UCRL- 82962
PREPRINT

CONF-791102--141

MASTER

CONTROL AND DIAGNOSTIC DATA STRUCTURES FOR THE MFTF

J. A. WADE
J. H. CHOY

This paper was prepared for submittal to the 8th SYMPOSIUM ON ENGINEERING PROBLEMS OF FUSION RESEARCH; IEEE; SHERATON HOTEL, SAN FRANCISCO, CA., NOVEMBER 13-16, 1979

11-12-79

Lawrence Livermore Laboratory

CONTROL AND DIAGNOSTIC DATA STRUCTURES FOR THE MFTF*

J. A. Wade and J. H. Choy
Lawrence Livermore Laboratory, University of California
Livermore, CA 94550

A Data Base Management System (DBMS) is being written as an integral part of the Supervisory Control and Diagnostics System (SCDS) of programs for control of the Mirror Fusion Test Facility (MFTF).[13] The data upon which the DBMS operates consist of control values and evaluative information required for facilities control, along with control values and diagnostic data acquired as a result of each MFTF shot. The user interface to the DBMS essentially consists of two views: a computer program interface called the Program Level Interface (PLI) and a standalone interactive program called the Query Level Interface to support terminal-based queries. This paper deals specifically with the data structure capabilities from the viewpoint of the PLI user.

Introduction

Not only is MFTF large in terms of physical size and number of subsystems, but also significantly more data must be acquired and archived than on previous fusion experiments (see Fig. 1). The question then is not what data to acquire, but how to present the data in an easily usable form. As with prior experiments, the difficulty lies not in acquiring data (in terms of both what to gather and how much), but rather in the tremendous volume that must be analyzed and reduced. From the standpoint of operations, the facility must be continually measured so that operations may progress. In addition, the goal of MFTF is not only continued operation, but understanding the plasma physics associated with the project. Diagnostic measurements utilizing both current and prior shots must be analyzed and reduced.

A DBMS that has been specifically adapted for MFTF--rather than specialized data manipulation routines incapable of future expansion--is being designed and implemented. Simply defined, a DBMS is a set of software tools that allows users to operate upon their data in a manner that is "close" to the way the data is thought of. This frees the user from concern with such incidental questions as disk files, disk addresses, etc. The data base is that collection of data known to the DBMS; for MFTF, this includes such items as set points for various subsystems; facilities measurements, such as valve settings, temperatures, vacuum levels, and neutral-beam conditioning status; control parameters required for operation of diagnostic instruments; and diagnostic data acquired as the result of a shot. In addition to current data (i.e. data required for continued operation of MFTF), the data base also contains archived data (i.e. data retained from prior shots).

Historical Perspective

The advantages of providing a DBMS for MFTF[1, 2] include reduced redundancy, increased consistency, and greater data independence. Instead of requiring each computer program to design, build, read, and write data in its own unique set of private files, the amount of redundant data can be reduced by centralizing the data in a standard form. Secondly, with an integrated data base, inconsistencies can be minimized. (The possibility of an inconsistency exists when the same data are stored in more than one place.) Third, a DBMS provides the capability for data independence. Usually, when a program is written that requires access to external data, the programmer builds a set of file-access mechanisms into the program, defining specifically how and where data are to be represented. Should the access methods need changing at a later date, the program must also be changed. These data-dependent problems are removed by inserting an interface that differentiates between the way the user views the data and the way the data are actually stored.

Although an available DBMS would seemingly suffice for MFTF, three factors rule out this possibility. First, a reference to read in the set-point portion of the data base must occur very quickly--returning within 4 ms, given certain constraints. Secondly, the amount of returned data can vary from as little as one character to as much as 32000 small integers (obviously, the greater the volume of data, the longer the data-base access). Finally, the environment within which the DBMS is to operate consists of several computers and their associated peripherals; data may or may not exist in the computer that contains the program requesting the data. These factors--taken individually--are not necessarily sufficient to warrant building a new DBMS, however, the combination is sufficient, especially in view of the results of extensive benchmarks run on the available systems[14].

A list of component parts for the DBMS currently under development is as follows:

1. The Program Level Interface (PLI) which provides a method of accessing the data base from computer programs that exercise control over MFTF and acquire and process diagnostic data resulting from a shot (see Fig. 2).
2. The Query Level Interface (QLI), which provides a method of accessing the data base from interactive computer terminals.
3. The specialized utility programs that perform such operations as saving shot data on magnetic tape and inserting prior shot data into the online data base from tape.

Hardware Overview

As Fig. 1 shows, the DBMS is implemented on nine Interdata computers (four 8/32's and five 7/32's) that are interconnected via a multiport shared memory. Each computer has its own local memory and disk storage; the 7/32's each have a 10-megabyte disk, and the 8/32's each have an 80-megabyte disk with one 300-megabyte disk and an additional 10-megabyte disk installed on one of the 8/32's. As auxiliary storage, two of the 8/32's each have one 1600-BPI, 75-IPS tape drive. The shared memory is arranged as two 64-kilobyte blocks. Seven of the nine computers incorporate MFTF operator's consoles designed for MFTF functions. (For a further explanation of the computer hardware, see the paper by Butner in these proceedings.[3] For details of the MFTF operator consoles, see the paper by Speckert in these proceedings.[4])

*Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract number W-7405-ENG-48.

[illegible]

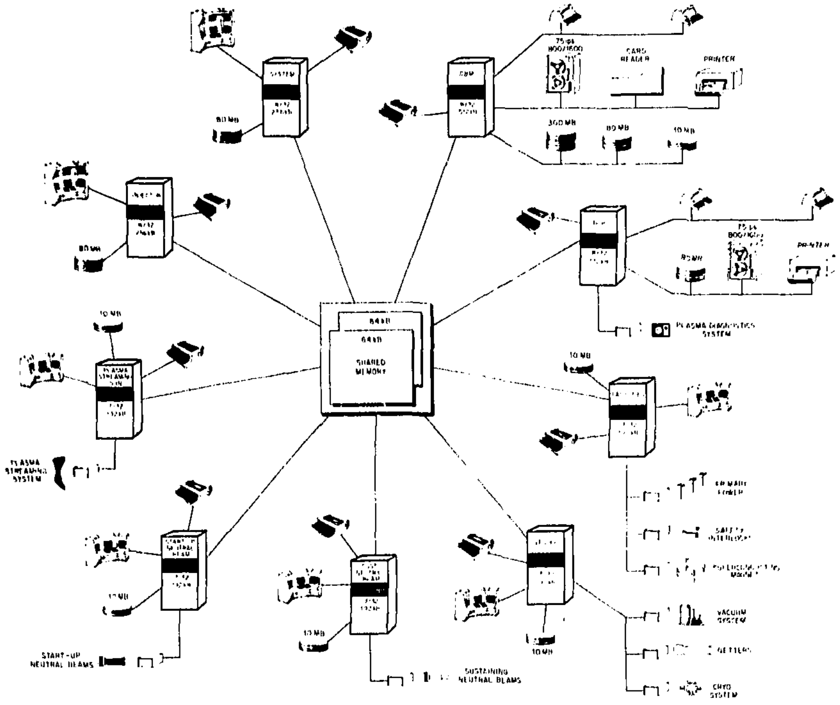

Fig. 1. MFTF control and diagnostics system.

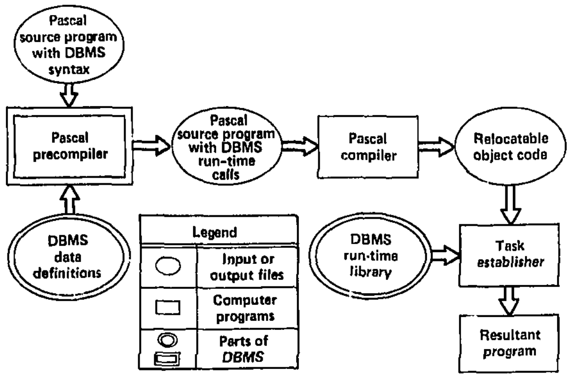

Fig. 2. Using the DBMS program level interface.

Each computer supervises one of the following major MFTF functions:

- Overall MFTF supervision
- Injector subsystems supervision
- Startup neutral beams
- Sustaining neutral beams
- Plasma streaming
- Vessel supervision
- Facilities supervision
- Data base management
- Diagnostic data processing

(For a further explanation of the overall MFTF system, see the papers by McGoldrick[5], Wyman[6], and Ng[7], in these proceedings.)

MFTF Data Base

Overview

As Fig. 3 shows, the MFTF data base may be logically divided into two main categories: data necessary for control of MFTF and data associated with plasma diagnostics. The facilities control portion is further divided into facility set points and facility evaluation. Facility set points include such data as voltages, currents, valve settings, neutral-beam aiming parameters, and timing duration. Facility evaluation data include current valve settings, temperatures, vacuum levels, and neutral-beam conditioning. The plasma diagnostic data may also be separated into two areas: instrument control parameters and diagnostics acquisition.[8] Data in the instrument control portion consist of the same types of information found in the facility set points; however, they are associated with the diagnostic instrumentation, rather than control of the facility. Similarly, the diagnostics acquisition includes data read from the diagnostics instruments during an MFTF shot. This specific area of data accounts for the requirement that the DBMS be able to handle four megabytes of data per five-minute shot cycle.

In view of the available hardware, it is important to emphasize that the MFTF data base is distributed across the entire set of computers--some data reside on the disk(s) of each machine, other data reside in the local memory of each machine, and still other data reside in the common shared memory. The reason for the distribution is that the associated MFTF functions are themselves distributed. The computer responsible for startup neutral-beam processes, for instance, need not know about that portion of the data base involving cryogenics, nor need there be interaction between certain aspects of the plasma-streaming system and personnel interlocks.

In addition to a division of data by subsystem, there are also various frequencies of access to the data base. For example, when the MFTF tank is being pumped down, most data associated with the injection system need not occupy space in memory. On the other hand, when a shot is in progress, data associated with pumpdown procedures need not occupy space. As a general rule in the DBMS, no space is allocated in memory for currently unused portions of the data base. Since data may be conveniently divided by subsystem, currently-in-use data associated with the sustaining neutral-beam system exist in the local memory of their respective computer. By contrast, overall MFTF timing data would exist in the central shared memory because of their more global importance.

The data residing in the data base have the following characteristics: There is a distinct parallel between data in the facility set-points area and data in the instrument control parameters area, since both are represented by a fairly large set of scalar values. For example, most valves have only one state (open or closed); for each neutral beam, only the next shot value and last shot value for voltages, currents, and timings need be maintained for control purposes, etc. Secondly, although there is also an apparent parallel between data required for facility evaluation and data from diagnostics acquisition, most of the data required for facility evalation (e.g., temperature sensor readings, valve openings, vacuum levels, etc.) are scalar in nature. On the other hand, neutral-beam conditioning cannot be accomplished without a history for each beam indicating that simple vectors of data are required. In this sense, a parallel does exist (i.e., some of facility evaluation and some of diagnostics acquisition are of a vector nature). Again, diagnostics acquisition is responsible for much of the data in the data base.

Most of the examples given above have concerned the set of data required for current facility operation or for current (and perhaps immediately prior) diagnostic shots. For quick comparison of past and present results, a complete history of MFTF is also necessary; accordingly, the data base must contain a large volume of historical data. When a user is referencing the data base, from either the PLI or the QLI, both types of data may be accessed.

Associated with the concept of archived data is the notion that no acquired data may be lost after

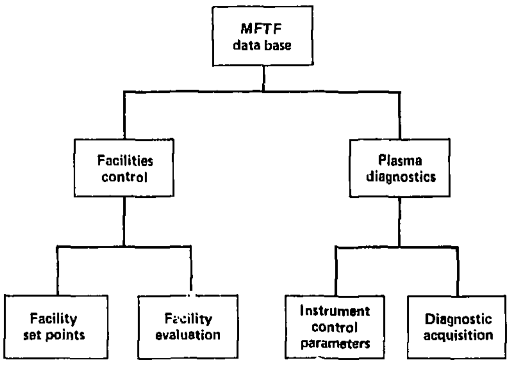

Fig. 3. The MFTF data base.

having been entered. Because certain parts of the data base are more critical than others, we have included a mechanism to request that data be "duplicated" in another physical part of the computer network. Thus, a request to write data into a critical part actually writes them to two distinct physical locations to ensure against loss. In order to establish a complete framework for the MFTF data base we turn next to a discussion of implementation concepts.

Logical View of the MFTF Data Base

The entire data base may be thought of as a set of tables. Each table has a name, as does each of its columns. As an example, consider the startup neutral-beam set-point table in Fig. 4. Its name is given in the upper left corner of the description, and each column has a name, such as beam number (beam no), fire, raw accel voltage (accel_vr), a source accel voltage (accel_va), etc. Each of the columns may be of a scalar, vector, or complex (in the sense of Pascal records) structure. Associated with each table is a set of rows. (For further description of the more theoretical aspects of such a data-base structure, see the discussions on relational database systems[2,9,15]. It is sufficient here to observe that the data base consists of named tables, with rows and named columns, a structure in which each column exists for each row defined in any table.

Along with the inherent implications of table structures, there must also be definitions to define the various physical properties of a table. A table may exist in the central shared memory, in local disk of one (only) of the computers or on the local disk of one (only) of the computers.

An additional concept is that of table ownership. The assumption here is that there is global read access to any table in the MFTF data base, however, table writing is restricted. In addition, since the count of rows per table may vary greatly, a variety of row-access methods is provided to minimize the time required to find a particular row. Associated with row count per table is the concept of memory-contained vs virtual tables. Whereas the startup neutral-beam set-point table is small enough to fit entirely in the local memory of the respective computer (i.e. it is memory-contained), the diagnostics data acquired from a particular diagnostics instrument may be so voluminous that they must be accessed in parts; the entire table is too large to fit in memory at one time (i.e. it is virtual). Given that

"Startup Neutral-Beam Set Point Table"

Purpose: Table primarily contains parameters which are sent to the LCC's.

stu_spt	beam_no	fire	accel_vr	accel_vs	_ _ _	calorimetry
	1					
	2					
	.					
	.					
	.					
	24					

Each column (other than beam number) contains two values: first, the latest value sent to the beam's LCC, and secondly, the next value to be sent.

Description	Size	Domain	Description	Size	Domain
beam_no	1	1..24	filament_duration	2 x 2	Integer
fire	1 x 2	yes, no	gas_duration	2 x 2	Integer
accel_v (raw)	2 x 2	integer	arc_duration	2 x 2	Integer
accel_v (at source)	2 x 2	Integer	accel_duration	2 x 2	Integer
accel_i	2 x 2	Integer	auto_conditioning	1 x 2	on, off
arc_v	2 x 2	Integer	rate	1 x 2	shot, manual
suppressor_v	2 x 2	Integer	accel_io	2 x 2	Integer
filament_v	2 x 2	Integer	accel_vo	2 x 2	Integer
gas_state	1 x 2	on, off	aiming_z	2 x 2	Integer
gas_percent	2 x 2	Integer	beam_dump_flow	2 x 2	Integer
beam_delay	2 x 2	Integer	calorimetry	1 x 2	on, off
filament_gas_delay	2 x 2	Integer			
gas_arc_delay	2 x 2	Integer			
gas_accel_delay	2 x 2	Integer			

Fig. 4. User access -- (user definition)

```
"Startup Neutral-Beam Set Point Table"

.
.
.
CONST  max_stu_nbs  = 24;  "Max number of startup neutral beams."
       max_sus_nbs  = 24;  "Max number of sustaining neutral beams.'
       max_nb_value = 24;  "Max value of neutral beam numbers."
.
.
.
TYPE   gas_type      = RECORD
                         state     :  (on, off);
                         percent   :  0..100;
                         pressure  :  INTEGER;
                         .
                         .
                         .
                         END;

       nb_type =       RECORD
                       beam_no     :  1..max_nb_value;
                       fire        :  (yes, no);
                       accel_vr    :  INTEGER;
                       accel_vs    :  INTEGER;
                       accel_i     :  INTEGER;
                       gas         :  gas_type;
                       .
                       .
                       .
                       accel_ih    :  ARRAY (0..400) OF INTEGER;
                       accel_vh    :  ARRAY (0..100) OF INTEGER;
                       gas_h       :  ARRAY (0..100) OF gas_type;
                       .
                       .
                       .
                       END;
.
.
.
RLN    stu_spt       :  ARRAY (0..max_stu_nbs)  OF nb_type AS ..(rln opts)..;
       sus_spt       :  ARRAY (0..max_sus_nbs)  OF nb_type AS ..(rln opts)..;
.
.

VAR    stu_sp        :  nb_type;
       sus_sp        :  nb_type;
```

Fig. 5. User access -- (Pascal, DBMS definitions).

```
"Startup Neutral-Beam Set Point Table"

1a.    stu_spt(beam_no = J).accel_vr := T;
1b.    T  := stu_spt(J).accel_vr;
2.     stu_spt(accel_vr = T).fire := no;
3a.    stu_sp := stu_spt(J);
3b.    stu_sp.gas.state := stu_spt (gas.pressure = 37).gas.state;
4.     .
       .
       .
       sum := 0;
       vec := LOC OF stu_spt(fire = yes);
       i   := 0;
       WHILE  vec (i)   0 DO BEGIN
              sum := sum + stu_spt(vec  i  ).accel_i;
              i := i + 1
              END;
       .
       .
       .
```

Fig. 6. User access -- (Pascal, DBMS usage).

certain data are more important than other data (i.e. more difficult to reconstruct), duplicate tables may be declared such that whenever the primary table is written into, its duplicate is also updated without intervention from the user.

Data Base Management System

Overview

As noted above, the DBMS is a set of software tools that allow the user to operate upon data in a manner that is "closer" to the way in which the data is thought of. This implies the existence of certain concepts as elements of the DBMS, which are outlined here. First, for the data base to exist, it must have a structural definition, both from the experimenter's point of view, and from the physical computer hardware viewpoint. Secondly, once we are able to define structures, the corresponding space (on disk and in memory) must be allocated, along with necessary supporting information. Finally, given that the data base is ready for access, a set of routines must exist to facilitate reading, writing, etc. This implies throughput requirements, in terms of both speed and volume of data. For example, How long does it take to obtain the accel current for sustaining neutral beam 20, or How long does it take to change the accel current? On the other hand, How does the DBMS respond to being required to store many megabytes of plasma diagnostics data?

In addition to certain minimal throughput requirements, we must be concerned with what users type on the console keyboards in order to access the data base, usually referred to as the user interface. This interface must at least have read, write, and search capabilities, in addition to whatever support is necessary. Again, the accel current for sustaining beam 20 must be obtained for a calculation in a computer program, it must be changed as a result of perhaps user input, or the beam numbers must be obtained for all beams that have an accel current greater than a specified value.

These capabilities must be available from applications programs (i.e. programs concerned with control of MFTF or programs concerned with plasma diagnostics data reduction following a shot) and interactively from terminals where physicists are searching through prior shot data, performing correlation studies, etc. Given that the capability to change data exists and that the data are considered valuable in some sense, there must also be a way of both selectively recording these changes (i.e., who initiated the change, when, and to what new value) and preventing unauthorized changes. (For a more detailed explanation of the MFTF DBMS, see the paper by Choy and Wade in these proceedings.[10])

User Interface

Noting that the data base is referenced both by prepackaged programs and by users from a terminal, the user interface consists of the PLI and the QLI. The PLI is meant for interaction upon the data base from computer programs that must perform fairly extensive analysis; the DBMS is quite simply the read and write mechanism. By contrast, the QLI is a terminal-based interactive program, built using PLI facilities, that provides a user at a terminal the capability to examine and modify the data base. Sufficient computational resources exist within the QLI to permit reasonably simple analysis to be performed without requiring that a full program be written. We assumed that the types of users interacting with the QLI have some—but limited programming experience—usually in conjunction with other disciplines such as plasma physics and various types of engineering; hence, the QLI is oriented as an abbreviated programming language.

Program Level Interface

The PLI consists of two parts; a precompiler, and a runtime subroutine library. Since the computer programs running on SCDS for controlling MFTF are being written using the Pascal programming language[11], the precompiler has as primary input a Pascal source program and produces a resultant source program as output. The primary function of the precompiler is to change occurrences of data base references from a syntax that is more easily understood by the person creating the source program into a syntax intelligible to the Pascal compiler.[12]

An additional aspect of the precompiler function is indicated in Fig. 2: both the source program containing DBMS syntax and a set of DBMS data definitions are supplied to the precompiler. The precompiler then uses the data definitions (which describe the data base) to transform requests to create, open, close, read, and write the data base into procedural references. The result is a source program with DBMS runtime calls that is then compiled by Pascal. The output from the compiler is combined with the DBMS runtime library to create the computer program. Whenever the program is subsequently run, the runtime routines are called when access to the data base is required, and thus perform the requesting operations.

Data Structure Capabilities

In view of the above discussion regarding the PLI, what data structure capabilities are available in the DBMS? First, the data structures available in the Pascal programming language are also available from the PLI. The precompiler parses declarative syntax to an internal form, storing it in a runtime symbol table maintained on disk. Upon the occurrence of an open command to the DBMS (indicating the program is intending to access a table in the data base), the associated declarative structures are read from disk into memory. Figure 4 uses the startup neutral-beam set-point table as an example. The initial work done to establish a new table in the data base is to generate a user definition—essentially how the data looks to the user. Here the user, in conjunction with the Data Base administrator, defines the table name, column names, the data structure of each column (including size, data type, domain, etc.), and other pertinent information.

Thereafter, it is transformed into a syntax acceptable by the precompiler for inclusion in the data base (see Fig. 5). By using Pascal syntax, along with a few minimal extensions, the sizes of various entities are defined with the Pascal CONST construct, and the actual definition of a given row of the table is specified with the Pascal TYPE construct. Note here that the data structure for all rows of a table have the same shape; the TYPE statement is used to establish the structure of any given row of the table. Following all size and structure definitions via CONST and TYPE constructs, the RLN construct is used to declare the table itself. This construct is not part of the Pascal syntax; it is parsed by the precompiler and removed so that the compiler does not see it. Upon recognition of the RLN construct, the runtime symbol table entry is created for later use. In more specific terms, the table name is "bound" to a specific data structure at this time, along with other information about the table, such as its overall size (i.e. the number of rows in the table), where it exists (in memory, on disk, etc.), on what computer within the SCDS network, row access methods to use upon the occurrence of read and write commands to the table, etc. Again, since the Pascal TYPE construct is being used to declare the structure of a typical row, the data structures available in Pascal are available in the data base.

Program Level Interface - Usage Constructs

After the set of tables has been declared within some program, the user may access the tables. This is done by first opening a given table (causing entries to be made in the DBMS runtime symbol table and data to be read from disk), then requesting reads and/or writes to the table. This is accomplished with a special syntax that is not a part of the Pascal syntax. As before, the precompiler parses these constructs, replacing them this time with constructs acceptable to Pascal. Figure 6 shows examples of how reads and writes are performed. In example 1a, the startup neutral-beam set-point table's accel raw voltage is being changed for a specific beam (the beam whose number is contained in the variable J) to the valve contained in variable T. Example 1b causes the data base to be read; accel raw voltage for beam number J is read and stored in the variable T. Example 2 shows a write to the data base (as did example 1a). However, in this case, the column name labelled "fire" receives the value of "no" wherever the accel raw voltage has the value contained in T. Given that the variable stu_sp is bound to the TYPE structure used to define individual rows of the table stu_spt (note the VARs in Figure 5), then example 3a causes the entire row (i.e., all column values) for beam J to be read from the data base and stored in the variable. This is of value when numerous operations with the same row of a table must be performed so that data base accesses are minimized. Example 3b shows a similar data base read, except that a subset of the row is read. There are times where one wishes to access a subset of the rows of a table; example 4 shows the generation of a sum. The first data-base access (the LOC OF construct) returns the locations of all rows in the table whose beams are intended to be fired (where fire = yes). Then, within the WHILE iteration, subsequent data base accesses are made, one for each row (and therefore beam) in the table, generating the sum of all accel currents whose beams are intended to be fired.

Summary

After an introductory discussion of the historical perspective of reasons for generating a new DBMS, what the MFTF data base looks like, and an overview of the Data Base Management System, the user view of the system is discussed. In particular, the Program Level Interface is explained together with its precompiler and associated runtime routine library. Examples of both declarative capability and usage capability are then given to demonstrate the use of the system.

References

1. Clarke, T. O., J. E. Crapuchettes, and P. D. Siemens, MFTF Data Base Management Study for University of California, Lawrence Livermore Laboratory, Livermore, California, September, 1977.

2. Date, C. J., An Introduction to Database Systems, Second Edition, 1976, pp. 3 - 9.

3. Butner, D., "MFTF Supervisory Control and Diagnostics System Hardware," Proceedings Engineering Problems of Fusion Research (IEEE), 1979.

4. Speckert, G. C., "The Man Machine Interface for MFTF," Proceedings Engineering Problems of Fusion Research (IEEE), 1979.

5. McGoldrick, P. R., "SCDS Distributed System," Proceedings Engineering Problems of Fusion Research (IEEE), 1979.

6. Wyman, R. H., "Results of Studies Performed on the Model of the MFTF Supervisory Controls and Diagnostics System (SCDS)," Proceedings Engineering Problems of Fusion Research (IEEE), 1979.

7. Ng, W. C., "An Overview of MFTF Computer Control and Diagnostics System," Proceedings Engineering Problems of Fusion Research (IEEE), 1979.

8. Coffield, F. E. and G. E. Davis, "MFTF Plasma Diagnostics Data Acquisition System," Proceedings Engineering Problems of Fusion Research (IEEE), 1979.

9. Codd, E. F., A Relational Model of Data for Large Shared Data Banks, CACM 13, June 1970, pp 377-387.

10. Choy, J. H. and J. A. Wade, "A Data Base Management System for the MFTF," Proceedings Engineering Problems of Fusion Research (IEEE), 1979.

11. Jensen, Kathleen and Wirth, Niklaus, PASCAL User Manual and Report, Second Edition, Springer-Verlay, New York, 1974.

12. Young, Robert, PASCAL/32 Language Definition, Department of Computer Science, Kansas State University, 1978.

13. Lindquist, W. B., R. Eckard, T. Holdsworth, L. Mooney, D. Moyer, R. Peterson, D. Shimer, R. Wyman, and H. Van Ness, "Overview of the MFTF Electrical Systems," Proceedings Engineering Problems of Fusion Research (IEEE), 1979.

14. Choy, Joseph H., MFTF Total Benchmark, June, 1979, LLL, Livermore, California, UCID 18209.

15. Kim, Won, Relational Database Systems, Computing Surveys, Vol. 11, No. 3, September, 1979.