SRC TR 87-54

Database and Process Control for RAMCAD

by

B.T. Sawyer and M. Pecht

DATABASE AND PROCESS CONTROL FOR RAMCAD

B.T. SAWYER and MICHAEL PECHT

After three years of development, the University of Maryland Reliability and Maintainability Computer Aided Design (RAMCAD) expert system for electronics has been implemented for on-site test and utilization by Westinghouse Defense Electronics in the design of the electronics for the Airborne Self Protection Jammer (ASPJ). This paper focuses on the database and system controller.

INTRODUCTION: DATABASE MANAGEMENT FOR ELECTRONICS DESIGN

It is commonly known that the thermal, mechanical and electrical variables of a printed circuit board do not lend themselves to conventional data management methods. It has been suggested that commercially available relational models are most suitable for handling electronic design data. RAMCAD has been developed with this in mind. Our requirements are that the system provide accurate, and concise information in a context that helps a designer make intelligent decisions.

The RAMCAD system uses exact analysis of the data based on computational methods which have been tested against measurements of real designs. Although military standards allow for approximations in the early stages of design we have taken pains to make the analytical results as close to the actual situation as possible. The benefit of such analyses to the designer is considerable. Potential flaws in a design appear early enough to allow for correction without undue loss of time or effort. Since decisions are based on the available data we feel that the best data helps one make the best decisions.

THE RAMCAD DATABASE

A practical solution to the database management problem presented by electronics design is to localize the data for a particular design and to limit the number of processes that have direct access to the local database. In the RAMCAD system each design (PWB) has its own database which is built from the global database of all available electronic parts. Secondly, translator modules mediate between the local database and the analysis and display modules. Restricting the number of programs that may access data allows better control and permits more flexibility in the number and type of modules that the system can support.

The local database consists of a Pascal file of records, one for each component, files of sorted keys for rapid access, a record for PWB parameters, and a number of ASCII files which contain layout data. A component record is of modest size (400 bytes) and contains all of the MIL-217E constants required to calculate reliability based on thermal analysis. The record also contains geometric data for placement, temperature and stress data, and coded character

fields which are linked to ASCII files of descriptors and labels. The PWB record contains data on failure rate, stress, temperatures, and analysis parameters.

The analysis and data reporting modules are independent programs developed by a number of programmers. Once the type and format of data input/output is established, a translator is created to supply data and, if necessary, another is made to pick up the results. These translators insulate the programmer from the task of reading the database and provide control channels for data flow. The model permits the system to adapt to changes in data structures and sources without disturbing modules that are operating on or displaying the data.

SCHEDULING AND CONTROL OF THE DESIGN

The RAMCAD executive controller integrates the system by presenting menus of options and executing requests. It permits open-ended choices of functions using a consistent and easily learned user interface. It verifies the existence of the necessary data and program files required to carry out a request and then initiates the sequence of program modules which perform the desired function.

The user interface consists of a series of menus which are defined as program constants. Each menu displays a set of tools that are available for a specified task. Once a tool is selected, the system takes care of the details. The intent is to have a "seamless" system where tasks appear to proceed as a single action rather than a series of independent program modules. Furthermore, the data structures used in the RAMCAD controller are flexible enough to allow us to add or delete modules, change execution sequences, and rearrange menu choices without major code revision.

The flexibility of the system comes from the method we use to verify files and schedule program modules. These functions are carried out using identifiers whose values are defined in the program declaration section. Each program and data file identifier is mapped to an actual file name and then grouped into sets which may be enlarged, changed or rearranged without affecting the code. The structure and operation of the controller is best explained by an illustration. The coplanar thermal analysis function will serve as our example.

```
menu123 : menu_rec =
    ( entries : 4;
      choicestr : 'CISE';
      title : 'THERMAL/RELIABILITY ANALYSES';
      labels : ('Coplane', 'Inline',
                'Standard Electronic Module',
                'EXIT to the Previous Menu',
                '', '', '', ''));
```

The statement, ch := Menu_Choice(menu123) performs the operation of menu display and selection, and returns the selected value to the character variable, ch. The resulting value of ch is passed through a CASE statement which determines the action to be carried out. The 'C' and 'E' options of this statement are:

```
CASE ch OF
  'C':BEGIN                          {Coplanar thermal analysis}
        pnode := Check_pgm_list(coplane_pgm_s);
        enode := Check_ext_list(bd_name,coplane_ext_s);
        dnode := Check_datfi_list(coplane_dat_s);
        Mark(node);
        node^.next  := Link_nodes(pnode,dnode,enode);
        IF node^.next = nil THEN BEGIN
          Schedule(coplane_pgm_s);
          complete := true;                END
        ELSE BEGIN
          Show_missing_list(node^.next,choice_name);
          complete := false;
        END;
        Release(node)
      END; {'C'}
  'E': BEGIN                                {Exit menu123}
        act_menu := 'menu12';
        complete := false
      END; {'E'}
 END; {case}
```

This illustrates how the controller handles menu selections. If a choice requires action, the presence of the appropriate files is checked. If they exist, a sequence of modules is scheduled. Otherwise the missing file names are shown and the menu returns a value of 'false' for the variable 'complete', which informs the menu control procedure that the choice can not be carried out and the active menu (act_menu) should be displayed.

The method of changing menus is illustrated by the exit ('E') option. If the user chooses to exit this menu (menu123) the value of the next menu is set (menu12) and 'complete' is set to false. This causes the menu controller to display the next menu (menu12) and await further action. The system control program only exits when the value of 'complete' is 'true.'

Verification and scheduling are accomplished by passing predefined values to Check and Schedule. The data structures for these values are illustrated below.

```
{identifiers for executable programs}
pgm_id_type  = (pgmF,ldb2ana, flowrow, coplane, tem2ldb,
                  reliable, pgmL);
pgm_id_array = Array [pgm_id_type] of str20;
pgm_set      = Set of pgm_id_type;
```

```
{identifiers for datafile extensions}
ext_type     = (extF,ldb,uky,boa,pwb,lay,extL);
ext_array    = Array [ext_type] of str4;
ext_set = Set of ext_type;

datfi_type   = {similar to the above declarations}
```

The assignment of a file name to each of the constant identifiers is done by means of the following program constant declarations.

```
program_map : pgm_id_array =
  ('', 'LDB2ANA.COM', 'FLOWROW.COM', 'COPLANE.COM',
       'TEM2LDB.COM', 'RELIABLE.COM',    ''  );

extension_map : ext_array =
  ('', '.LDB', '.UKY', '.BOA', '.PWB', '.LAY', ''  );

datafile_map :  datfi_array =  as above
```

The program, extension and datafile identifiers are used to build sets of values for each of the sequences in the system. Each set is defined as a program constant and given a name that identifes the function it carries out. The constant sets for the thermal analysis coplane program are:

```
coplane_pgm_s : pgm_set = [ldb2ana,flowrow,coplane,tem2ldb,reliable];

coplane_ext_s : ext_set = [ldb,uky,boa,pwb,lay];

coplane_dat_s : datfi_set = [];
```

The main program's Check_xxx_list functions scan the sets of values to see if the corresponding files exist. If not, a pointer to the missing name(s) is returned. If all files exist, coplane_pgm_s is sent to the Schedule procedure which writes the names of the files to a text file. As soon as the control program ends, a background program executes the programs in the text file and finally runs the controller to complete the cycle.

The flexibility of this programming method is apparent, in that any series of programs can be executed by grouping identifiers into a set. A new module is added by creating an identifier and a file name, and putting them in the appropriate declarations. A module is removed by taking its name out of the list.

ACKNOWLEDGEMENTS

This effort has been supported in part by the Institute for Defense Analyses, Westinghouse Defense Electronics, and the University of Maryland Systems Research Center under grant No. CDR8500108 from the National Science Foundation.