FAULTS: AN EQUIPMENT MAINTENANCE AND REPAIR SYSTEM USING A RELATIONAL DATABASE

by

David C. Mudie and Norman H. Chang

Memorandum No. UCB/ERL M90/95

18 October 1990

FAULTS: AN EQUIPMENT MAINTENANCE AND REPAIR SYSTEM USING A RELATIONAL DATABASE

by

David C. Mudie and Norman H. Chang

Memorandum No. UCB/ERL M90/95

18 October 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

FAULTS: An Equipment Maintenance and Repair System Using a Relational Database†

David C. Mudie & Norman H. Chang *

Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

FAULTS combines a forms-based user interface with a facility-wide relational database to record equipment maintenance and repair events as they occur. The semantics of preventive maintenance (PM) and repair events are formalized to create clear and unambiguous maintenance reports.

Storing equipment PM and failure information in an organized database has several benefits. Accumulated information is automatically indexed to aid diagnosis of failures as they occur. Equipment failure information is available to other utility programs for display and statistical analysis, to produce summaries such as preventive maintenance intervals, mean time between failures, predicted downtimes, performance trends, etc.

FAULTS is scheduled to enter common use at the Berkeley Microfabrication Facility in August 1990. Lab technicians have entered knowledge of over a hundred different pieces of equipment into the database. The system is expected to provide a significant improvement in the way information is used for the management of preventive maintenance and equipment repairs.

1. Introduction

The current practice of equipment maintenance in the semiconductor industry consists of three activities: *equipment qualification*, *preventive maintenance*, and *field repair*. Equipment qualification includes a number of tests that must be completed before the equipment is released to production. Preventive maintenance is conducted at regular intervals and consists of routine inspection and cleaning. Unscheduled field repairs are often necessary during production to resolve equipment malfunctions.

Today, most of the information concerning *equipment qualification* and *preventive maintenance* is collected on paper records. Operators or technicians must refer to the appropriate records while conducting these activities. Information concerning *field repairs*, however, is usually managed in an ad-hoc fashion. Malfunctions are reported orally or recorded in a free-style text report by the operators. Even when using a paperless, computerized system such as COMETS [1] (a popular work-in-progress system) or BLIS (the Berkeley Laboratory Information System), the causes and actions taken to correct malfunctions are lost in long unstructured reports. Since this information has no formal structure, it is difficult to perform even simple

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

* Norman Chang is currently at HP Labs, Palo Alto, California

analysis to discover and correct common causes of malfunction.

In this paper we present a new scheme for managing the information concerning equipment maintenance and repair activities. Our solution to the above problems is designed with four goals in mind. First, the system must help facility management keep track of equipment maintenance events as they occur. Second, the system should help maintenance technicians conduct off-line equipment maintenance by reviewing previous maintenance cases. Third, the system must be usable and maintainable by equipment operators, maintenance technicians and facility managers without need of assistance from expert programmers. Finally, the system should cooperate with automated diagnostic systems by maintaining failure histories of equipment in the facility and by providing a mechanism that allows a diagnostic system to automatically schedule necessary maintenance tasks [2].

The result is the Berkeley **FAULTS** system. **FAULTS** is a program that allows a user to interact with a relational database [3] through a forms-based interface to record preventive maintenance and field repairs of manufacturing equipment. All the information necessary to identify the type and cause of equipment failure, time of occurrence, time of repair, etc. is stored in the relational database. Unlike existing paperless systems, this information is organized in symbolic form so that other application programs can be developed to perform facility-wide analysis.

The following section of this paper provides an example scenario demonstrating how the **FAULTS** system meets the first three goals outlined above. Section 3 describes the data structures used to model maintenance and repair activity, and briefly discusses how the implementation of these structures meets the fourth goal of our system. Section 4 concludes the paper with a discussion of our results.

2. An Application Example

This section presents an example of how the **FAULTS** system is used throughout the lifetime of a typical equipment repair event. In our facility, the lifetime of a repair event (commonly referred to as a *problem*) is marked by three steps: *reporting, diagnosis,* and *clearing*. Additionally, once an event is recorded by the system, it may be *analyzed* by a diagnostic program or report generator.

2.1. Problem Reporting

During an LPCVD (low pressure chemical vapor deposition) processing step, Bob (the operator) observes that the display terminal of the TYCOM (Tylan furnace controller) on the LPCVD furnace has suddenly stopped responding to his commands. Bob tries the "reset" command he knows, but the TYCOM terminal still does not respond. Bob switches over to an ASCII computer terminal next to the malfunctioning TYCOM, and invokes the **FAULTS** equipment maintenance system to report the problem.

Figure 1 shows the first screen of the equipment maintenance system. This screen is called the *Equipment Status Board.* Like other **FAULTS** screens, it composed of an information display occupying most of the screen and a list of available operations at the bottom. The Equipment Status Board lists problems reported for all equipment in the facility. The Status Board currently shows a problem on the LAM plasma etcher, but Bob's TYCOM problem has not been reported yet. Bob executes the "New" command to report a new problem.

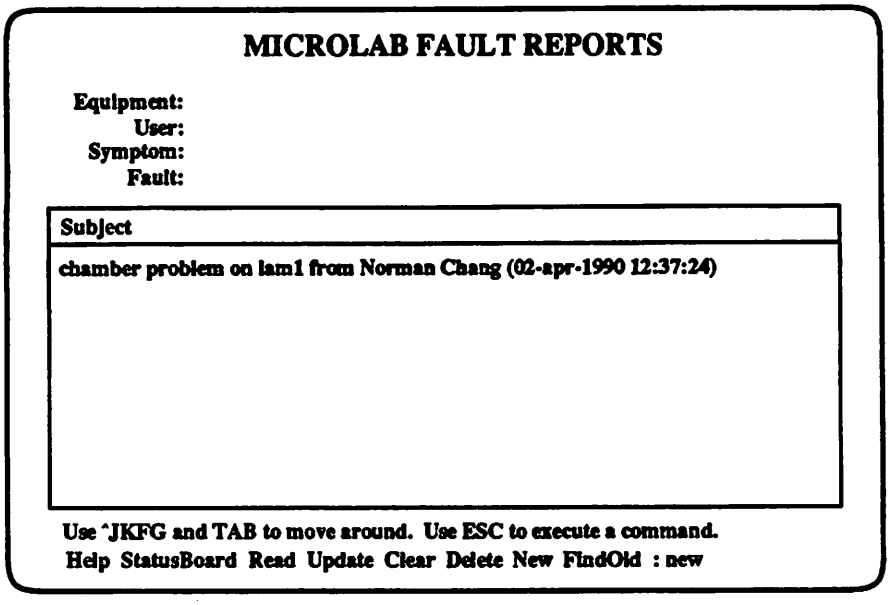

Fig. 1. The Equipment Status Board. Users can browse current problems, initiate a new report, or examine old reports.

Bob is prompted for the name of the equipment he is working on ("tylan16") and then is asked to describe the malfunction he is reporting. **FAULTS** presents a menu of problems known to occur on the LPCVD furnace tube, and Bob selects the entry most descriptive of his problem: a "terminal" problem within the group of "tycom" errors. Figure 2 shows the screen during this selection process. "Escape" commands are available if none of the menu items seem correct or if Bob decides to cancel the report.

When Bob has identified the observed symptoms of his problem, the screen changes to that shown in Figure 3. This screen confirms information such as the name of the malfunctioning equipment and the date the report is being filed. A short summary line is automatically built from known information to help technicians identify this report at a glance. Bob is prompted to type in free-form text comments so he can report on the particular circumstances of this failure or describe symptoms not found in the previous menus.

When Bob is done entering comments, the report is released as a "pending" problem. **FAULTS** adds the new report to the Equipment Status Board, and automatically notifies the responsible technician and supervisor via electronic mail. Bob can also send "carbon copies" of the report to himself or other operators of the equipment.

2.2. Problem Diagnosis

Kate, the technician responsible for tylan16, receives electronic mail describing Bob's problem on the machine. Kate is unfamiliar with the TYCOM control system, so she invokes the **FAULTS** system on her workstation and executes a command to find previous occurrences

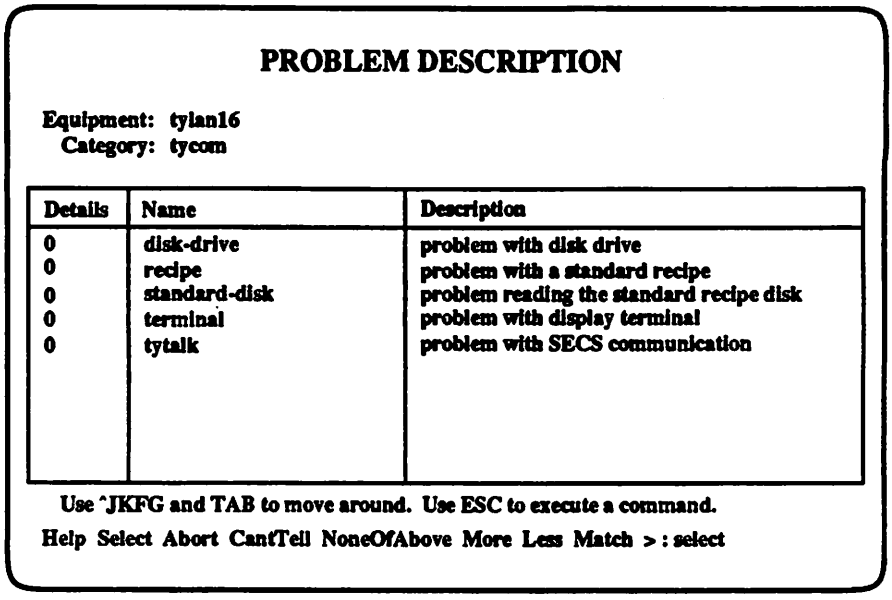

Fig. 2. Identifying problem symptoms. An operator observes a TYCOM malfunction (the control screen has no response) and selects the **terminal** symptom from the Tylan furnace menu.

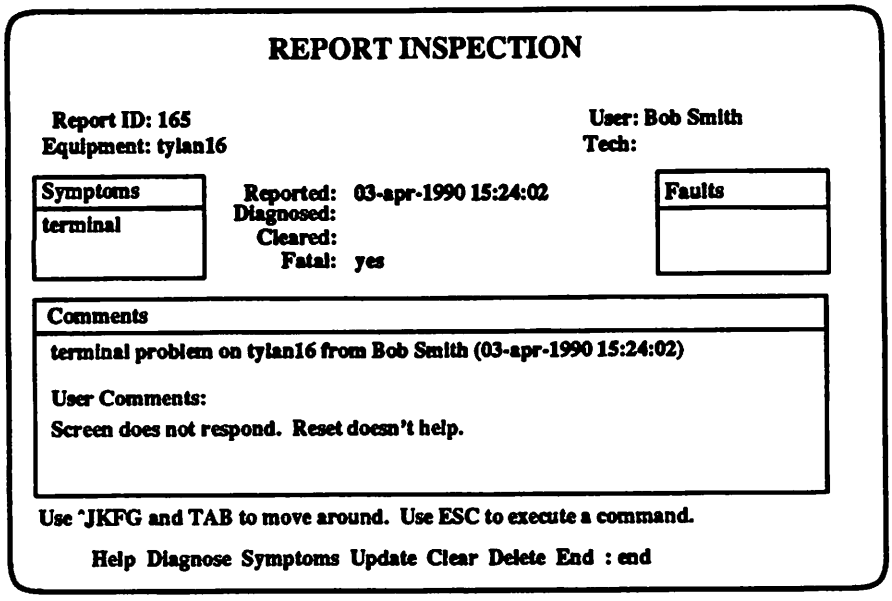

Fig. 3. Completing a problem report. The report can be checked and modified by the operator before it is released to the Status Board.

of similar failures. The system locates several reports that may be relevant, as shown in Figure 4.

Kate browses the contents of the old reports, as shown in Figure 5, and learns that on a previous occasion the TYCOM display screen had locked up because of a faulty microchip in the controller. Armed with this knowledge, Kate inspects the furnace control system and discovers that one of printed circuit boards in the controller is indeed defective. She orders a replacement board from the equipment manufacturer and is told that the new board will take four days to arrive.

Still within the **FAULTS** system, Kate records her *diagnosis* of the problem by choosing from a menu of faults known to occur on the Tylan furnace tubes, as Bob chose from a menu of symptoms. This process is shown in Figure 6. Kate also enters a brief note stating that the furnace will remain down for another four days while waiting for the replacement part. The revised report is "posted" to the Equipment Status Board so operators will be aware that the furnace is out of commission.

2.3. Problem Clearing

A week later, the replacement circuit board arrives. Kates fixes the controller and verifies that the furnace tube is operating normally. She enters the **FAULTS** system and executes a command to *clear* the problem report. **FAULTS** prompts Kate to confirm her diagnosis of the faulty controller board, then asks her to type in more text comments describing the repair procedure and any peculiar circumstances of the repair. Kate notes that the replacement board took longer to arrive than promised. When her comments are complete, **FAULTS** removes the report

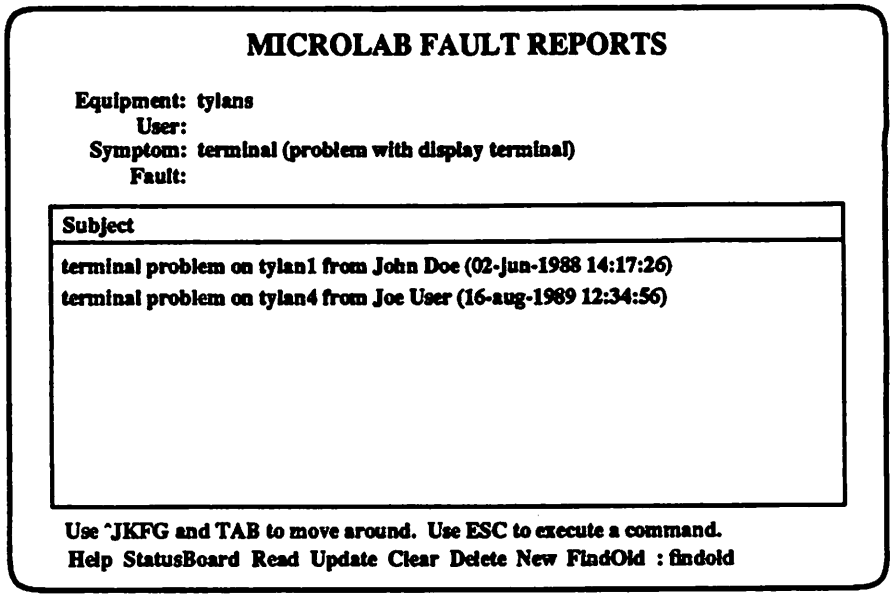

Fig. 4. Locating old reports. **FAULTS** queries the database for reports concerning the specified symptom.

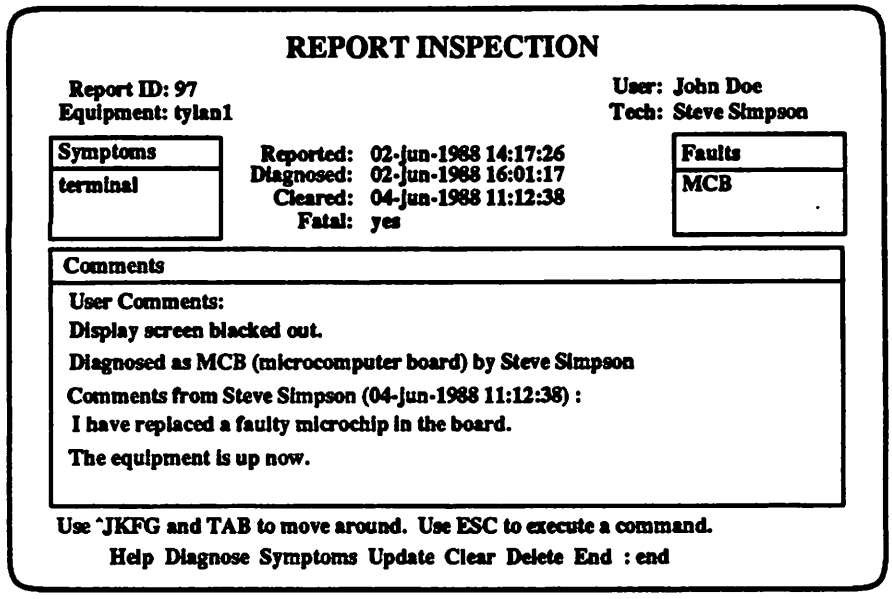

Fig. 5. Browsing a problem report. This screens displays all the information concerning a particular maintenance event. The text comments may be scrolled to read farther.

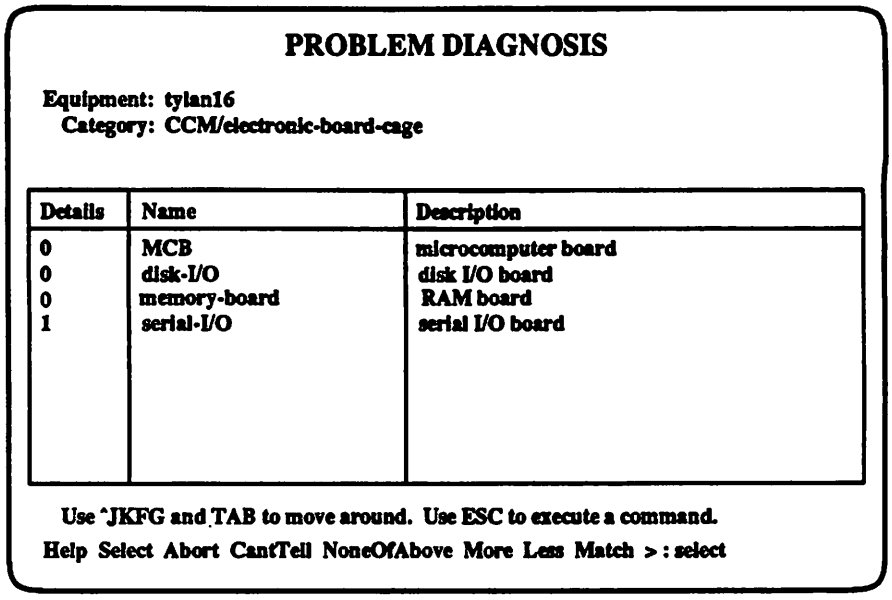

Fig. 6. Identifying equipment faults. **Electronic-board-cage** (defective electronic board cage) was selected under the **CCM** (central control module) category. Finally, the technician chooses **MCB** (defective microcomputer board) from the four choices under **electronic-board-cage.**

from the Status Board and sends out electronic mail announcing that the furnace is back in operation.

This example demonstrates the day-to-day use of **FAULTS** to manage current equipment problems and to maintain a history of previous equipment failures. **FAULTS** also provides a set of command screens to update the contents of each fault and symptom menu. This allows qualified technicians to create new problem categories when the need arises, without requiring the support of a database expert. Each command screen in **FAULTS** has an accompanying *help* screen describing the command options available. (As an example, the help screen for the Equipment Status Board is shown in Figure 7.) These on-line screens help novice users learn to operate the system without need of a bulky manual.

2.4. Failure Analysis and Downtime Statistics

Information accumulated in the **FAULTS** database is available to independent analysis programs as well as to human operators. One important function of these programs is to summarize equipment maintenance information. For example, facility managers often ask "How much equipment downtime occurred in the last three months?", "How much money did we spend on furnace maintenance last year?", and so on. Technicians frequently ask questions such as "What was the failure last time this symptom was observed?" or "How often does the TYCOM controller fail on this furnace?" Since the information associated with each equipment maintenance event is stored in a well-defined relational format, analysis programs and report generators can query the **FAULTS** database to produce the answers to these questions.

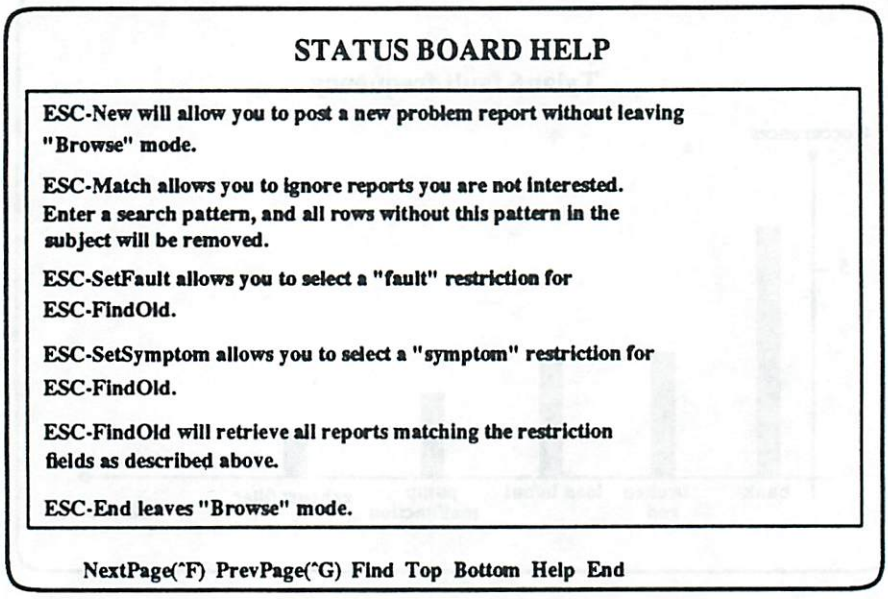

Fig. 7. Example of a **Help** screen. Such a screen can be called up from any of the **FAULTS** command screens.

Simple plotting of various downtime statistics such as equipment downtimes and failure frequencies (a ''Pareto'' chart) can give powerful visual summaries of the cleanroom condition. As an example, Figure 8 summarizes the failures that occurred on our Tylan furnaces during the last year.

3. Design and Implementation of the FAULTS Data Model

The ''Report Inspection'' screen (Figure 5 in the previous scenario) gives some indication of the amount of information associated with each equipment maintenance event that occurs in a microfabrication facility. Details such as the date of occurrence, the operator and technician involved, and some description of the maintenance required must be captured and stored for future analysis. To support efficient storage and retrieval of this information, our record-keeping system must coerce the information surrounding each event into a clear and unambiguous format. We derive this format by defining the semantics of an equipment maintenance event and identifying the information that must be captured. In this section, we describe the data abstractions used by **FAULTS** to represent an equipment maintenance event, and we outline how this data is stored and manipulated in a relational database.

3.1. The FAULTS Data Model

An object-oriented data model was used to design the **FAULTS** database. The database uses five major data types: *labusers, resources, comments, reports,* and *faultsymps*. The first four items are straight-forward collections of the information associated with real-world objects.

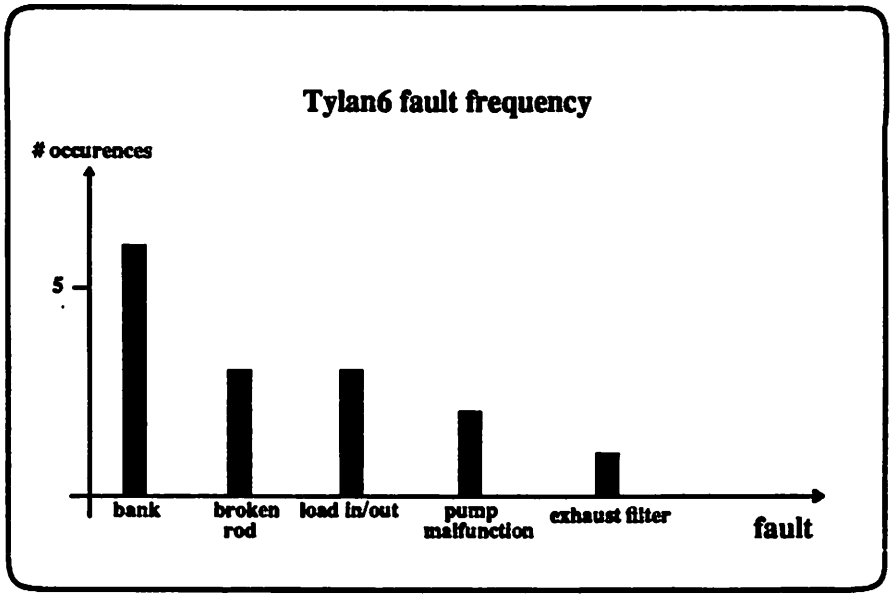

Fig. 8. A histogram showing fault frequencies on Tylan furnaces over the last year.

The fifth data type, *faultsymps*, models diverse equipment failure modes. It has no real-world counterpart. We present each of these data types in turn and discuss the relationships that exist between them. Figure 9 shows an Entity–Relationship diagram of the **FAULTS** data model.

Labusers

A *labuser* is a person involved with the microfabrication facility. (At U.C. Berkeley, we refer to our facility as the "MicroLab" hence *labuser* instead of *fabuser*.) Administrative staff, technicians, and equipment operators are all *labusers*. Two attributes are stored about each labuser:

- *name*: First and last name of the user, e.g. "Bob Smith"
- *labuser ID*: A unique number to identify this labuser

Resources

A *resource* is an item that is used in the microfabrication process. In general, *resources* include chemicals stocked by the facility, tweezers used to handle wafers, and many other items. **FAULTS** is only concerned resources classified as *equipment*. Three attributes are stored about each resource:

- *name*: A brief name identifying the resource, e.g. "tylan16"
- *class*: The general group this resource belongs to, e.g. "equipment"
- *resource ID*: A unique number to identify this resource

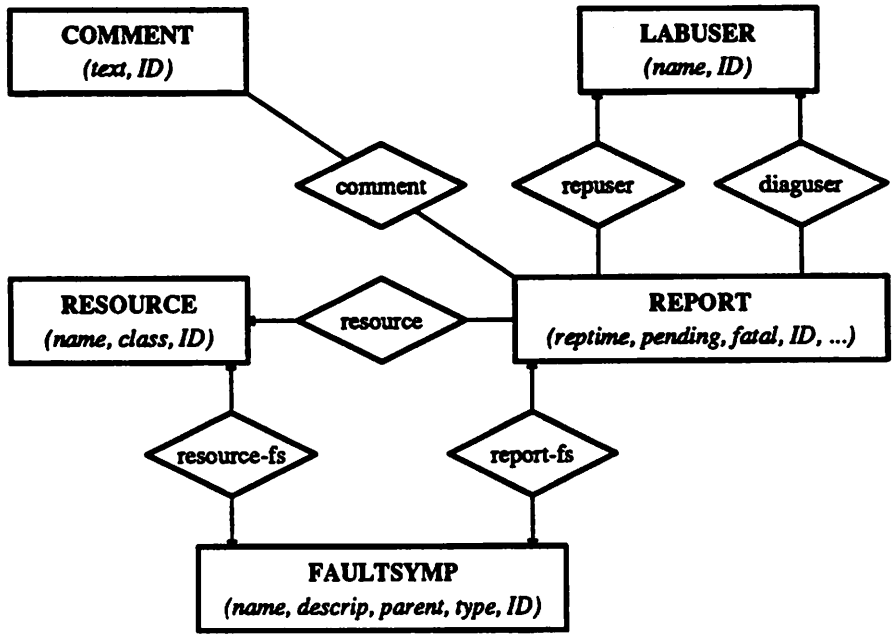

Fig. 9. Entity–Relationship Diagram of the **FAULTS** database.

Boxes indicate database objects
Italics indicate attributes of an object
Diamonds indicate relationships between objects
An arrowhead follows a "many-to-one" relationship

Comments

A *comment* is a block of ASCII text of arbitrary length. *Comments* are used to store unformatted text entries for later retrieval. Two attributes are stored about each comment:

- *text*: The contents of this text block
- *comment ID*: A unique number to identify this comment

Reports

Each *report* represents a single equipment maintenance event. A *report* object stores all the data unique to this event, such as date of occurrence and current status, and makes reference to objects such as *labusers* and *resources* that may be associated with more than one event. The following attributes are stored about each report:

- *reptime*: The date this event was entered into the system
- *diagtime*: The date this event was diagnosed by a technician
- *cleartime*: The date this event was cleared from the Status Board
- *pending*: True if this event is still on the Status Board

- *fatal*: True if this event prohibits operation of the equipment
- *repuser*: A reference to the *labuser* who entered this event
- *diaguser*: A reference to the *labuser* who diagnosed this event
- *resource*: A reference to the *resource* this event occurred on
- *comment*: A reference to the *comment* storing text for this event
- *report ID*: A unique number to identify this report

Faultsymps

To cross-reference similarities between equipment maintenance events, **FAULTS** requires a clear and unambiguous method to describe the nature of each such event. Our solution is a hierarchical structure of *fault* and *symptom* categories, with an entry for each situation that can occur.

The qualitative information that describes a maintenance is divided into two conceptual groups: *faults* and *symptoms*. A *symptom* is defined as an occurrence that may be observed by an operator or diagnosis module when an equipment malfunctions. An operator might observe the *symptom* that the motorized wafer boat has gotten stuck during a furnace run. A *fault* is defined as the underlying cause behind an equipment malfunction. *Faults* are first identified by a maintenance technician's diagnosis, then verified by the person who actually repairs the equipment. As an example, the "wafer boat stuck" symptom might have been caused by a "boat controller" fault.

To avoid ambiguity of terms, operators and technicians must agree upon a standard terminology for each of the faults and symptoms that can occur on equipment. **FAULTS** enforces this standard by identifying each fault or symptom with a single descriptive keyword. Users then select from a menu of known keywords, rather than describing the problem in their own language. Each keyword is paired with a long description of the fault or symptom to make identification easier.

The faults and symptoms for a specific piece of equipment are logically grouped into hierarchical categories to simplify the search for a particular entry. In order to avoid duplication of information, hierarchies can be extended upward to include clusters of similar equipment. These hierarchical categories are developed by consulting with an expert technician for each piece of equipment. A technician with sufficient privileges can modify the hierarchy of categories without leaving **FAULTS** if he discovers a problem category that was not foreseen by the expert. An example of these hierarchical structures is shown in Figure 10.

When specialized equipment of the same type are grouped together in a cluster, the corresponding specialized faults and symptoms are earmarked for that particular equipment while others remain generic throughout the cluster. As an example, hierarchical categories are created for a furnace cluster that contains 16 individual reactors. Among these, there are 10 different kinds of reactors that use different sets of gases over various temperature and pressure ranges. **FAULTS** will only display the fault and symptom entries appropriate to the reactor in use.

Faults and symptoms share many of the same properties, so they are abstracted into a single data type called a *faultsymp*. The following attributes are stored about a *faultsymp*:

- *name*: The keyword name of this faultsymp
- *description*: The long description of this faultsymp
- *type*: The type of this faultsymp, either "fault" or "symptom"

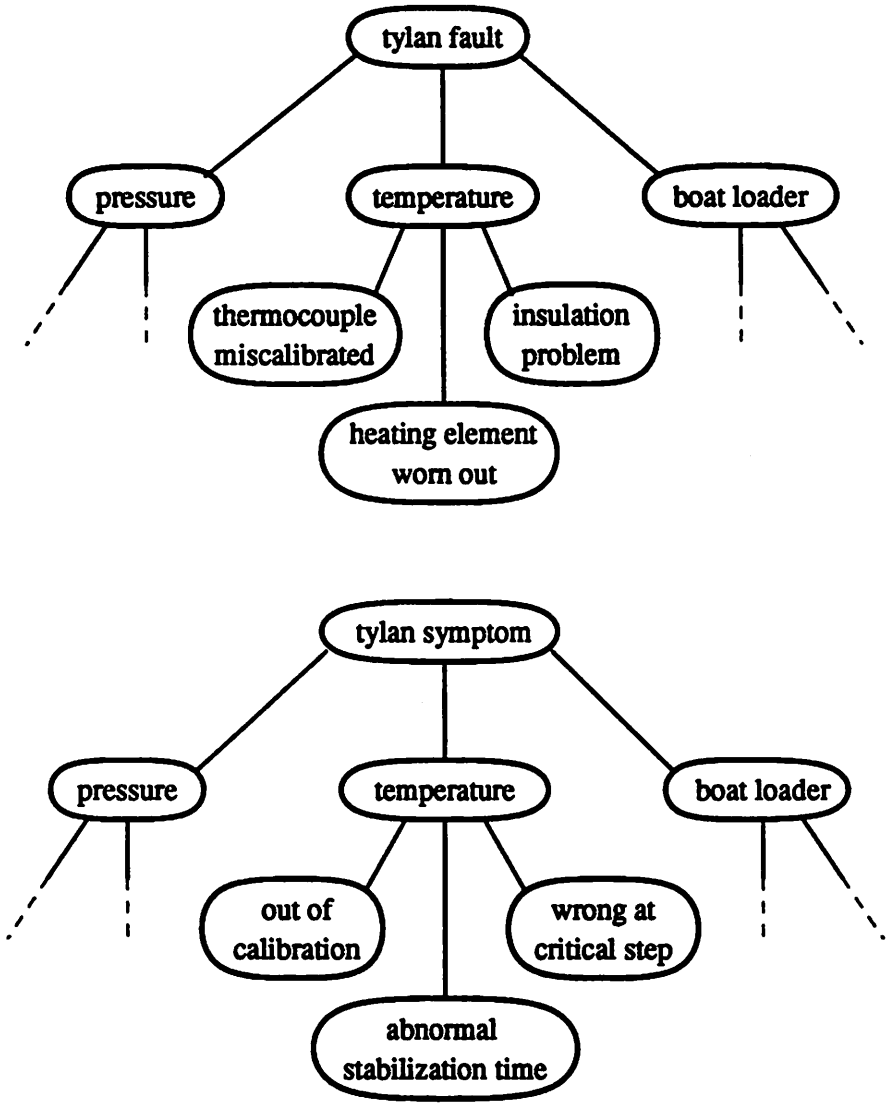

Fig. 10. An example of *fault* and *symptom* hierarchies.

- *parent*: A reference to this faultsymp's superior in the hierarchy of categories
- *faultsymp ID*: A unique number to identify this faultsymp

Additionally, **FAULTS** recognizes a many-to-many relationship *report-fs* that associates *faultsymps* with the *reports* they describe, and a many-to-many relationship *resource-fs* that associates specialized *faultsymps* with the *equipment resources* on which they occur.

3.2. Implementation of the FAULTS Data Model

The data types defined above are stored in a commercially available relational database. This database contains a table to represent each of the five data types, along with numerous index and cross–reference tables to define how data objects are associated with each other.

To insulate programs from the details of database implementation, we created a library of interface procedures called **faultlib**. This library is used to achieve **FAULTS'** fourth major goal of cooperation with other programs.

The forms-based user interface of **FAULTS** is itself built using the **faultlib** primitives. Diagnostic systems use **faultlib** to initiate problem reports and query equipment history without specific knowledge of the database structures. Several other applications use the **faultlib** library, including a prototype equipment performance monitor and a report generator that compiles statistics of equipment downtime and fault frequency.

FAULTS and **faultlib** were written using the **C** programming language, the **SQL** query language and **OSL**, a high-level user interface language used with the INGRES Corporation's **INGRES/ABF** database system [4]. **faultlib** is built from 2000 lines of **C** and **SQL** code, while the command screens of **FAULTS** user interface require another 2100 lines of **C**, **SQL**, and **OSL** code.

4. Current Status and Results

The prototype **FAULTS** system will be integrated into the Berkeley Lab Information System in August 1990. The new information management system will handle all equipment maintenance events in the facility. The old text-based system is automatically maintained for read-only browsing and archival purposes. Fault and symptom hierarchies have been created for over a hundred pieces of equipment in the facility.

Initial testing indicates that the **FAULTS** system will successfully provide technicians and management with a tool for monitoring equipment maintenance events, reviewing equipment history, and resolving equipment malfunctions. Downtime statistics and equipment failure rates can be calculated daily to apprise management of equipment performance trends and maintenance that should be performed soon.

Acknowledgements

The authors would like thank Katalin Voros, Robert Hamilton, and the staff of the U.C. Berkeley MicroLab for donating their valuable knowledge, time, and patience to this project. We would also like to thank Gary May for his help with many of the fault and symptom hierarchies, and our research advisors David A. Hodges, Lawrence A. Rowe and Costas J. Spanos for their support and guidance.

References

[1] *COMETS Reference Manual*, Consilium, 1990.

[2] N. H. Chang, C.J. Spanos, *Chronological Equipment Diagnosis with Evidence Integration*, SPIE Conference on Applications of Artificial Intelligence VIII, 1990.

[3] H.E. Korth, A. Silberschatz, *Database System Concepts*, McGraw-Hill, Inc.,1986.

[4] *INGRES/ABF Reference Manual*, Ingres Corporation, 1988.