data types that will need to be represented in the future are: imprecise or fuzzy data, data associated implicitly by class with a unit of measure, message data associated implicitly by class with protocol methods. Ingres (and more recently some other RDBMS's) now supports user defined data types and a rule management system, both of which may be very useful in implementing the new data types.

FIGURE 21 depicts the dependencies or entity-relationships between the primary database objects at the most general level. Each of the primary database objects corresponds to a table in the database (and a set of database applications). Some of the individual programs open direct connections to the database and others use an external UNIX plain file as an intermediate storage area for data that is periodically transferred to or from the database. There are several different cases dependent on the type of data and how it is used that determine what type of interaction with the database a particular program or operator has:

- direct connection to the database every time the program requires data
- no direct connection to the database, the application produces data and consumes

data stored in ASCII text files and that data is uploaded and downloaded by ancillary programs or scripts

- the database is connected to directly under some circumstances and indirectly through plain files in others

Direct connection is usually the case when data must be updated in real time. No direct connection is typically the case when data need not be updated in real time, and/or data is created in constant streams and at such a rate that to keep a direct connection open would unnecessarily add to system load. In the last case, cases one or two are applied accordingly when a subsystem can conditionally alter program flow.

4.2.3 BCIMS Equipment Control subsystems

This section briefly describes the design and implementation of the equipment control facilities of BCIMS. Entity relationship diagrams for the database objects described in this section (and others) may be found in Appendix 1.

Equipment control

The design goals for equipment control applications of BCIMS have been to provide modular, easily extensible subsystems to support equipment control and communication. The approach taken was to avoid defining any specific architecture for equipment interface modules or subsystems. However, some architectural concepts that have been applied in other components of BCIMS have also been applied to equipment control subsystems. For example, external configuration files are used to specify operational parameters, and modular code libraries have been developed. In all other aspects architecture has been allowed to vary with the specific equipment control application. Because equipment interface is essentially an equipment-specific undertaking, this approach has proven to be a good one. The software/hardware system architecture of equipment control is depicted in FIGURE 22.

There are two primary equipment control subsystems of BCIMS. One is used to send a sig-

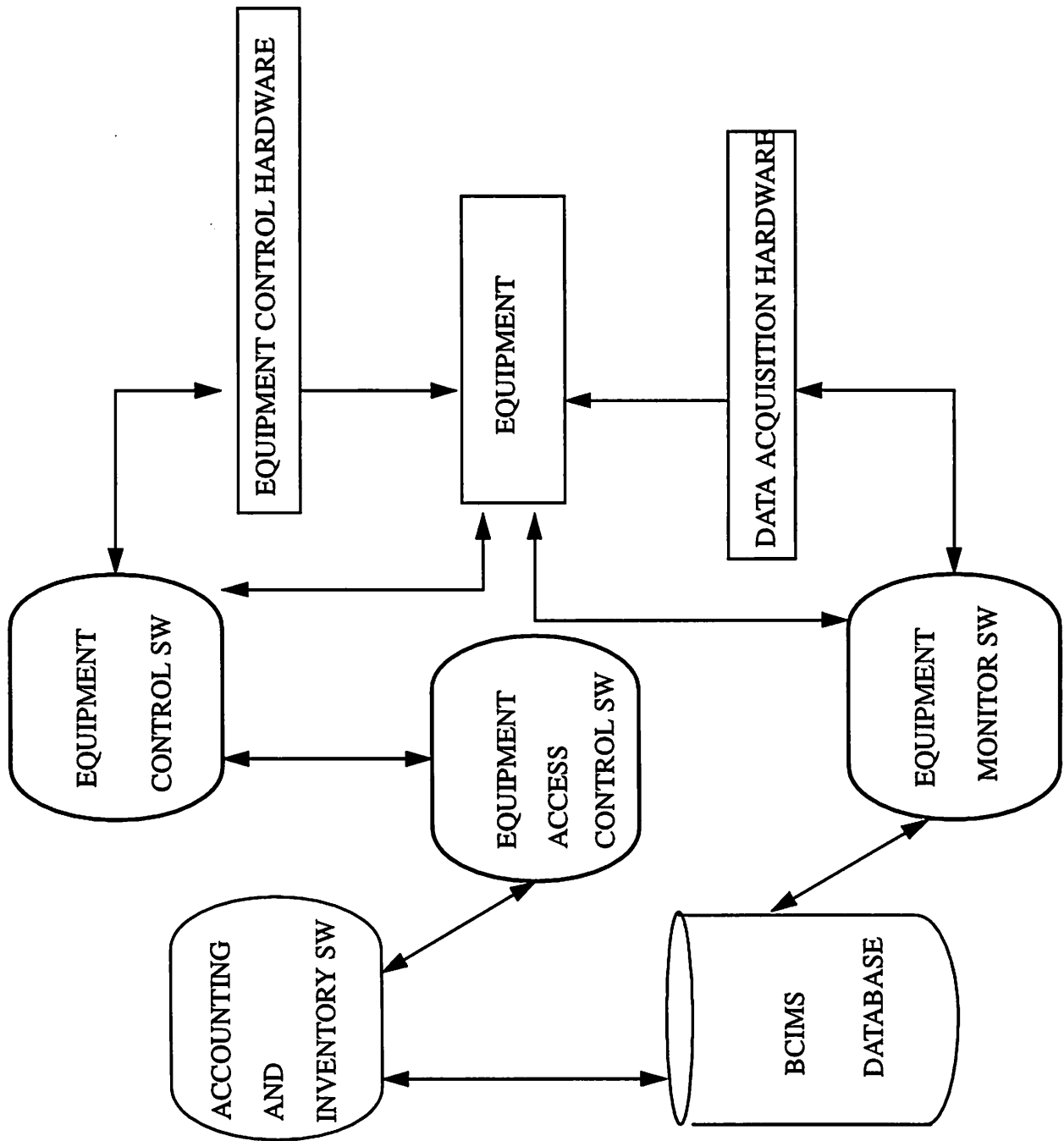


FIGURE 22

Software and hardware data acquisition systems

nal to equipment specific hardware interlocks that can be used to enable or disable equipment use (**eqcntl**). The other is a system for communication with laboratory equipment which supports the Semiconductor Equipment Control Standard (SECS) protocol (**tytalk**, **lamtalk**, **flattalk**, **nanotalk**) [SEMI '92] [Lam '88]. Although the SECS protocol software can be used to control equipment, it is almost solely used in the laboratory to monitor process related data, and therefore is discussed in the following section on Process Monitoring.

The software portion of the eqcntl subsystem consists of the eqcntl program, the **equilib** library and a UNIX plain file used to specify configuration. Qualifications data, labuser data and accounting data are created and/or consumed by eqcntl. The modular code library, **equilib**, consists primarily of functions that drive the I/O hardware through which signals are sent to the hardware interlocks that are attached to processing and analytical equipment in the lab. The I/O hardware is a commercial product called the Taurus Lab. It is a Z80 based controller unit and one or more I/O port boards (the laboratory has two), each of which provides 64 one bit ports. The hardware interlocks, and everything else on the outbound side of the I/O board have been developed by equipment technicians and development engineers who also maintain the processing equipment. The eqcntl software, in itself has no positive information or feedback about the equipment or hardware interlocks. However, it was implemented under the assumption that setting a bit (raising the voltage) will "enable" the equipment for use and clearing the bit (lowering the voltage) will "disable" the equipment. Thus the laboratory managers have complete control over precisely what they wish to occur when a signal is sent by BCIMS, and the system is completely portable and environment/equipment independent. In addition, BCIMS is also independent from its own hardware components. Any I/O hardware similar to the Taurus may be used in place of the Taurus Lab; its capabilities are common and available from a wide variety of manufacturers who supply I/O, data acquisition or process control hardware. Eqcntl participates in the accounting process by logging each "on" or "off" command (equipment enabling/disabling). The log is time-stamped and is used to track use of equipment so that equipment use time may be billed. In addition, eqcntl plays a part in equipment problem reporting. At the end of each session of

use, when the user executes an off command to disable the equipment, the operator is prompted to report any problems with equipment. In this way technicians are assured to receive prompt problems reports.

As previously described, when a lab member wishes to use equipment he or she selects a Wand menu option to “enable” the equipment. The menu option corresponds to the eqcntl program being executed with two argument strings, the first is “on” and the second is the name of the equipment (as given in the BCIMS database). Eqcntl connects to the BCIMS database and performs queries on the labusers and qualify tables in the database. The query is keyed on the User ID (UID) and the name of the equipment. The labuser’s request to enable the equipment is granted if:

- the computer account name corresponds to an active labuser
- the equipment name requested exists in the database of equipment
- the labuser is currently qualified to use the equipment
- the status of the equipment allows enabling (i.e. its not “locked”)
- no scheduling conflicts exist
- the limit on number of pieces of equipment enabled will not be exceeded
- the equipment is not already enabled

In addition, eqcntl’s commands to the hardware must be completed successfully. The labuser is warned if the hardware communication was not successful.

4.2.4 Process Monitoring

Software for process monitoring will be described in this section. A group of subsystems based on a code library that implements the Semiconductor Equipment Communication Standard (SECS) protocol is described. Additionally, a Work In Progress (WIP) subsystem and the Berkeley Process Flow Language (BPFL) is discussed. Entity relationship diagrams

for the database objects described in this section may be found in Appendix 1.

SECS protocol equipment communication

Primary design issues to be addressed for the SECS protocol equipment interfaces were that the protocol be implemented according to the standard, and that the design support modularity and easy extension. The equipment is a limited resource to which access must be managed. Therefore the client-server model was deemed appropriate.

A code library has been developed which implements the generic SECS protocol [SEMI '92] in the C programming language. Equipment specific libraries (one per equipment) implement the equipment dependent portions of the protocol implementation. The SECS software that has been developed includes:

- **tytalk/tytalkd** for the Tylan furnaces
- **lamtalk/lamtalkd** for a LAM Research etcher
- **nanotalk/nanotalkd** for Nanometric's spectrophotometer

In addition to the use of short concise library functions, modularity has been provided for by "protoizing" the generic and equipment dependent portions of the SECS protocol implementation. That is, a template implementation was developed, and that implementation may be "cloned" each time a new equipment specific interface is required. The system is easily extensible because relatively few modifications need be made to the clone in order to tailor it to the specific equipment. To tailor a new clone, the values of environmental parameters are specified in the prototype include files (header files). Additionally, an instance of a record-like data structure, used to specify exactly what SECS messages the equipment supports, must be edited. An instance of a data structure must also be edited to specify the mapping between an input user command to send a supported message, and the function that will be the handler for the request (i.e. the function that will be called to implement the message). And finally, any special equipment specific handler functions must be supplied.

The server daemon process that may be compiled from the prototype library supports two modes of operation. One is a batch mode. In batch mode a user runs the client process executable with arguments that specify a message to be sent. The other mode is interactive. In this mode the client process makes a connection to the server daemon such that all user input is passed directly to the equipment controller. Interactive access to the equipment controller is restricted to a few qualified users because in this mode all output to the furnace is unchecked. Select functions of the SECS protocol equipment interfaces are available from the Wand and Staff subsystem menus. An enhanced equipment interface is available for process monitoring of the Tylan furnaces, called Tycom Communication. In addition to the SECS equipment communication server daemon and client process, a C program (**tytasks**) integrates scripts (**tylan_monitor**, **tylan_monitor_online**) developed to continuously monitor a furnace, store the data, and optionally display the data concurrently. FIGURE 23 shows the main menu for the Tycom Communication subsystem.

```

+argon  TYCOM COMMUNICATION: leslie
CATEGORIES(space).....TASKS(space)
. d Display Status (1-16,18,20)      . c Collect data (during run)
. r Recipes (1-16)                  . l Look at or copy old data
. S Restart tycom interface          . v Collect data and view (on xterm)
. t Tylan watch (tywatch)            .
.==>T Tylan monitor (tytasks)        .
.                                     .
.                                     .
.....
HELP                                IN PROCESS(%)
- Enter character next to item
- SPACE moves between main menus
- Use ESC to suspend
- Use Control-C to exit
- Use ? (and !) for help info- Use ^

```

FIGURE 23 Tycom Communication Subsystem

WIP

Work in progress (WIP) is used to monitor entire processing runs and groups of processing runs and encompasses information about any and all steps and the associated equipment and resources. WIP utilizes the Berkeley Process Flow Language (BPFL) to represent the information. The design and architecture of the WIP system is not described in this paper. It is described in detail in a paper entitled "The Berkeley Process-Flow Language WIP System" [Hegarty, Rowe, Williams '90]. The WIP system has been experimented with in the laboratory and has been a subsystem available on the Wand menu in the laboratory. A baseline process is being developed which will become a standard test-bed component for WIP. WIP will run the baseline process in connection with the Berkeley Computer Aided Manufacturing (BCAM) group's workcell controller, currently under development [BCAM User's Manual '94].

Sensor Data Acquisition

The Berkeley Laboratory Infrastructure Monitoring Program (BLIMP) is a hardware software system that supplies sensor data on utilities to the labuser. FIGURE 24 displays BLIMP sensor status output for a number of different sensors. The primary design issues addressed during development of the original BLIMP involved the reliable delivery of sensor data at a range of sampling frequencies. As with the other BCIMS equipment interface subsystems, the client-server model was chosen. However in this case it is the stream of data (rather than connect time to a server daemon) that is a limited resource that must be managed. Client processes wishing access to data do not directly or indirectly address the data acquisition hardware. To reliably supply data at desired sampling rates, given the resources available for equipment purchase, the data acquisition system must utilize the full I/O bandwidth. Thus the system was designed to eliminate server daemon overhead associated with client process connect time. The server daemon process operates asynchronously and independently from any data consuming processes. An added benefit is that the reduced

complexity of decoupled operation makes the design more robust.

+argon						
Utility	##	Description	Date	Time	Value	Units
n2	1	(ln tank level - 30 inche	8/4/95	8:47	73.53	inch. N2
n2	2*	(n2 flow meter)	8/4/95	13:15	499.90	slpm
n2	3	(house n2 pressure - 75#	8/3/95	12:58	84.10	PSI
o2	4	(o2 flow meter)	8/3/95	12:16	2.33	slpm
o2	5	(house o2 pressure - 25#	8/3/95	12:51	48.20	PSI
rodi	6*	(di tank level - 30 inche	8/4/95	12:46	89.60	inch. h2o
rodi	7	(rodi resistivity - 18 Mo	8/3/95	8:19	18.16	Mohms
rodi	8*	(make-up flow rate)	8/4/95	12:32	-0.05	gpm di
utility	23	(compressed air press. 8	8/4/95	13:04	89.70	PSI.
utility	24	(vacdrain vacuum - 15 inc	8/3/95	15:09	17.84	inch. Hg
utility	30*	(delta barometric pressur	8/4/95	13:10	3.44	Torr
utility	32	(chws temp. 45-55 deg.)	8/4/95	9:54	42.88	Deg. F
utility	33	(supply air temp. 64-68	8/3/95	20:50	55.40	Deg. F
utility	35	(icw pressure - 50 psi mi	8/4/95	13:13	82.50	PSI.
spin2	40*	(relative humidity)	8/4/95	13:15	85.55	%RH
spin2	41*	(air temp)	8/3/95	11:55	25.70	deg C
svgcoat	42*	(relative humidity)	8/4/95	9:56	42.17	%RH
svgcoat	43*	(air temp)	8/4/95	11:43	22.70	deg C
svgcoat	44*!	(diff air press)	7/30/95	12:15	-0.13	in h2o
gases	45*	(h2/n2 cylinder pressure	7/30/95	15:14	2039.57	psi
gcaws	46*!	(gcaws envn chamber temp)	6/26/95	8:26	-44.71	deg C
--More--						

FIGURE 24 BLIMP sensor status output

The BLIMP subsystem consists of a server daemon, one or more client processes, a script that downloads a program to the data acquisition hardware, and ASCII text files (used to store data). The server daemon runs on the machine to which the data acquisition hardware is attached by an RS232 line. Typically, (although the configuration may be changed), whenever the host machine is rebooted a script is run to program the data acquisition hardware. First a signal to halt acquisition is sent to the data acquisition hardware, then a fresh copy of the program, (which incorporates current information from a configuration file) is downloaded followed by a command to begin data collection. When the programming script has completed, a program is executed that continuously collects data. The data is collected from a UNIX domain **socket** bound to character-special device file. This device file is a link to the **tty** hardware port that the remote hardware is connected to (by an **RS232** line). The server daemon process, called **blimpd**, is also started whenever the host machine reboots.

Blimpd runs continuously listening at a socket, accepting connections from client processes, and providing them with data. A recent major revision of the BLIMP software, (BLIMPIII) has been designed by the author and an undergraduate student (Eric Ng), and written in C++. This latest revision was driven by the growing number of sensors being monitored, and the increased client demand for sensor data by researchers. BLIMP has been one of the most heavily voluntarily used (as opposed subsystems researchers and staff are required to use in order to work in the lab) subsystems in the Microlab. The BLIMPIII revision enhances the methods that have been encoded for handling noisy data gracefully. It also adds facilities for broadcasting data to a large number of clients so that sample frequency to a single client is not affected by the number of clients that wish to receive data. Rather than the usual method of queuing client's for data, clients "register" for service and are warned if the requested sample frequency cannot be provided. Thereafter the client may receive broadcasted data. Receipt of data does not require connect time to the server daemon, so an otherwise limited resource (the connection to the server) is obviated.

Several different client programs have been developed and experimented with. The most often used client processes have been those used by laboratory staff. The laboratory staff normally only require updates once per hour, because the utilities data does not usually change very often. In contrast, lab member research project clients of BLIMP typically require high sample rates. The system is designed to support any client program(s). There have been requests to maximize the sample rate for lab member research projects. The maximum sample rate that could be obtained primarily varies with the number of sensors that must be sampled (given the existing data acquisition hardware), although there are several data acquisition system programming factors that may introduce overhead in a manner which is not linear in the number of sensors. When the minimum sensors required by laboratory staff (19) are supported the best sample rate that could be achieved was approximately 3 samples per second. The limiting factor in this sample rate was found to be the data acquisition hardware, not the BLIMP software. Without faster data acquisition hardware, we could

not empirically determine the best sample rate that could be achieved by the software given high throughput hardware, however experiments were done using simulated data input to BLIMP III at much higher sample rates. Based on the results of the simulation, we estimate sample rates of one tenth of a second could be reliably achieved on the current platform, and that higher rates are possible.

4.2.5 Faults Equipment Maintenance and Diagnosis Tool

In this section, the design and implementation of the Faults subsystem is briefly described. For an in-depth description of Faults see the paper by its author [Mudie '91], which is summarized here.

Equipment maintenance and diagnosis

Faults is an equipment maintenance and repair tracking system which also provides a mechanism for failure analysis and diagnosis. The facility for failure analysis and diagnosis may be utilized directly by the operator or by a diagnostic software application. There were several issues to be addressed in the design of Faults. Typically, in existing semiconductor manufacturing environments, information describing equipment status or qualification and maintenance is stored on paper or in unstructured computer text files. Problem reports are reported in unstructured text, or even orally to technicians by operators. These unstructured means of storing the information make it extremely difficult or impossible to perform any useful system analysis for diagnosis.

To address these issues, the following goals for Faults functionality were developed:

- provide structured equipment qualification and maintenance tracking tools for the facility management
- support structured maintenance case history recording and browsing for technicians
- design the user interface and supply operators so that the users of the system may

also maintain it without the need for a programmer

- provide a mechanism for integration with a variety of other CIM applications that require data on equipment status and history, such as automated diagnosis system

Based on these design goals, issues to be addressed include:

- developing appropriate data structures
- formalizing the semantics of equipment maintenance and repair information
- utilization of the forms-based interface for naive users
- development of a consistent application interface for subsystem integration within the CIM framework

An object-oriented paradigm is used as the data model in the development of the Faults database tables. Five classes of object are defined: **labusers**, **resources**, **comments**, **reports** and **faultsymps**. All but the faultsypms class have straightforward physical counterparts in the manufacturing environment. The faultsypms type is the parent class to the fault and symptom classes. To enforce structure and to formalize the semantics of Faults information, a hierarchical structure composed of faults and symptoms is used. Through standard knowledge acquisition techniques, information describing typical faults and symptoms was gathered, and an unambiguous terminology for faults and symptom description was developed.

The forms-based user interface uses persistently displayed menu items and restricted operations on clearly labeled fields. The interface is designed to minimize human error and provides a structure to the actions of entering new data, browsing historical data, and modifying hierarchies. This makes it possible for labusers to maintain the system without expert help.

Lastly, the mechanism of a uniform integration point for other CIM subsystems is implemented in the same way that the other major BCIMS subsystems provide this support. A modular code library, faultlib, has been developed. The library contains code modules which

may be linked in to any applications which require integration with Faults data and data structures. An application utilizing faultlib use BRDS as the operational integration mechanism.

4.2.6 BCIMS Facility management subsystems

This section describes the design and implementation of several BCIMS subsystem devoted to facility management. FLIP and CIMtool are only briefly described. For in-depth description of FLIP and CIMtool see [West '89], [Rowe, Williams '87], [Smith, Rowe '91], and [Picasso '90].

Facility management

Facility Layout Information Program (FLIP) and CIMtool are two distinct but related subsystems devoted to facility management. FLIP is one of the subsystems that was developed early, during the mid to late 1980's, and is a precursor to CIMtool which was developed in the early 1990's.

Design issues addressed in the development of both subsystems included:

- database information representation
- database application design
- user interfacing to a large system of complex and volatile data

The information representation design has survived unchanged since it was developed as part of FLIP. The FLIP database tables have been incorporated into CIMtool; they are used to create the CIMtool graphical interface to the laboratory.

The FLIP database is object oriented in that database objects are isomorphic to corresponding physical objects in the laboratory. For example, FLIP database objects include the lab itself, walls, vertices (of objects), equipment, material, utilities, etc. A primary challenge to

the development of the FLIP database was the representation of geometric positional data so that the objects could be displayed. At the time when FLIP was developed, database systems did not support either geometrical or scientific data types, or user-defined data types (Ingres now supports user-defined data types). Positional attributes for the objects had to be explicitly defined. The approach taken was to use a two-dimensional coordinate system common to all objects, and to represent “height” and “bottom” of an object only when necessary. The height attribute is a relative measurement of the distance between the bottom of an imaginary bounding box enclosing an object, and “bottom” is the vertical elevation relative to the zero height attribute of the lab object to the bottom of the bounding box enclosing an object. Thus the representation is a mixture of absolute and relative coordinate systems within a single framework. This approach minimized the data required to represent laboratory objects.

A great deal of energy devoted to the design and implementation of FLIP went into the development of the menu-driven and graphical user interface. X Windows is utilized, however the user interface issues that were faced went far beyond simply using the resources provided by X at the time (mid to late 1980s). One of the design issues addressed was creation of an interface and program flow that would allow a naive user to modify database data. To this end FLIP was designed to consist of separate modules. Each devoted to handling one of the interface areas according to functionality. Modules developed included:

- view generator
- data entry editor
- specialized built in graphics and text handling capabilities
- an events and scheduling manager

FLIP was designed so that all of these special built-in capabilities could be replaced by toolkits in the future, and that is exactly what successfully occurred in the development of CIMtool. CIMtool utilizes the capabilities of FLIP, and goes further by integrating the database

tables and some of the facilities of other subsystems such as Faults.

The CIMtool application design is based on the primary goals of:

- providing a tool that is easy for naive users to learn an use, and
- integration of a wide variety of dissimilar CIM data into a representation that offers a high level view of the status of the laboratory.

Rather than design built-in facilities to support these features, CIMtool is based on the GUI toolkit, Picasso [Picasso '90] and an Ad Hoc Query Interface to the database [Smith, Rowe '91]. Using Picasso allowed developers to quickly prototype CIMtool. Multimedia and hyper-text extensions to CIMtool have also been developed (also based on Picasso). Additionally, the Faults and BLIMP III BCIMS subsystems are integrated under CIMtool. CIMtool is a specifically applied example of the Ad Hoc Query Interface (referred to as AQI in the remainder of this paper). The AQI allows end-users to specify and run ad hoc queries without requiring any special knowledge of query languages or database design. The user is presented with panels of switches that can used to incrementally build a query.

The data structures developed for CIMtool are Common Lisp Object System (CLOS) objects (Picasso is written in Lisp also). CIMtool is object oriented in a fuller sense than FLIP in that a true object hierarchy is used in the graphical representation of objects. For example, an etcher object class is a subset of an equipment object class, and a specific type of etcher would be a subset of the etcher object class. This hierarchy is most general at the top and becomes increasingly specific, mirroring the procedure used to develop queries using AQI.

4.2.7 Administrative applications

In this section, the design and implementation of the administrative subsystems of BCIMS are described. Entity relationship diagrams for the database objects described in this section may be found in Appendix 1.

The design goals for BCIMS administrative applications were primarily to produce applications that are secure, robust, capable of reversing the results of transactions that are already committed to the database, and keeping secure records of transactions. In addition, special attention had to be paid to minimizing data entry errors.

Administration

Primary administrative applications within BCIMS are **acct**, **inven**, **vd**, **labusers**, **purchase**, and **contracts**. All administrative applications store data in the database, some of them directly access the database every time the program is run and others do not. Administrative data is characterized as being generated frequently from many different sources simultaneously, and it is critical that data points are never lost. The generalized procedure used in BCIMS for processing of data with these characteristics is:

- put data from sources and/or database table(s) into a UNIX plain file
- review/edit data using an editor specialized for the task
- conclude with a file/hardcopy report and/or updating/appending to database table(s)

All administrative data is constantly collected from sources and is uploaded into the database twice per day. After the database upload new plain files are downloaded from the database. Administrative staff may upload or download data on demand at any additional time.

The **acct** program is used to calculate charges for laboratory use and generate accounting statements for a specified period. **Act-upload** is used to upload and download accounting activity data. The **acct** application consists of a program executable, several tables in the database, and also UNIX plain files containing data. The procedure for producing the routine

periodic statements is:

- upload data into the database from the plain file it is constantly collected in
- generate a statement for the period (usually a month long interval) into a *plain UNIX* file
- review and edit the statement in the plain UNIX file using a dedicated accounting editor
- finalize the statement, producing a new finalized plain UNIX file and hardcopy

The plain UNIX files that contain the statements and finalized reports are always written to files named for the accounting period of the report and an extender indicating type of statement. The path to the files is hard-coded for security. The tables that store data for acct (also accessed by other programs) are **activity** and **acctperiod**. As the names suggest they store the charges activity and the record of statement periods. When statements are generated or finalized the acctperiod table is updated to reflect the current state for that period's accounting and a time-stamp. The specialized editor (**vd**) is designed to reduce the probability of data entry errors by restricting operations. **Vd** also records who is editing and when, uses locking on all files being opened for updates, and always creates a backup copy of a file opened for editing. In addition, an option is provided for the labuser to voluntarily store notes that will also be time-stamped.

The **inven** program also utilizes the **vd** editor. The same general procedure is followed in the case of inventory record maintenance as described above for accounting reports. Separate fields display current inventory levels, inventory location and reorder thresholds. The **vd** editor is used to provide control over what can be done within each field type. For example, fields may be restricted to a specific number of digits, or be restricted from editing.

The **labusers**, **purchase**, and **contracts** subsystems are all forms based database applications. In addition to providing a consistent user friendly interface, Ingres forms definitions are used as a mechanism to provide very specific control over the editing of fields. For example,

for security reasons certain fields in the labuser forms may only be updated by someone logged in as a specific labuser at a specific terminal during specific business hours. Additionally, attribute definition and type checking are used to very narrowly define acceptable input for fields, thereby decreasing the possibility of errors in data entry.

4.3 BCIMS Hardware Architecture

In this section, the current hardware architecture is described. For a historic description see Section 5.1.

Hardware architecture

The current hardware architecture is a distributed computing system composed of two compute and file servers, one for production called Argon and one for development called Radon, and client workstations [Sun '90]. X terminals and alphanumeric terminals are connected to the servers over a local area network. Typically, 50 people are logged into Argon and 20 people are logged into Radon during the day. The development machine is also a standby system that can replace the production machine if needed.

In addition to the servers, workstations and terminals, we have several specialized hardware subsystems that are used for data acquisition, equipment control, and direct connection to processing or analytical equipment. These specialized subsystems are connected by RS232 links to one of the workstations or terminal servers or directly to the production server. FIGURE 25 shows current hardware components of the Microlab equipment cluster.

The data acquisition system is a commercial product (an Acurex). It monitors approximately 40 sensors, primarily concerned with utilities and resource levels. The equipment control hardware is a commercial product (a Taurus). It implements the capability to signal the hardware interlocks (developed and managed by the laboratory technicians and engineers) that are used to control access to equipment and resources. Direct connection to equipment is

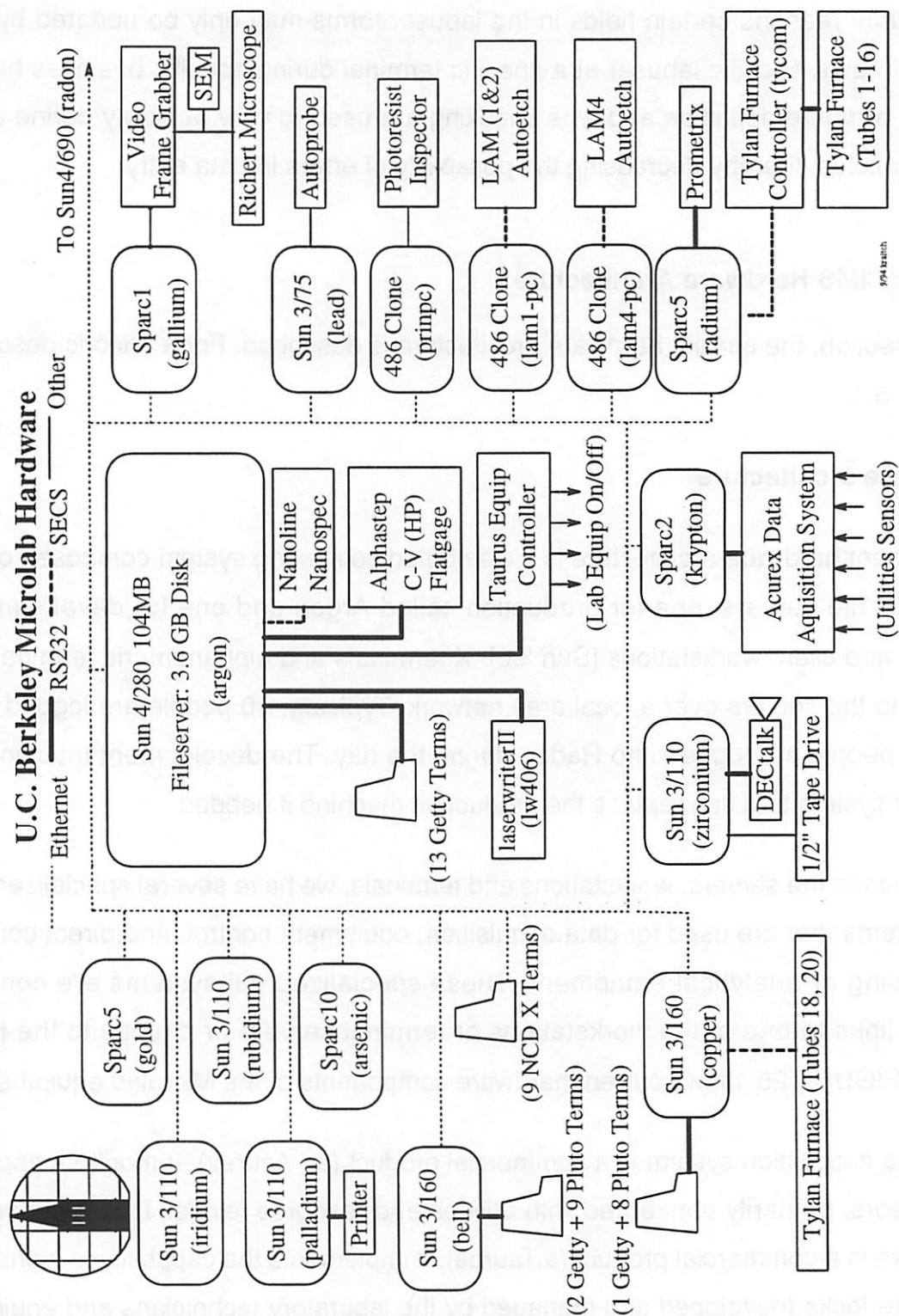


FIGURE 25 The BCIMS multi-cluster network

via RS232 links. Direct equipment connections utilize the SECS protocol whenever possible, and specialized protocols when the equipment does not support SECS.

5.0 History and organizational facts

5.1 History

Semiconductor research has been ongoing at UCB since the advent of the microchip. EECS/ERL established an early integrated circuits fabrication laboratory in 1962 [Microlab '93]. In 1979, planning for a new laboratory to support fabrication of higher density chips began. Construction started in 1981 and the new facility opened in 1983. Computerization of the new laboratory began at its planning stage, and some support for utilities was implemented during its construction.

The facility, now called the Berkeley Microfabrication Laboratory (or Microlab for short), includes class 100 clean rooms and fabrication equipment for semiconductor research. At about the same time, in the early 1980's, professors Hodges and Rowe started research in computer-integrated manufacturing (CIM) and the Microlab became the testing facility for CIM software. The first version of the BCIMS was installed in 1984 to support operations in the Microlab. Since then it has been in continuous use. BCIMS evolved through a cooperative research project of five years, with substantial input from the users and staff of the Microlab [Voros, Ko '89].

The initial computing environment was a centralized timeshare computer with alphanumeric terminals connected to it. In the late 1980's the environment changed to a distributed computing system composed of two compute and file servers: one for production and one for development. Workstations, X terminals, and alphanumeric terminals are connected to the servers over a local area network. In the summer of 1990, these machines were upgraded to Sun4's. The same software and database system that ran on the previous platform, DEC VAX and Sun3, now runs on the Sun4. We were able to move BCIMS to different machine architectures after making relatively few modifications.

The project slowly moved from the research phase into routine use until it became an indis-

pensables tool for Microlab management. Many self contained modules were added along the way and we now have all necessary features to fully support a university microfabrication laboratory. The BCIMS software was released to the Industrial Liaison Program [ILP '94] for distribution in 1992.

With Professor Spanos joining the faculty in 1988, the emphasis in CIM research shifted to equipment communications, monitoring, and control. The Berkeley Computer-Aided manufacturing (BCAM) system is a framework built to facilitate the experimentation with various CAM applications. BCAM currently supports real-time monitoring, real-time SPC, diagnosis, recipe generation, and equipment modeling, as well as run-by-run process control for multi-step sequences such as photolithography [Leang, Thomson, Bombay, Spanos '94]. The efficient integration of these applications has been based on the common data structures for equipment models and for process recipes. BCAM [BCAM User's Manual '94] was released for distribution in 1993.

5.2 BCIMS facts

- Total lines of original code written as part of BCIMS: approximately 200,000
- Lines of code in source directory *~micro/src*: 162,386
- Lines of code in code libraries *~micro/src/lib*: 31,290
- Size of the entire distribution of BCIMS: approximately 200MB (includes a sample database, but not third party software such as the RDBMS)
- There are approximately 160 lab members who are "active" in the lab in any given month.
- Peak number of lab members using BCIMS in the Microlab at a single point in time is approximately 50.
- BCIMS has been validated over the years of its evolution by operating the University of California's Microlab

5.3 Organizational Facts

Microlab Supplies and Expenses (S&E)

- In the fiscal year 1993/94 "Computer maintenance and upgrade" cost the laboratory \$13,812.24, about 3% of "Total Supplies & Expenses" for FY 93/94.
- In the fiscal year 1988/89 "Computer maintenance" cost the laboratory \$4,746.76, less than 2% of the figure given for "Total Supplies & Expenses" for that same fiscal year.
- System administrator's salary is NOT included in either of the above.

Other Related Expenses

- Ingres database license fee (purchased by BCAM research group)
- Other licence fees (purchased by BCAM research group)
- Software packages and utilities used by BCIMS or CIM researchers (beyond those typically bundled with the host computing environment) have included: Frame-Maker, Allegro Common Lisp, CLOS, X Windows, Motif, Ingres RDBMS, Gnu Emacs, GNU C and G++, OCTtools, KIC, MAGIC, SUPREM, VEM, Spice, Fluent Creare X, and Phoenix.
- Professional Staff Programmer/Analyst (one full-time employee)

Personnel

- The total number of graduate students who have participated in the original BCIMS project is eight, working with four professors.
- Several professional staff programmer/analysts and undergraduate assistants have been employed over the years.

6.0 Future research and development Directions

In this section some ideas for future or continuing CIM research projects are briefly listed.

Future directions

The workcell controller developed by the BCAM group should be brought on-line in the laboratory in connection with WIP and BPFL so that the individual systems can be experimented with, and the integration of these separate systems may be effected and experimented with.

An automated diagnosis system would be an extremely valuable tool for semiconductor fabrication. This would be a natural extension to the work begun by Faults, which can provide the data required for such an application.

Sensor data on the status of system resources and equipment control subsystems should be coupled so that researchers may work towards an "intelligent" equipment control mechanism in the lab. The current system provides no connection between these systems other than the human observer.

Current technology (or advanced) user interface techniques must become the standard in use in the Microlab. Hardware upgrades should include: more color displays - preferably touch screens, easy to manipulate pointing devices, voice recognition and more voice synthesis (currently we only have voice synthesis for broadcast of safety alarms over the intercom). We should use the new interface hardware along with video to build a multimedia version of the Wand. This will provide CIM researchers with a better test-bed. Experimental data should be gathered so that the effects on productivity and decreased human error may be examined.

Use of hypermedia techniques should be expanded beyond the instructional system developed for IC instruction (IC-HIP). Users should have hypermedia tools added to BCIMS to replace the currently practice of using unstructured files for storage of lab research notes

and other records.

Now that RDBMS's, including Ingres, have recently implemented support for user-defined data types, we may begin to take advantage of database object types based on *geometrical* and scientific data types in the design of new applications.

Another new capability that is now available in commercial databases, including Ingres, is rule based knowledge management. Perhaps this is one method that should be explored through its application to integrating the sensor feedback with equipment control.

A subsystem which implements a uniform interface to the various configuration points of the BCIMS software should be implemented. The representation of the BCIMS parameters should reflect the correlation of parameters (if any). A first revision may warn a staff member about other related parameters. A second revision might implement sanity checking on configuration changes.

Ultimately, now that the underlying framework is stable we should work toward automating at a higher, facility-wide level. This facility-wide automation should be enhanced in two primary areas. The first is the area of visualization of scientific data and visualization of the entire system state in real-time. The second is in relieving the human components of the system from time consuming busy work. A higher level of intelligence throughout the system should be strived for, and is now feasible in terms of the processing power and hardware which has become available recently.

To go further towards a higher level of system automation, intelligent assistant agent software can be developed which may potentially be of great service to the administrative and technician staff who support the laboratory. In addition, such software agents may play a useful role for researchers and staff as an aid in data collection and analysis.

7.0 Appendix 1: Database Entity Relationship Diagrams

FIGURE 26	CIM Database: Primary top-level entity-relationship diagram	71
FIGURE 27	CIM Database: Accounting tables entity-relationship diagram	72
FIGURE 28	CIM Database: Labuser tables entity-relationship diagram	73
FIGURE 29	CIM Database: Gases tables entity-relationship diagram	74
FIGURE 30	CIM Database: Inventory tables entity-relationship diagram	75
FIGURE 31	CIM Database: Purchase tables entity-relationship diagram	76
FIGURE 32	CIM Database: Qualify tables entity-relationship diagram	77
FIGURE 33	CIM Database: Reserve tables entity-relationship diagram	78
FIGURE 34	CIM Database: Resource tables entity-relationship diagram	79

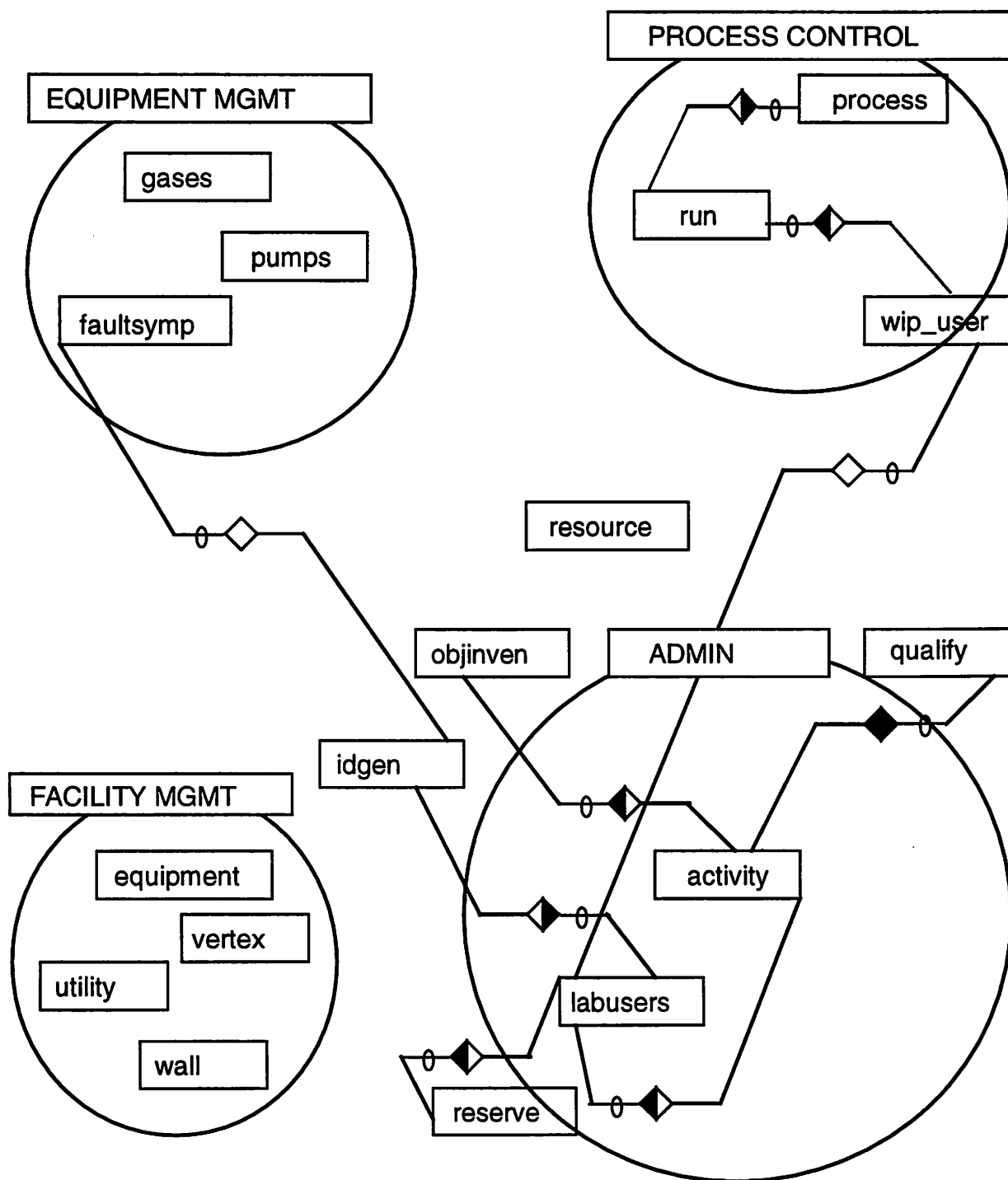


FIGURE 26

CIM Database: Primary top-level entity-relationship diagram

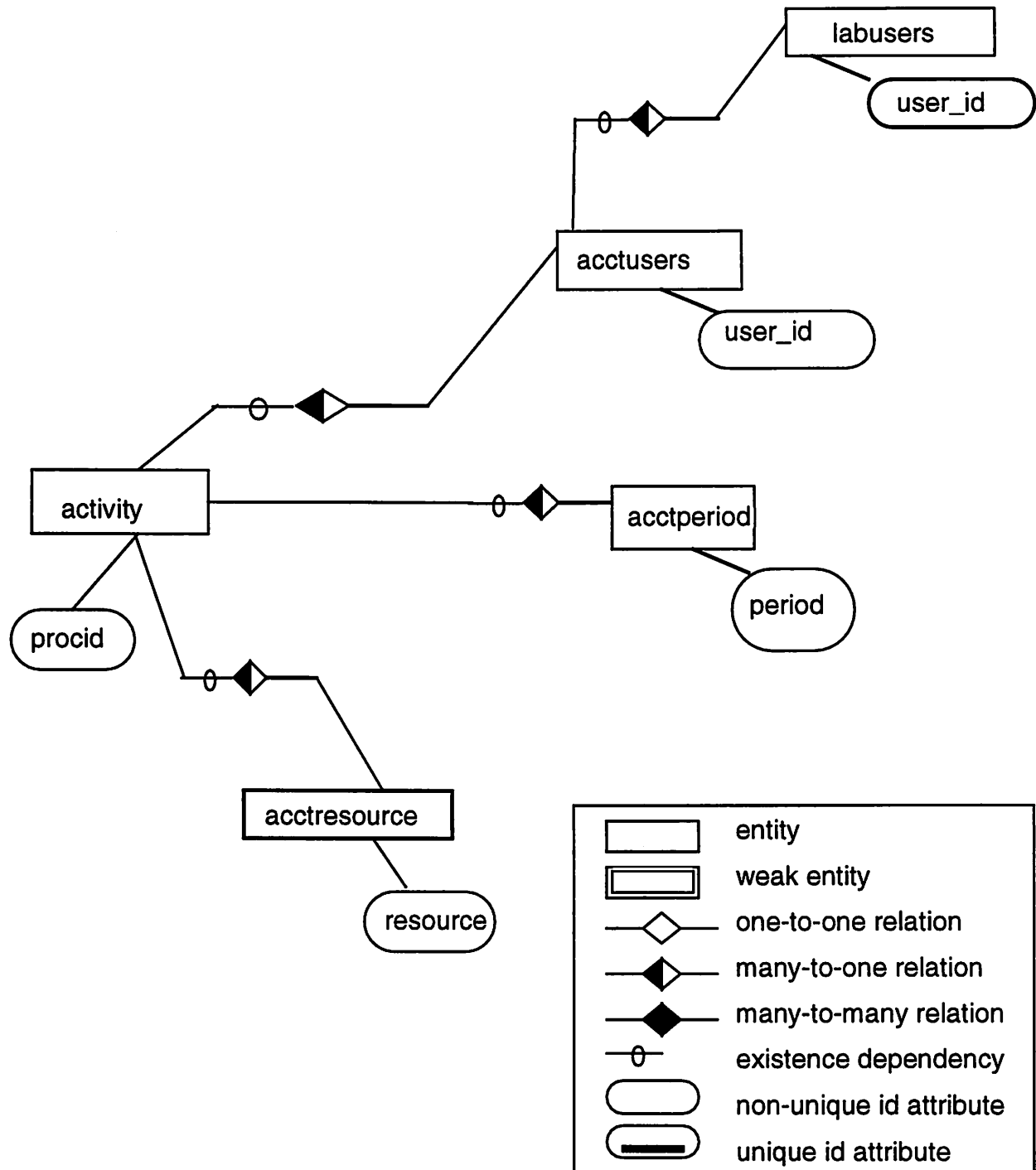


FIGURE 27

CIM Database: Accounting tables entity-relationship diagram

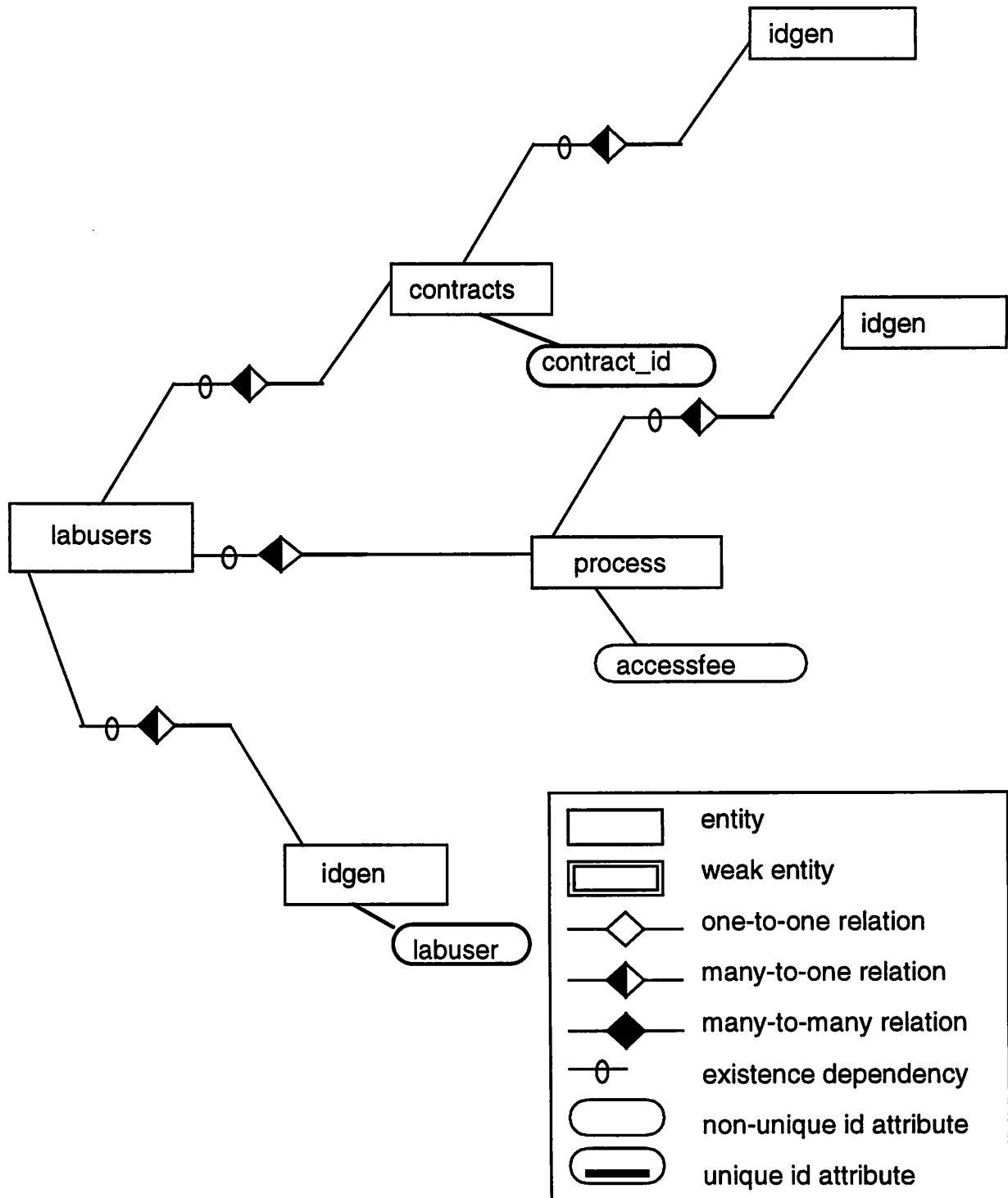


FIGURE 28

CIM Database: Labuser tables entity-relationship diagram

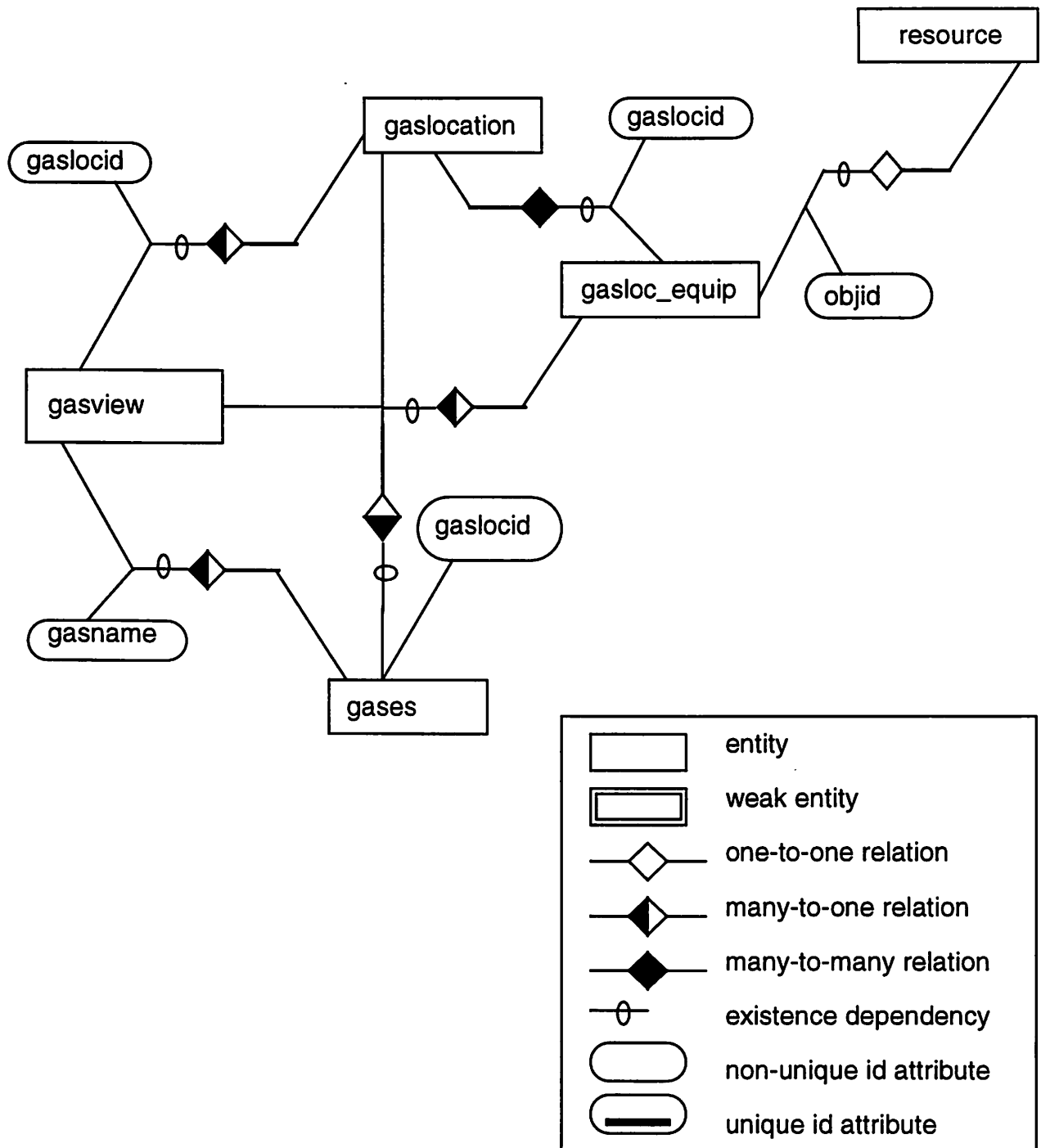


FIGURE 29

CIM Database: Gases tables entity-relationship diagram

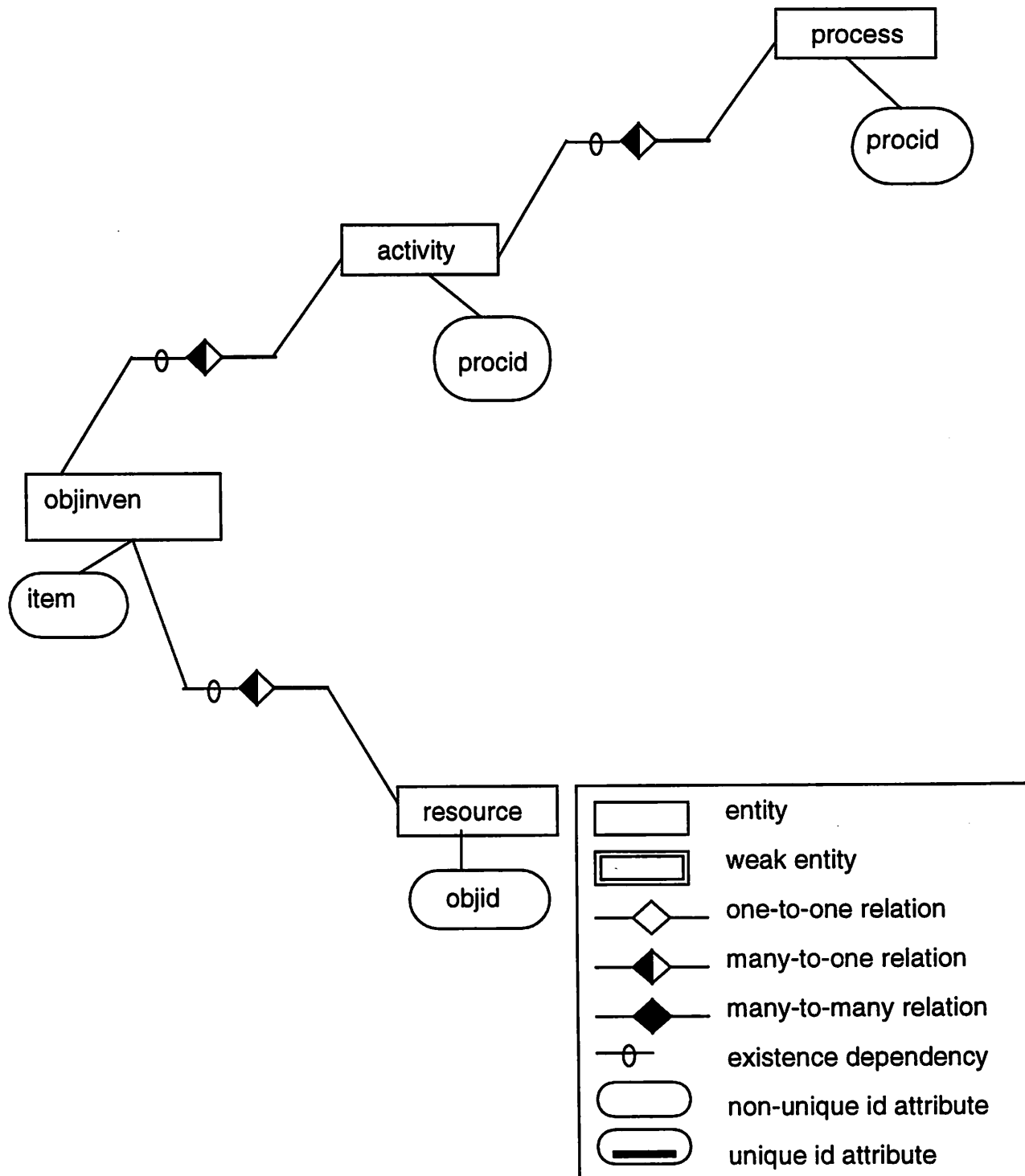


FIGURE 30

CIM Database: Inventory tables entity-relationship diagram

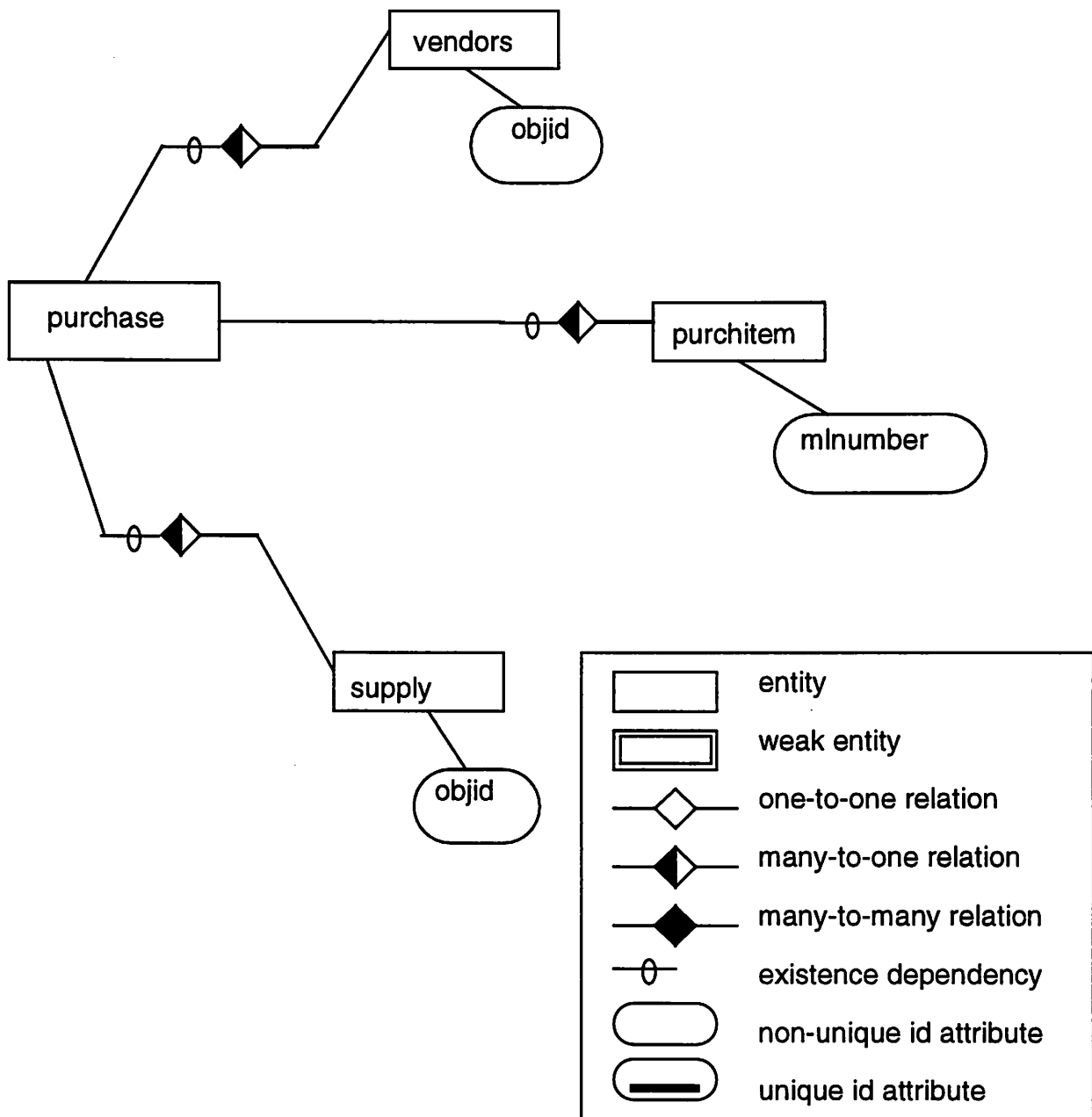


FIGURE 31

CIM Database: Purchase tables entity-relationship diagram

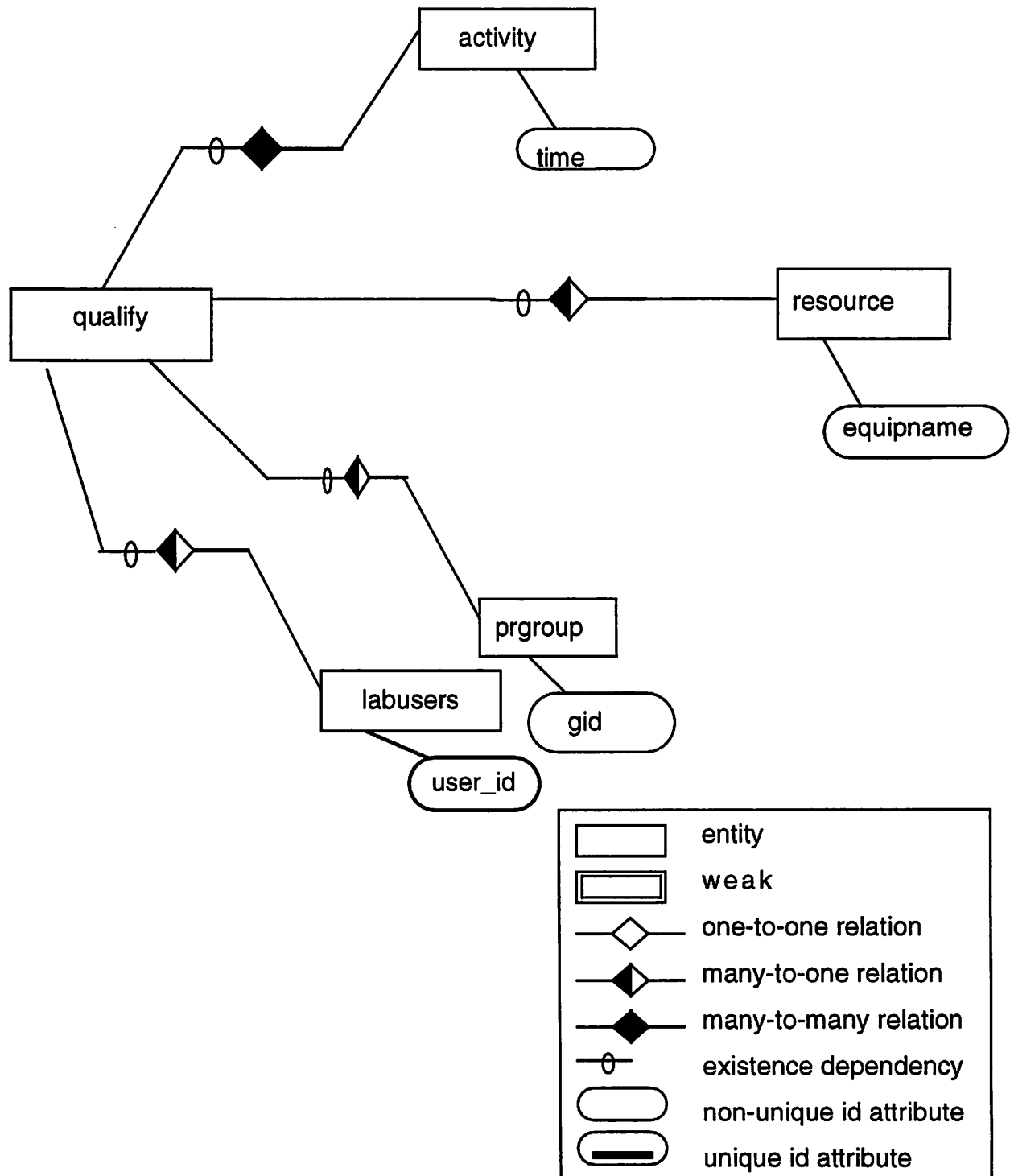


FIGURE 32

CIM Database: Qualify tables entity-relationship diagram

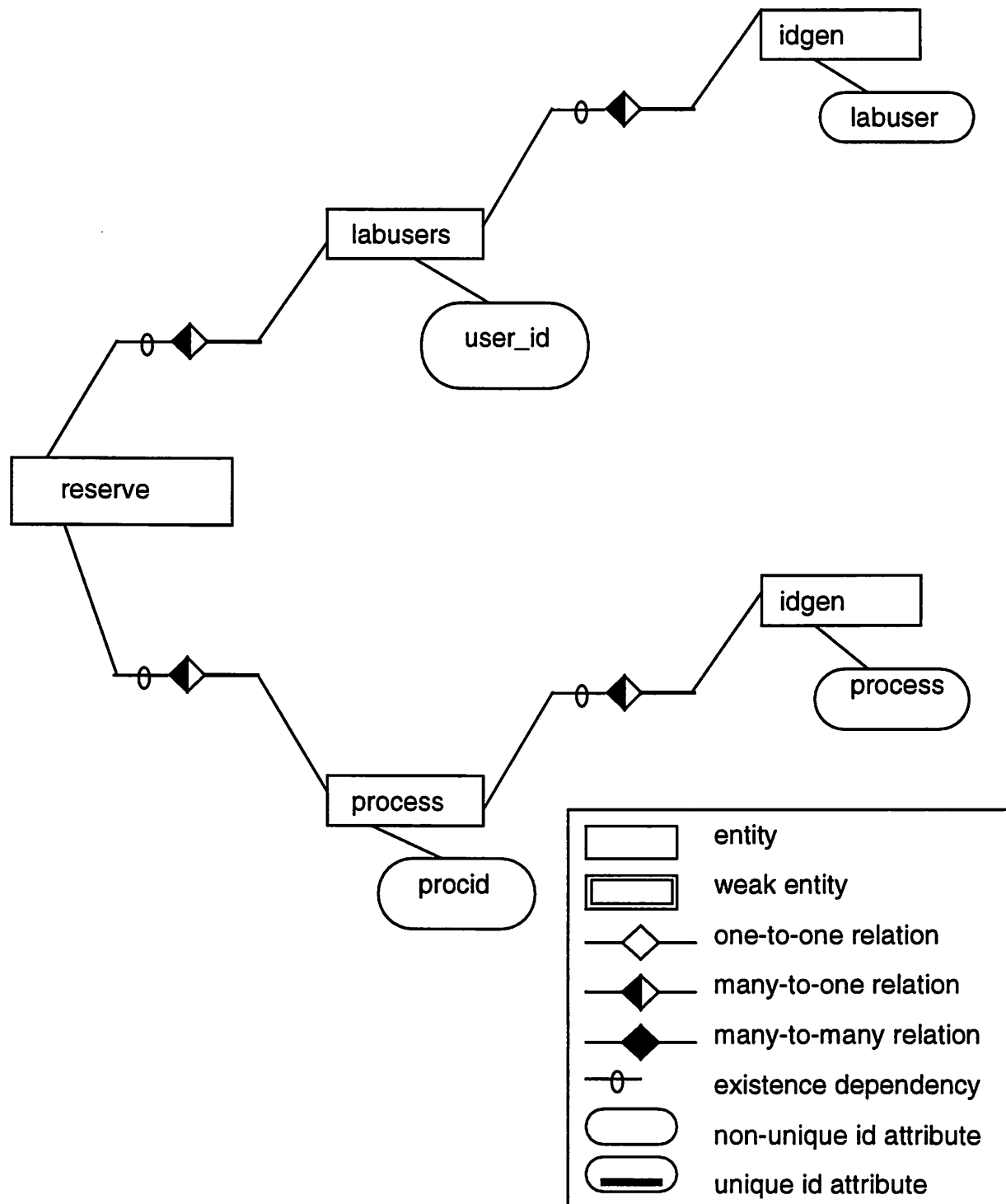


FIGURE 33

CIM Database: Reserve tables entity-relationship diagram

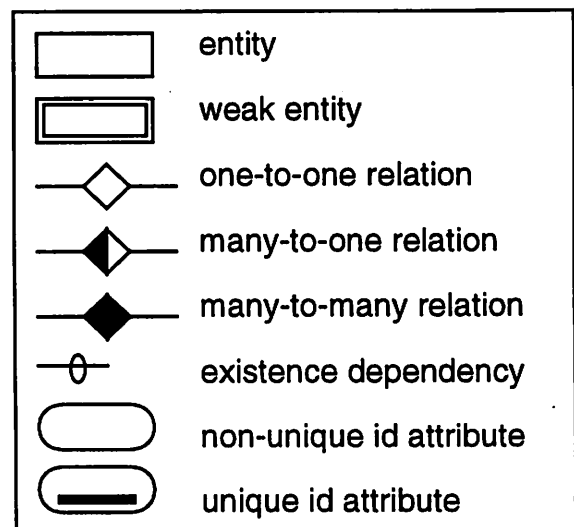
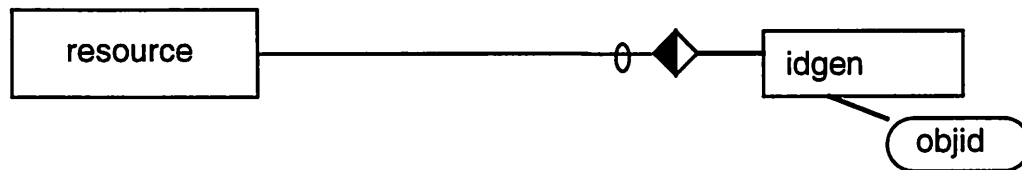


FIGURE 34

CIM Database: Resource tables entity-relationship diagram

8.0 Appendix 2: Technical description of the Wand login system and related processes

This appendix will describe the Wand system and the group of related processes, assuming a basic knowledge of Unix at the system level.

As a precursor to describing the implementation of the Wand, we introduce some terminology. The notion of a single “wand session” means one span of time logged into the Wand, delimited by an initial **login** at the login terminal, and a logout of the Wand session resulting in a **exit** of the login shell at the end. A tty or pty are the Unix character-special devices. The tty device is associated with a single specific hardware unit, typically a single RS232 port on a hardware i/o device. The pty, or “pseudo terminal” driver is a software construct. It is typically not associated with any specific hardware unit, and may be associated with any hardware port. The terms “HUP”, “TERM”, and “KILL” refer to the Unix **signals**, SIGHUP, SIGTERM and SIGKILL as defined in the Unix system C include file, signal.h.

There were difficult technical issues to be addressed in the design of the Wand’s facilities for user login shell handling. As described above the major issue was the design of a mechanism to suspend a login shell and disconnect and reconnect it from a **connection** to a character-special device. The resulting solution and implementation takes full advantage of the Unix facilities for process control (job control). The Unix process control facilities depend on the terminal I/O system to control access to a terminal, or tty. When a person logs into a terminal, a job, or group of processes that are controlled and associated together with a terminal (or tty) and a user (or uid), are assigned a unique identifier called a process group identifier (PGID), a process identifier (PID) number. An instance of a data structure for each terminal contains a PGID as one of its fields. The shell and the terminal that it is running on are assigned to the PGID, which is the same number as the login shell’s PID, when the terminal is first opened [Leffler, McKusick, Karels, Quarterman ‘89].

The suspension of the Wand process was dealt with by developing a strategy by which the existing Unix facility for suspending or “backgrounding” processes (normally not applied to the root or login shell), could be applied to a login shell. The approach taken was to engineer a replacement for the **getty** process which opens and initializes a tty, prompts for and reads a login name, and invokes the login process. This replacement process is called a **pluto**, and a terminal running connected to a pluto process is referred to as a “pluto terminal”.

While the getty process associates one specific tty with one specific login, the pluto associates with one specific tty and any pty(s). The pty becomes the master in the master-slave pty-tty pair. The user’s shell is then associated with the pty. The **ioctl**s TIOCSTOP and TIOCSTART can then be used to suspend and awaken the login shell.

By changing the PGID in the status portion of the tty data structure, the process that is associated with, or actively using a tty can be changed. Normally this would result in termination of the login shell process because every Unix login shell requires a parent associated with a special -character device, or other parent process. To satisfy the Unix requirement a process is forked during the initial login procedure which will function as the parent process to the login shell. The login shell’s PID may now be written into a tty’s data structure as the tty PGID number, thereby associating it with a tty, but it may also be transferred to another tty/pty without termination of the login shell. Most of the facility of the intercommunicating processes mentioned at the beginning section is in performing tasks required to support the ability to move the Wand login from terminal to terminal in this fashion. FIGURE 20 depicts the intercommunicating processes and their relationship to one another.

A server executable called **waved** is also one of the component processes, it is a daemon which can be connected to through a Unix domain **socket**. The daemon is a server process that typically is started at boot time and continues to run, **accepting connections** and providing resource(s) to client processes, as long as the machine is up. If the waved daemon is **killed** then a new_waved process is automatically executed. Waved’s domain of responsibil-

ity is allowing and disallowing lab logins and maintaining login statistics and user identification parameters. The resource waved supplies is “lab login” (and the external files that must be created in support, to allow other processes to start a Wand for a user). The other components to the Wand make up a collection of hierarchically organized processes that perform some initialization functions and thereafter manage the user login.

When a new wand login is started, the first process to be **forked** is an **incant** process. It forks a copy of itself and the child incant gets a pty for the user by trying to open each pty in order until it **open**s successfully. Incant then records the pty name in the user’s Wand file. The pty is locked for exclusive use. Then the child incant forks an **overseer** process and **wait**s on the process until overseer **stop**s itself with SIGSTOP, indicating that it is ready to accept connections. The child incant then **signal**s its parent_incant with a SIGTERM causing it **exit**. Upon the termination of the parent, the waved daemon receives a SIGCHLD signal, and continues on to server the next client request. The child incant (and now only incant process) sends a SIGCONT to restart the overseer process, then waits on the overseer to **term**inate. When the overseer process **exits**, the incant process continues on to fork a **clruser** process which logs the user out from the current Wand login session.

The overseer process is responsible for connecting and disconnecting the users Wand login shell. The overseer is running or stopped for the entire duration of the Wand login session. The overseer exits when the user selects the Wand option to log out, thereby ending the Wand session. When the overseer process is first forked by incant it **bind**s to a socket so it can **listen** at the socket for connect requests, and upon accepting a connection, communicate with a pluto terminal. Overseer records the pluto terminals name into the users Wand file. The overseer then executes **setuid**, **setgid**, and **initgroups** so that it effectively “becomes” the user by inserting itself into the hierarchy of standard Unix processes typically present for any login, in a position directly above the login shell process. Overseer then sends a SIGSTOP to itself, signaling the parent incant process to continue. The parent,

incant, restarts the overseer with SIGCONT. Thereafter the overseer listens for connection requests from pluto terminals.

Before a connection request to an overseer from a pluto can be made the pluto must connect to waved to get the socket identifier for the users overseer process. The pluto then attempts to connect to the socket. When overseer receives a connection request and accepts it, it determines the pluto terminal's host and tty device names from the socket, and awakens the users Wand login shell at the pluto terminal. If this is an initial connection, the overseer first looks up the user's default lab shell (usually a Wand shell) in a configuration file then forks the shell and waits on the shell process for either a stop or a terminate. A stopped lab shell with no exit causes the overseer to disconnect the lab shell from the pluto terminal, then stop and listen for another connection request at the socket. If the user's lab shell exits, the overseer sends an out of band message (OOB message) through the socket. the OOB message will be received by any and all pluto terminals that may be connected. The overseer then exits, and the parent incant process completes the accounting records for the users lab login time. If overseer receives a HUP signal it will stop the lab shell, disconnect from the pluto and resume listening at the socket for connection requests. If the overseer receives a TERM signal it will **hup** and **term** the lab shell before **killing** it (to allow it to exit gracefully) then it will disconnect the lab shell from the pluto and exit. The hup of the overseer can be sent by a remote process when the user has left a lab shell running at one pluto terminal and then tries to "call up the Wand" at another terminal. In this way the lab shell is transferred to the new pluto terminal. The term signal is sent by the cluser function that can be used as a stand-alone program that is run by staff to clear "wedged" or leftover lab shells, or by **waved** when it receives a new initial lab login request for a user already logged in, or after a machine reboot when the Wand may log a user out because a period of downtime has occurred unexpectedly.

The ancillary files used for configuration, data transfer and locking, by the Wand login system are plain Unix files. Each entry in the configuration file specifies one component of the

main menu or a submenu. Fields in the configuration file (Menudefs) consist of

- text of the menu item as it will appear on the menu
- a character indicating the type of executable to be executed
- the Unix command line or built in function to be executed
- arguments to the executables (if any)

The code libraries for the UI contain the routines that interpret the contents of the specification file and map the tokens (strings) to routines that implement operations. The implementation of routines for handling external files as data transfer points is also in the code libraries. The file formats are those that are useful in transferring data to and from the database or to and from editors and then later to or from the database. The locking files are files that are used to indicate that a particular process holds a lock on some resource. Most of the locks used by the system are advisory.

9.0 Appendix 3: Program Listing

An asterisk in front of the program name indicates that the program is directly accessible from a Wand, or Staff menu. Two asterisks indicate that the program is executed as a direct result of a "single asterisk" program, but is not directly accessed from a Wand or Staff menu. All other programs are only used from the Unix prompt (or shell scripts). This program listing was created from the BCIMS manual index *~micro/man/whatis* with a little editing. For a complete list of BCIMS programs, please refer the manual pages shipped with the BCIMS distribution. For details on a particular program, please refer to the BCIMS manual page for that program.

Xfig (1)	menu oriented xfig executor
*acct (1)	calculate member charges for microlab use
activitygraph (1)	convert activity records to input for "graph"
*addpr (1)	add a new resource (equipment, part) to the database
*alarm	send alarm messages out over lab terminals and/or the dectalk
addresslabel (1)	create page(s) of 30 PostScript address labels.
alpha2xg (1)	convert data dumped from as200 to <x,y> data suitable for xgraph
appt (1)	make a personal appointment
as200get (1)	ASCII interface to Alphastep 200 automatic profilometer.
baseline, hotpotato, bmics, devline, proclog (1)	track wafers through the microlab.
blimp	GUI to the Blimp subsystem to monitor data from sensors for resources
blimpscan	setup sampling hardware for data acquisition
blimpwatch	parses data looking for alarm conditions and emails a warning to specified technicians or operators
blimpraw (1)	display raw sensor data from a file
*broadcast, broadcast_remote (1)	broadcast a message using a synthesized voice
*buddy (1)	sign up list for working in the lab at night
bundle	bundle specified files into a shell script
caldepend (1)	list equipment/staff maintenance calendar dependencies
*chmode (1)	change protection modes on files and directories
*ckout, ckin (1)	check out/in lab supplies

Appendix 3: Program Listing

*clruser (1)	kills a users' wand
*comments (1)	report comments and/or problems
*cont	part of blimp, a component in the continuous monitoring of blimp data
*continmon	part of blimp, a component in the continuous monitoring of blimp data
*diwatch	watch blimp sensor data concerning deionized water levels
*dolabdist	bundle and rdist BCIMS software
*dolabmake	make indicated sources for BCIMS
*epstat (1)	equipment problem status board
*eqcntl (1)	control lab equipment
*eqdepend (1)	list equipment/utility dependencies
*evac_alarm (1)	broadcast a lab evacuation alarm over the DECTalk
*faultreports (1)	menu oriented command executor
**faultrpt (1)	reports statistics concerning FAULTS
*faults, **faults.exe (1)	equipment maintenance and repair tracking system
*fax (1)	create a FAX cover sheet
filemod (1)	edit files with locking
file-monyr	appends month and year to filename.
**filerev (1)	display a file in reverse order line by line
*fixfault	shell script access to Faults allowing maintenance for a specified equipment's current problem entries.
*flat-stress (1)	compute stress from flatgage data
*flatget (1)	ASCII interface to FLATGAGE flatness measurer
**flip	X10 windows application, facility management information tool
*formpr	print out the specified laboratory form (P.O. forms for example)
**flip2fig (1)	convert FLIP format data to fig format
*gases (1)	gases program
getimage (1)	take a snapshot of a region of the screen
*hp (1)	reverse polish notation, screen oriented stack calculator
**hpget (1)	ASCII interface to HP9836
**hptalk (1)	wand interface for hpget program
**incant	controls the start-up of a "magic Wand" for each user who logs into the lab
itemin	accept terse commands to change inventory level records

iupload*	print an Ingres Quel command on stdout that can be used to upload the specified Ingres tables with data from the specified plain Unix file
*inven (1)	query and update lab inventory
isort (1)	turn copyout from INGRES into a delimited textfile
joblist	view, edit list of prioritized jobs assigned or requested according to computer account login name
jobreq (1)	fill out TCS job request form electronically
labdo (1)	program that helps manage and maintain Microlab files
labedit (1)	editor for microlab files
*labcharges	shell script that will tell how much time has been logged in the lab
*labcost	Display the amount of time that has been charged on a specified piece of equipment. Optionally specify a specific single user, time periods can be specified for a single day, week, month.
*labcostnew	script that reports this months lab costs to the archive and truncates the labcost table in the database.
*labdist	script to build the BCIMS distribution
labfinger, f (1)	query user information a la finger
labgraph (1)	display and print graphs of lab accounting history
*labhelp (1)	get help on a specific topic
*labhist (1)	list lab usage history
*labhist.s	wrapper for labhist that pipes labhist output through the less program.
*labingres (1)	provide restricted access to ingres utilities
*labmailto (1)	send personal messages to other members.
labmake (1)	method for remaking Microlab system files
*labman (1)	list lab documentation for a specified chapter
labstart	shell script that is run at the host computer boot time to start the BCIMS software system
*labtalk (1)	'talk' to somebody from inside the lab
*labusers, **labusers.exe (1)	labusers database management, an INGRES application by forms (abf)
*labwall (1)	like the UNIXwall program, posts messages on all lab terminals
*labwho, lw (1)	show who is in the lab now
*lamstart	TERM and restart or just start the lamtalkd server daemon.

Appendix 3: Program Listing

*lamtalk (1)	interface to the LAM Plasma Etcher via the SECS protocol
*late (1)	list possibly late purchase orders
*lendstat (1)	lent items status board
**lookat (1)	a screen-oriented file perusal program
luinfo (1)	list lab user information on the terminal
*maintmatrix (1)	menu oriented maintenance matrix executor
*maintreminder (1)	send preventive maintenance reminders
*makelabel (1)	create PostScript labels.
*manlog (1)	manually log in member's lab time, staff time, or equipment time
**microdate (1)	return today's date as month_day_year
**microdatget (1)	get or list data files from a remote machine
**microhash (1)	create a hashed data file from a text table
*microman, mman (1)	list documentation on specific topic
mkaliases (1)	create mailing lists of qualified users
mkcalendar (1)	send and update the list of preventative maintenance messages
mkprgroup (1)	download the process group database
mkprocesscap (1)	download the process database, and create hashed data files
mkqualify (1)	download list of qualified users
mkresourcelist (1)	download the resource database
mkschedule (1)	download a list of equipment reservation times
mkuserid (1)	download the username database
mkvendors (1)	download list of Microlab vendors
modprt (1)	print files that have been recently modified
*mtitalk (1)	UNIX interface to the MTI omnichuck
*nanoline (1)	interface to the Nanometrics line width measuring system via the SECS protocol
*nanotalk (1)	interface to the Nanospec via the SECS protocol
*nukepluto (1)	kills a pluto on a given terminal
*obf, obf.exe (1)	Microlab Objects-By-Forms system
**pip-pos (1)	position a RasterOps Picture In Picture on an X display
*pm (1)	send phone messages easily
*pmstat (1)	preventative maintenance status board
*prdata (1)	display/print Taurus port assignments

premake (1) configure a Makefile or shell script with local variables
*prgases, prquartz (1) download and print out an INGRES table
*prinfo (1) list process information on the terminal
*process (1) automated recording of processing a baseline lot
*procjob (1) process staff job board
*proget (1) ASCII interface to Prometrix resistivity measurer
*protalk (1) wand interface for proget program
*prpumps (1) print out pumps database
*purchase (1) purchase program
*qualify (1) check who is qualified on what; add new qualified users
rcmp (1) police leftover mann and cif files
**repfault, repmaint, repupdate, fixfault (1) FAULTS shell scripts
*reserve, reserve.exe (1) reserve time on lab equipment
**reservedmail (1) mail a reminder about an equipment reservation
**resleft (1) tracks how much of a resource is left
*resleftmenu (1) menu oriented command executor
**rpluto, trpluto (1) programs to remotely communicate with magic wand
*sensorgraph (1) graphical display of sensor history
*sensorhist (1) ASCII display of sensor monitoring
*sensornotify (1) turns of alarm mail from blimp sensors
*sensors (1) monitor utility sensors in facility
*sensorstat (1) sensor acquisition program
setclock (1) set the clock on a z29 terminal
setpath (1) script that sets common environment variables for Labmembers
*spc (1) Microlab Statistical Process Control system
*spcdata (1) manage Microlab SPC data batches
*spcextract (1) filter Microlab SPC data
*spcgraph (1) plot Microlab SPC graph
*spcshow (1) display Microlab SPC graph
*spcwand (1) menu oriented command interface to Microlab SPC system
*special_chemicals (1) special chemicals status board
squeeze (1) squeeze out newlines
*staff (1) menu oriented command executor

Appendix 3: Program Listing

stripmail (1)	remove irritating header lines from mail files
*tca_clean (1)	tca cleaning maintenance of tylans
*tech (1)	list technician assignments
*techjob (1)	technician job board
tellme (1)	tell me a message at a certain time
timewarp (1)	manipulate time and date
*treset (1)	reset a microlab terminal
*tychk (1)	the outlaw tylans program
*tylan (1)	menu-based interface to Tylan programs
*tylanioproblem, post_fault (1)	programs to post a fault about a tycom I/O problem.
*tyreset (1)	reset the tytan communications daemon
*tytalk (1)	UNIX interface to Tylan TYCOM controller
*tytalk2 (1)	UNIX interface to Tylan FCS10 Tube Computer
**tytasks (1)	monitor TYLAN furnaces
*tywatch (1)	watch the data from a Tylan Furnace
ubergraph (1)	a beefed-up version of the standard UNIX "graph" program
utimes (1)	update file access and modify times
**vd (1)	visual database with command structure like the editor 'vi'
**vichmod (1)	create and/or edit lab files by staff members
**vipr (1)	edit (process) files with locking
*visitor (1)	look at/add to the list of visitors who have seen the lab
wand (1)	menu oriented command executor
**wavehost (1)	identify Microlab wand server
xwd32topnm (1)	convert a 32 bit X11 or X10 window dump file into a portable anymap

10.0 Bibliography and References

[BCAM User's Manual '94]

David C. Mudie. BCAM 3.1 User's Manual (12/94). EECS/ERL Industrial Liaison Program, College of Engineering, University of California, Berkeley CA 94720.

[Hegarty, Rowe, Williams '90]

Christopher J. Hegarty, Lawrence A. Rowe, and Christopher B. Williams. "The Berkeley Process-Flow Language WIP System". Presented at SRC TECHCON '90, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley CA 94720

[Hegarty '91]

Christopher J. Hegarty. Process-Flow Specification and Dynamic Run Modification for Semiconductor Manufacturing. Memorandum No. UCB/ERL M91/40, 15 April 1991, Electronics Research Laboratory, College of Engineering, University of California, Berkeley CA 94720.

[Hodges, Rowe, Spanos '89]

David A. Hodges, Lawrence A. Rowe, Costas J. Spanos. Quality and Productivity in Semiconductor Manufacturing. 1989 EECS/ERL Research Summary. Department of Electrical Engineering and Computer Sciences, Electronics Research Laboratory, EECS/ERL Industrial Liaison Program, University of California at Berkeley.

[ILP '94]

Industrial Liaison Program Research Software 1994-1995. Department of Electrical Engineering & Computer Sciences, University of California, Berkeley CA 94720.

[INGRES '91]

Ingres/SQL Reference Manual for the UNIX and VMS Operating Systems, Release 6.4 (December 1991). INGRES Corporation, 1080 Marina Village Parkway, Alameda CA 94501.

[Lam '88]

Lam Research Corporation. Lamlink Communications Manual, Revision D1, September 15, 1988.

[Leang, Thomson, Bombay, Spanos '94]

Sovarong Leang, Shang-Yi Ma, John Thomson, Bart Bombay, and Costas J. Spanos. A Control System for Photolithographic Sequences. Submitted for publication to Transactions on Semiconductor Manufacturing, November 1994.

[Leffler, McKusick, Karels, Quarterman '89]

S. Leffler, M.K. McKusick, M. Karels, J. Quarterman. The Design and Implementation of the 4.3BSD UNIX Operating System. Addison-Wesley Publishing Company, Inc., 1989

[Microlab '93]

Microfabrication at Berkeley (informational booklet). EECS/ERL Industrial Liaison Program, College of Engineering, University of California, Berkeley CA 94720.

[Mudie '91]

David C. Mudie. Faults: An Equipment Maintenance and Repair Tracking System Using a Relational Database. Memorandum No. UCB/ERL M91/44, 23 May 1991, Electronics Research Laboratory, College of Engineering, University of California, Berkeley CA 94720.

[Picasso '90]

P. K. Schank, J. Konstan, C. Liu, L. A. Rowe, S. Seitz, B. Smith. Picasso Reference Manual. Memorandum No. UCB/ERL M90/79, 11 September 1990, Electronics Research Laboratory, College of Engineering, University of California, Berkeley CA 94720

[Resende '87]

Mauricio Guilherme De Carvalho Resende. Shop Floor Scheduling of Semiconductor Wafer Manufacturing. Ph.D. Dissertation, 1987. Department of Industrial Engineering and Operations Research, University of California, Berkeley CA 94720.

[Rowe, Williams '87]

Lawrence A. Rowe and Christopher B. Williams. An Object-Oriented Database Design for Integrated Circuit Fabrication. Memorandum No. UCB/ERL M87/43, 20 May 1987, Electronic Research Library, College of Engineering, University of California, Berkeley CA 94720.

[Rowe, Williams, Hegarty '90]

Lawrence A. Rowe, Christopher B. Williams and Christopher J. Hegarty. The Design of the Berkeley Process-Flow Language. Memorandum No. UCB/ERL M90/62, 26 July 1990,

Electronics Research Laboratory, College of Engineering, University of California, Berkeley CA 94720

[Schank, Rowe '92]

Patricia K. Schank and Lawrence A. Rowe. An Introduction to Semiconductor Manufacturing and Markets. Memorandum No. UCB/ERL M92/35, 7 April 1992, Electronics Research Laboratory, College of Engineering, University of California, Berkeley CA 94720.

[SEMI '92]

1992 SEMI International Standards, Equipment Automation/Software Volume. 805 East Middlefield Road, Mountain View CA 94043.

[Sharma '88]

Amit Sharma. "Berkeley Fabrication Facility Monitoring System". Master of Science Research Project Report, 1988. Department of Electrical Engineering and Computer Sciences, University of California, Berkeley CA 94720

[Smith, Rowe '91]

Brian C. Smith and Lawrence A. Rowe. An Application-Specific Ad Hoc Query Interface. Memorandum No. UCB/ERL M90/106, 21 November 1990 (Revised 23 May 1991), Electronics Research Laboratory, College of Engineering, University of California, Berkeley CA 94720.

[Sun '90]

Sun Microsystems Inc. "Network Programming Guide" (on the client server model). SunOS 4.1, Revision A of 27 March, 1990, Sun Microsystems Inc., Mountain View CA.

[Voros, Ko '89]

Katalin Voros and Ping K. Ko. Evolution of the Microfabrication Facility at Berkeley. Memorandum No. UCB/ERL M89/109, 23 September 1989, Electronics Research Laboratory, College of Engineering, University of California, Berkeley CA 94720.

[West '89]

Alex C. West. FLIP: A Graphic User Interface for Management and Utilization of Facilities. Memorandum No. UCB/ERL M89/39, Electronics Research Laboratory, College of Engineering, University of California, Berkeley CA 94720.

Additional Reading

[Aho, Sethi, Ullman '86]

A. Aho, R. Sethi, and J. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley Publishing Company Inc., Reading MA, 1986.

[Silberschatz, Peterson '89]

A. Silberschatz and J.L. Peterson. Operating System Concepts (Alternate Edition). Addison-Wesley Publishing Company Inc., Reading MA, 1989.

[Teorey '90]

Toby J. Teorey. Database Modeling and Design: The Entity-Relationship Approach. Morgan Kaufman Publishers, San Mateo CA 94401, 1990.

[Yourdon '88]

Edward Yourdon. Managing the System Life Cycle (2nd Edition). Yourdon Press, Englewood Cliffs, New Jersey, 1988.

11.0 Acknowledgments

The vision to transform the Berkeley Microfabrication Laboratory into a fully computer integrated facility was provided by Professor David A. Hodges along with Professor Lawrence A. Rowe, of Electrical Engineering and Computer Sciences (EECS), University of California at Berkeley. Together with their students and staff participants, their vision has become a reality in BCIMS. Professor Roger C. Glassey of Industrial Engineering and Operations Research, and his students, have also taken part in the area of shop floor scheduling. Professor Costas J. Spanos and his students have, and continue to make major contributions to the evolution of automation in the Microlab by producing and integrating the subsystems for statistical process control. The Microlab's manager, Katalin Voros has guided the integration of the various projects into standard use in the Microlab. Her support in providing user feedback and facilitating the input from the Microlab's users and staff has been invaluable. Graduate students who have participated in the BCIMS project are (in chronological order):

Michael Klein (Hodges)
Christopher Williams (Hodges, Rowe)
Mauricio Resende (Glassey)
Christopher Hegarty (Rowe)
Brian Smith (Rowe)
Amit Sharma (Hodges)
Norman Chang (Spanos)

Undergraduate assistants who have taken part in the BCIMS project are:

Thomas Muller
James Hopkin
Scott Miles
Alex West
Adhi Gaduh
Eric Ng
Vadim Gutnik

Acknowledgments

Professional staff who have contributed to BCIMS are:

David Mudie (software design, engineering and systems analysis)

Lauren Massa (software design, engineering and systems analysis)

Christopher Hylands (systems administration)

Mark Kraitchman (software design, systems administration)

We gratefully acknowledge the generous support for this research from the Semiconductor Research Corporation, the National Science Foundation, California MICRO, and Microlab operating funds.

12.0 Author Acknowledgment

I would like to thank Dean David A. Hodges for his generous encouragement and for his astute advice. It has meant a lot to me and made a real difference in my life. Professor Lawrence A. Rowe has my thanks for tirelessly reviewing and offering excellent editorial advice. I am sure that I am a better writer today for his efforts. Professor Costas J. Spanos has my gratitude for his very generous support of my professional and academic endeavors, and for having a sense of humor. I would also like to thank Katalin Voros for her extensive review and suggestions.

David Mudie and Christopher Williams have my appreciation for helping me in my early days with BCIMS. I would especially like to thank David Mudie for all his help and advice.

Lastly, but most importantly I would like to thank my husband Mark for putting up with my hectic schedules and my frequent neglect of my half of the chores.