INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9020905

The integration of expert systems into energy management system centers using a dispatcher training simulator

Chen, Ming, Ph.D.

University of Washington, 1989

Copyright ©1989 by Chen, Ming. All rights reserved.

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

The Integration of Expert Systems into Energy Management System Centers Using a Dispatcher Training Simulator

By

Ming Chen

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Washington

1989

Approved by Mark J Damborg
(Chairperson of the Supervisory Committee)

Program Authorized
to Offer Degree Electrical Engineering

Date 11 December 1989

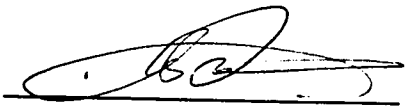
© Copyright by

Ming Chen

1989

Doctoral Dissertation

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 300 North Zeeb Road, Ann Arbor, Michigan 48106, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature 

Date 11 December 1989

UNIVERSITY OF WASHINGTON

ABSTRACT

The Integration of Expert Systems into Energy Management System Centers Using a Dispatcher Training Simulator

By

Ming Chen

Chairperson of the Supervisory Committee: Professor Mark J. Damborg
Department of Electrical Engineering

Expert systems (ES) are receiving considerable attention as potential tools to aid in the operation of power systems. Naturally, the expert system must be integrated into the power system's EMS center to play a role in system operations. Generally, there are two approaches to this integration: the appended and the embedded. The appended approach involves separate computers, separate displays and a datalink to transfer data between the expert system computer and the EMS computer. The embedded approach involves direct integration of the expert system into the EMS computers and displays.

The embedded system is expected to be more desirable. First the dispatcher can interact with the expert system using the standard EMS interface. Second, the performance of the expert system should be better because of the "tighter" coupling between the expert system and the rest of EMS software and database. However, the performance of such a system is difficult to predict and experiments on an actual system are almost impossible because of the likelihood of disrupting normal operation. Consequently, it is desirable to have an environment for developing an embedded system "off line". A Dispatcher Training Simulator (DTS) provides such an environment if it could simulate the system behavior of interest.

In this thesis, the issues involving the embedded approach are studied and the basic results are presented. Two versions of the embedded approach are investigated: the database integration approach and the full integration approach. The ESCA DTS is used to provide a realistic EMS center environment. Two expert systems, CRAFT and VCES, are used as examples to illustrate the integration approaches. The CRAFT-DTS system is used to illustrate the database integration approach and the VCES-DTS is used to study the full integration approach.

The research demonstrated that an expert system can be integrated into an EMS using the embedded approach. The database integration approach verified that all processes could exist on a single computer and would function properly.

However, since the ES and DTS were separate processes under the control of the operating system, their interaction was somewhat awkward and difficult to manage. These control problems were solved by the full integration approach in which the expert system was installed within the DTS and controlled simply as another DTS function. This approach has the additional benefit of using the standard DTS man-machine interface for the expert system. Hence, the expert system is also easier to use by the operators because the user interface is familiar.

An important result of this study is that it provides a prototype for enhancing EMS center functionality by other expert systems or non-procedural programs. That is, the interface problems that were solved in this integration work are directly applicable to the integration of other programs written in the ES language. We expect that they will provide guidelines when working with programs in other languages. The enhancement of EMS functionality with expert systems and other non-procedural programs is expected to be very important in future EMS centers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This research was sponsored principally by the Electric Power Research Institute, Palo Alto, CA, the ESCA Corporation, Bellevue, WA, the Puget Sound Power and Light Company, Bellevue, WA and the National Science Foundation. The author wishes to thank these organizations. Additionally, the author expresses sincere appreciation to the members of the examination committee, Professors M. J. Damborg, C. C. Liu, A. Holden, P. O. Lauritzen and R. Palmer. Particular gratitude is due to Prof. Damborg for his invaluable guidance and great efforts which have improved the quality of this thesis immeasurably. The author also expresses his gratitude to ESCA people: T. M. Athay, W. W. Owen and Davis Hwang for their assistance and cooperation.

Finally, he wishes to thank his host family Judy and Professor V. Schomaker, for their valuable encouragement and friendship.

CHAPTER 1

INTRODUCTION

The application of expert systems (ES) to power systems has received great attention in recent years. A number of ESs have been developed or are under development in various areas [1-10]. Yet little has been reported on the actual implementation of an ES in an energy management system (EMS) which is the subject of this dissertation. A dispatcher training simulator is used to provide a realistic environment for the integration work. The purpose of the study is to develop a way of "tightly" coupling an ES with the EMS database, software and man-machine interface. Such an integrated ES can take advantage of the full power of the EMS and enhance the EMS functionality. Two versions of the integration have been studied: the database integration and the full integration. This work complements our work to install an expert system CRAFT in the operating center of Puget Sound Power and Light Company using an appended approach [6].

1.1 CHALLENGING PROBLEMS IN ENERGY MANAGEMENT SYSTEM

Power systems are examples of systems which are large, complex, non-linear and dynamic. The power system control center, which is also called an energy management system (EMS) center, is usually equipped with computers to perform the supervisory control and data acquisition (SCADA) functions. The SCADA computers collect system data and perform routine or tedious computational work. They also maintain the system database, provide information to the operators via the man-machine interface, and carry out the control commands. The system operator or dispatcher is the supervisor. The basic task is to maintain a match between generation and load, ensuring that equipment operates economically and within allowable bounds.

As power systems grow ever more complex, control center operators and dispatchers face unprecedented challenges. In addition to handling normal operating problems, system operators and dispatchers must be able to deal effectively with power system contingencies caused by severe weather conditions, equipment failures, human errors or other abnormal conditions. Managing and responding to the huge flow of information

into power system control centers during abnormal conditions is becoming an increasingly intricate task. Expert system technology, incorporating expert knowledge with the power of the computer, is a possible answer to the challenges.

1.2 OVERVIEW OF EXPERT SYSTEMS APPLICATIONS TO EMS

Expert system (ES) technology is a branch of artificial intelligence [11]. An ES is also called a 'knowledge-based' system. A properly designed expert system can incorporate the knowledge of experienced engineers into a computer program and automate the expert's reasoning process thereby increasing the efficiency of a problem solving process. The main feature of expert system technology is the separation of the knowledge base from the inference mechanism. Thus, it is possible for the knowledge base to grow as more knowledge is acquired without disturbing the expert system's operation. When rules are used to represent the knowledge, an ES is called a 'rule-based' system.

The basic idea of expert systems is to take advantage of the high speed and big memory of computers for decision-making and problem solving. Three major motivations can be summarized for the expert system application: knowledge storage, automation and a new approach to problem solving. First, the ES provides a reservoir of experiences of human engineers. Second, the ES helps to automate decision-making processes that are performed by human engineers. The ES also provides a new problem solving technique that complements the conventional algorithmic methods.

Early applications of expert systems were mainly in medical diagnosis and therapy [12], natural language understanding [13,14], computer system configuration and troubleshooting [15].

For the operation of power systems, a number of ES applications such as fault diagnosis [16-20], alarm processing [2,51], restoration and remedial control [16,22-24,40] and unit commitment [21] have been developed. Most of the expert systems developed are experimental and off-line. The difficulty of actual implementation comes from the security concerns of the power system and the technical issues. The power system application requires that an ES be well studied, tested and mature before actual implementation. It is not allowed to perform on-line tests that would endanger the

system normal operation. In addition, the efficient tools for ES development may not be available in EMS centers [6,7].

The University of Washington has developed a number of expert systems for power system operational aids [6]. The implementation of expert systems has been studied [16,25,26]. One of the expert systems developed has been successfully installed into a control center using the appended approach [25].

There are two natural approaches to implementing an expert system: the "appended" and the "embedded" approaches. In some literature [10], they are called the "distributed" and the "monolithic" approaches. The appended approach involves a separate computer, separate displays and a datalink for transmitting data between the EMS and the ES computers. The embedded approach involves integrating an ES directly into the EMS system computers and displays.

The embedded system is expected to be more desirable. First, the dispatcher can interact with the expert system using the standard EMS interface, thus the dispatchers may feel "natural" when they use the EMS system with an additional expert system tool. Second, the performance (response time) of the ES should be improved because of the "tighter" coupling between the expert system and the rest of the EMS software and database. Third, the implementation may be easier because it is not necessary to build a datalink between the two computers. However, the performance of such a system is difficult to predict and experiments on an actual system are almost impossible because of the likelihood of disrupting normal operation. Consequently, it is desired to have an environment for developing an embedded system "off line". A Dispatcher Training Simulator (DTS) provides such an environment if it can simulate the system behavior of interest. Of course, in addition to providing a means for evaluating the concept, such a system would provide a tool for training dispatchers in the use of the ES.

1.3 SCOPE OF THIS WORK

This dissertation presents new work in the embedded approach to implementation: the integration of expert systems into EMS center software using a dispatcher training simulator (DTS). Two specific approaches of the integration have been developed. One is the database integration approach, the other is the full integration approach. The

ESCA DTS [35] and two expert systems, CRAFT and VCES, are used to instantiate the two approaches respectively.

There are four basic questions to be answered in the study:

(1) Can an expert system that is built using an AI language tool be integrated into an EMS center which uses standard, procedure based software for program and data management? How?

(2) Will the integrated expert system degrade the overall EMS center performance to an unsatisfactory level?

(3) Does the full integration yield the expected benefits?

(4) How can a generic expert system be integrated into an EMS environment and what are the requirements for the EMS hardware and software?

Our successful work answers these questions. Following this introduction, Chapter 2 reviews two expert systems, VCES and CRAFT, and the appended approach, all due to previous work. Chapters 3 and 4 report the basic results of this dissertation with Chapter 3 presenting the database integration approach. The full integration approach is presented in Chapter 4. As a conclusion, Chapter 5 gives a comparison and evaluation of the three approaches. Future implementations of ES in EMS centers and the requirements for the EMS host system are discussed in Chapter 6.

CHAPTER 2

REVIEW OF TWO EXPERT SYSTEMS FOR POWER SYSTEM OPERATION

Two expert systems have been developed at the University of Washington to assist power system dispatchers during abnormal operations. One is for power system voltage control (VCES) [8] and the other is for transmission line fault isolation and customer restoration (CRAFT)[17]. Since they are later used to illustrate the integration work, this chapter gives a functional review of the basic structure and features of these expert systems. This review is followed by a discussion of the "appended" approach to integrate an expert system into an EMS center.

2.1 EXPERT SYSTEM VCES

A practical requirement in power system operation is to maintain bus voltages within certain upper and lower limits. Abnormal voltages may lead to equipment damage and possibly to blackouts or brownouts in the system. As the power system loads have grown in recent years, many power systems operate closer to their limits with less reserve margin available. Hence, they are more vulnerable to any unexpected load variations and network contingencies which may cause bus voltage or line overload problems.

The VCES is a rule-based system that uses a sensitivity model and heuristic rules to develop remedial actions to low voltage problems in power systems, providing the voltages are only a moderate amount too low and have not approached "collapse." The principle of VCES, its development and performance evaluation are the subject of previous papers [8,23]. Since it is used to illustrate the full integration approach, this section reviews some background and salient features of the VCES.

Essentially, the VCES is based on the sensitivity model of the power system. The power flow equations (PFE) describe the power system behavior in the steady state. Given the power system topology, component parameters, power generation and demand, the PFE can be solved to obtain the system state, i.e. the bus voltages and angles. The sensitivity information is used in the VCES to select controls and to estimate the control effects.

Heuristic rules also exist for the voltage control problem. An experienced power system dispatcher can usually find appropriate solutions to cope with voltage limit violations. In dealing with a voltage problem, a dispatcher usually uses three types of controls: switchable capacitor banks, transformer tap changers, and the generator excitations. The existence of empirical knowledge suggests that an expert system approach may be useful. The readers are referred to [8] for the empirical rules which were implemented in the VCES.

Since the VCES was developed as an on-line operational aid, its performance is important. A systematic approach was developed for the performance evaluation of the VCES and to identify the computation bottlenecks of rule-base execution [17,28]. Section 2.1.4 reviews this issue.

The first version of the VCES rule base was implemented in OPS5, a LISP based Official Production System. The newer versions were implemented in the C-based OPS83 for faster execution and easy interface with other programming languages.

2.1.1 STRUCTURE OF VCES

As with any expert system, the VCES consists of three major parts: a knowledge base (KB), an inference engine (IE) and a working memory (WM). Figure 2-1 shows the structure. The separation of the domain knowledge from the inference procedure and data is an important feature of the expert system structure. This arrangement allows incremental improvements within the knowledge base thereby easing development. The VCES knowledge base contains a set of rules. Each rule represents a piece of knowledge. The inference engine contains a general control strategy that decides which rule will be executed next. The working memory is a database for the ES. An inference engine can be reused by other expert systems. Like many other ES language tools, OPS83 provides an inference engine. Hence, the ES developer can focus on designing and building the rule base and the working memory.

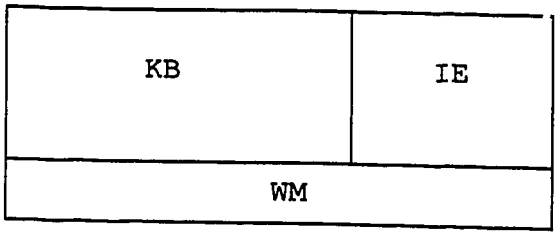


Figure 2-1 The Structure of the VCES

2.1.1.1 THE DATA STRUCTURE OF VCES

The basic data types supported by OPS83 are Integer, Real, Logical, Character and Symbol. Combinations of the basic types, such as Array, Record and Element, can be constructed to represent complex objects.

Symbolic data is an alpha-numeric string with a variable length. This flexible data type differs from the fixed-length character string type in FORTRAN or Pascal. The structure of the "Element" is similar to the "Record" which is described by a number of attributes. Each attribute has its own data type. The major difference between Element and Record is that only the element data type can occur explicitly in working memory while the record type can not. As a result, rules are matched to working memory elements while functions or procedures can operate on records as well.

The following examples show how the working memory elements 'bus', 'reactive_controller' and 'voltage_violator' are defined:

```
type bus = element
(    name : integer;          -- bus number
     type : symbol;           -- bus type, pq or pv bus?
     voltage : real;          -- bus voltage
     v_upper : real;          -- upper limit of the bus voltage
     v_lower : real;          -- lower limit of the bus voltage
     status : symbol;         -- energized or not
     last_rc : integer;       -- the reactive controller last used
     q_demand : real;         -- reactive power demand at the bus
);
```

```
type reactive_controller = element
(    id : integer;                -- id number of the control
     class : symbol;              -- Cap., Tap or Generator
     setting : real;              -- setting value
     steps : real;                -- range of change
     upper : real;                -- upper limit of the setting
     lower : real;                -- lower limit of the setting
     status : symbol;             -- energized or not
     location : integer;          -- at which bus the controller is
);

type voltage_violator = element
(    violator : integer;          -- bus id of the problem bus
     type : symbol;               -- high or low voltage violation
     controller : integer;        -- the controller id
     sensitivity : real;          -- sensitivity of the violator bus
                                  -- to the controller
     delv : real;                 -- deviation from the target volt.
     delv_max : real;             -- maximum available voltage
                                  -- adjustment this control can make
     priority : integer;          -- priority of the controller
);
```

2.1.1.2 RULES IN VCES

The rule base contains the knowledge extracted from the dispatchers' operational experience, from computational analysis and from simulation.

Each rule in the rule base has the form of "IF ...THEN ...". The "IF" part , or the left-hand side, states the conditions. The right-hand side, or the "THEN" part, states the actions. Both sides operate on working memory elements.

The following rule is used to classify a voltage problem. The English version of the rule is given first, then the OPS83 implementation of the rule.

Rule 1 : **IF**

the task is to classify the problem AND
a load bus voltage Vi is below its normal
voltage limit AND
no voltage problem has been detected for this bus

THEN

identify bus i as a low voltage bus AND
compute the voltage deviation: 1.0 (p.u.) - Vi

OPS83 version of the above rule:

```
rule classify1
{  &task     ( task id=classify; );
   &bus      ( bus  type=pq;  voltage < @.v_lower; );
             ~( voltage_violator  violator=&bus.name; );
   -->
   make      ( voltage_violator  delv=1.0-&bus.voltage;
                                 type=low;  violator=&bus.name; );
};
```

Rules are organized into "tasks" in the rule base. Each task consists of a number of rules. This organization facilitates the development of the rule base and improves the efficiency and transparency of the rule base. Following is a list of major tasks in VCES:

Classify_voltage_problem
Select_controls
Prioritize_controls_for_problem_bus
Implement_controls_suggested_by_empirical_rules
Estimate_control_effects

During the execution of the VCES, only one task is active at a time. The activation of tasks is chained by making and removing working memory elements. The first task

classify_voltage_problem is activated when a voltage violation is detected. Upon completion, each task schedules the next task by making a working memory element and at the same time deactivates itself by removing a working memory element. For example, the following rule deactivates the task classify_voltage_problem and activates a new task to select_controllers.

```
rule classify10
{
 &task (task id = classify_voltage_problem;);
-->
 remove &task;
 make ( task id = select_controllers;);
};
```

2.1.1.3 CONTROL MECHANISM OF INFERENCE ENGINE

The execution of the VCES rule-base is controlled by the inference engine. The control mechanism of the engine is data-driven. Based on the data pattern in the working memory, the inference engine determines which rule should fire next. The rule-firing chain forms a decision process. Each such decision process involves three steps: Match-Select-Act. Figure 2-2 illustrates the concept.

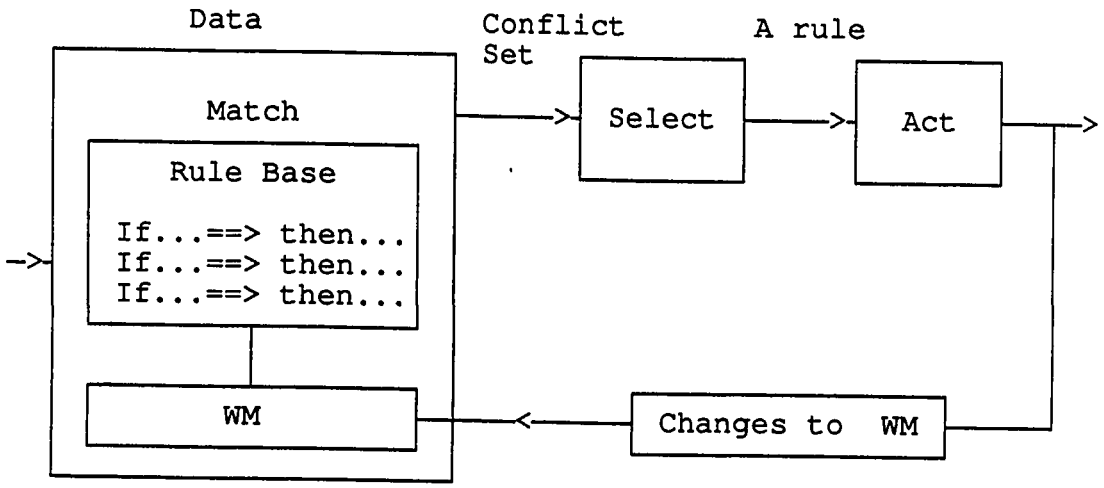


Figure 2-2 Match-Select-Act cycle

First, based on the WM data, the OPS83 inference engine performs pattern matching between the WM data pattern and the RB rules using a fast pattern matching algorithm [28]. The output of the matching operation is a set of rules that are satisfied by the data in the WM. The set of rules forms a "conflict set".

The most applicable rule is then selected from the conflict set by the inference engine based on the specified conflict resolution strategy. The criteria commonly used for conflict resolution are when a rule appears in the conflict set (the rule that matches the most recently-added or most recently-changed working memory element may be executed first), the specificity of the rule (a rule with more attributes in the IF part is a more specific rule), the ranking assigned to the rule by the programmer.

The selected rule is then fired, i.e. the action part of the rule is executed.

Normally the actions will change the data in the working memory and the "Match-Select-Act" cycle will repeat. This process continues until either no more rules are matched to the data in the WM or an explicit halt is encountered.

2.1.2 BASIC OPERATION OF VCES

The logical flow chart in Figure 2-3 shows that the basic operation of the VCES begins with retrieving the system state to determine the existence of a voltage problem. If no voltage problem exists, then it is done. If a voltage problem exists, then the VCES classifies the voltage problem and determines the severity level.

Next, the most suitable controls for correcting the problem are identified and checked for availability. The basic approach is to use sensitivity analysis, calculated from a power flow, to determine which control will have the most corrective effect on each of the problem voltages. The available controls are then ordered according to decreasing impact.

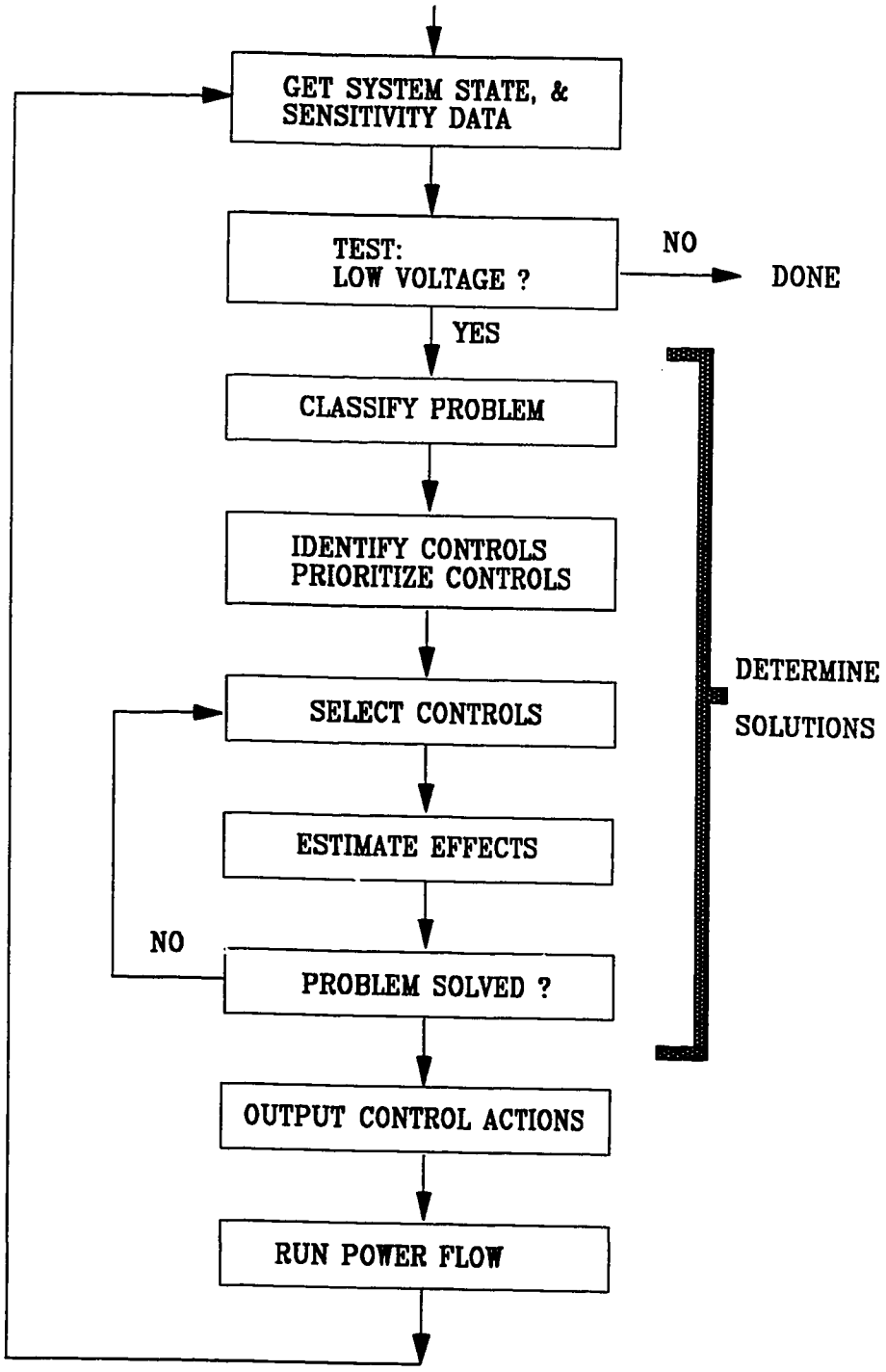


Figure 2-3 Basic operation of the VCES

The final selection of the controls is done in a loop of "selection-estimation". The first control on the control list is examined to determine if it is enough to return the voltage to the target level. If it is enough, then the effects of the control on all problem buses are estimated to see if all voltages are within the limits. If the first control is not enough, then it is set to its limit and the next control on the list is examined. This process is repeated until a collection of the controls is identified which is estimated to solve the complete voltage profile problem. Those controls are issued and recorded in the database.

An iterative solution may be necessary since the proposed solution is an approximation. Remember that a sensitivity model is used and that it is based on a linearized analysis of the power system. Voltage sag problems, on the other hand, are very nonlinear phenomena.

2.1.3 STAND-ALONE VCES TEST SYSTEM

The stand-alone VCES test system in Figure 2-4 was used for development, study and performance evaluation of the VCES [23]. The system contains five major parts: A system database, the VCES, a power flow computation (PF), a sensitivity computation (SNS) and a reactive power dispatch linear programming (LP) routine [33]. The solid lines in the diagram between the blocks stand for the data paths. The dotted lines stand for the control paths issued from the VCES.

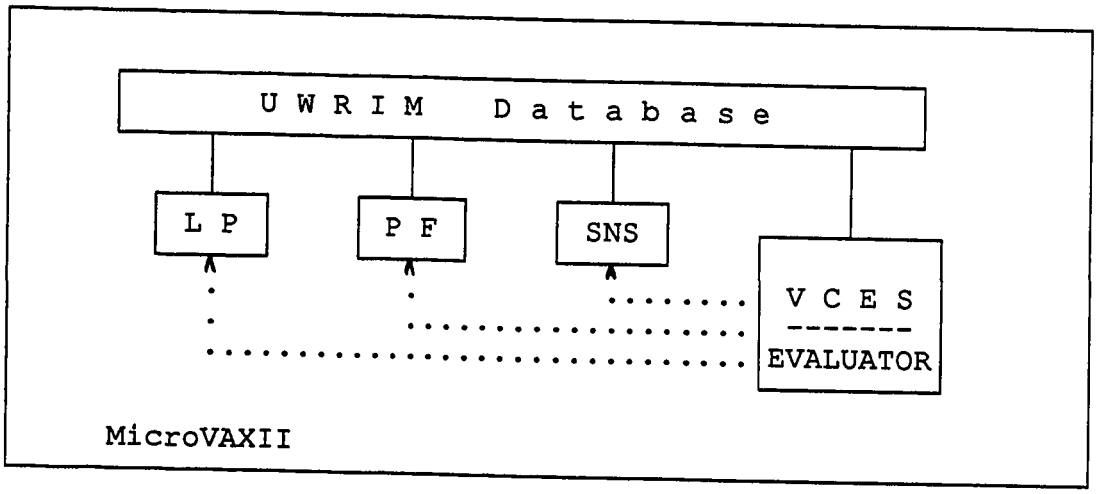


Figure 2-4 Stand-alone VCES test system

Power system control centers usually have a customized database system. To address the issue of expert system implementation, the relational database is chosen due to advantages of data independence of application programs, community view of common data, simple data access by application programs and easy maintenance [29,30,44]. Specifically, the relational database used is UWRIM (University of Washington Relational Information Management). The IEEE 30-bus test system [41] is modeled in the UWRIM database which also contains the sensitivity information. The power flow subroutine is called to simulate the power system behavior and to verify the control actions proposed by the VCES. The LP subroutine is used to obtain an alternative solution to compare to the VCES results. The results are evaluated by a rule based evaluator.

In this system, the SNS, PF and LP are written in FORTRAN. UWRIM is also FORTRAN based. The VCES with the evaluator is written in OPS83. The capability of OPS83 to call routines written in other languages greatly facilitates the development of this system.

2.1.4 PERFORMANCE EVALUATION OF VCES

Performance is important for on-line applications. The performance of an ES can be evaluated in terms of its solution quality, its accuracy and its speed. The systematic approach used in the performance evaluation of the VCES can be applied to other expert systems.

Extensive testing, using the IEEE 30-bus system, was conducted to establish the capability of the ES. The expert system was made to compete against a conventional solution approach, a linear programming reactive dispatch program [33]. The performance of the two approaches was compared using the criteria of number of cases solved, number of iterations for each solution, number of control devices used, number of unrealistic controls and the solution time spent [31,32].

A wide range of scenarios was created and the power flow was run to simulate the system behavior in response to the contingencies. When the VCES reached a solution, the power flow was called again to verify the effects of the control actions suggested by the

VCES. Then for the same scenario, the LP was called to give an alternate solution which was compared with the VCES solution using a rule based "evaluator".

The VCES has been found to work very well for all test problems on the IEEE 30-bus system. It has worked well in the sense that it has quickly converged to a solution which is sensible. It also has worked well when compared to a linear programming solution [31,32]. For cases where the voltage problems were severe, the VCES solution was achieved with less computation time, with fewer control actions and with fewer iterations than the LP solution.

Furthermore, the computational efficiency was analyzed by breaking down the processing cost to several basic tasks. By measuring and evaluating the time spent for these basic tasks the computational bottlenecks were identified. The "match-select-act" cycle was broken down into 5 basic tasks: attribute testing (AT), inter-conditional testing (IT), conflict resolution (CR), working memory change (WMC) and rule firing overhead (RF).

This breakdown helps to identify bottlenecks of the ES solution process and helps to improve the performance of VCES. For detailed information about the subjects discussed here refer to [31,32].

2.1.5 DISCUSSION

The VCES represents a new problem solving technique which combines the rule based approach with mathematical analysis to solve power system problems. Many potential expert systems developed for solving problems such as security assessment, transaction evaluation and unit commitment share the same features and can be approached in a similar manner.

The VCES is applicable when the system operates at steady state and the voltage violation is not very severe. A linearization around the operating point can be performed and the sensitivity analysis can provide a good approximation for the selection of the controls and evaluation of the effects.

The VCES rule base can not solve severe voltage problems, such as voltage collapse. In such cases, the sensitivity of a problem bus to the reactive controls cannot be accurately calculated and may even be estimated with the wrong sign. Then closing a capacitor bank or raising a tap position may result in further drop of the problem voltage.

Except for the three types of controls used in the VCES, other controls such as load shedding and topology changes may be considered in power system practice. The VCES rule base can be enhanced if additional heuristics are available for such cases.

2.2 EXPERT SYSTEM CRAFT

The system called CRAFT (Customer Restoration And Fault Testing) helps dispatchers perform on-line analysis to locate faults causing customer outages. It is only one of a number of possible expert systems which could be developed to aid dispatchers in managing network emergencies. The principle of CRAFT, its development, test and performance evaluation have been reported in the literature [16,17]. CRAFT has been successfully installed and operated in an EMS center using the "appended approach" [25]. A more direct integration of CRAFT into EMS centers has been studied. The results will be reported in Chapter 3. To better understand the new work, this section presents as background the salient features of CRAFT. The appended approach that implements CRAFT into a control center is discussed in section 2.3.

Faults may occur in transmission lines due to many causes such as stormy weather conditions and traffic accidents. Typically each line is equipped with power circuit breakers on both ends which protect equipment from damage by tripping within cycles after a fault occurs. In a subtransmission system, such as the Puget Power 115 kv transmission system, many power lines have distribution substation taps serving loads midway along the line. Numerous sectionalizing switches are also installed on such lines which permit faulted segments to be isolated. Some of these switches operate automatically in the event of faults, perhaps with a time delay. For example, one type of switch opens a fixed number of seconds after a tap point loses its supply. Other sectionalizing switches must be manipulated manually, either from the control center or by field personnel. Figure 2-5 shows a simple transmission line in the Puget Power system. It has two stations, two circuit breakers, two switches (one manual, one

automatic), two line segments and a load tap transformer. Figure 2-6 shows a more complex transmission line in the Puget Power system, one with many taps and switches.

In case of a fault, the standard practice is for the dispatchers to determine what line is faulted and which load taps are in service from the alarm information provided by the EMS system. The goal is to determine where the fault can be located to have generated the observed alarms and to have left the observed pattern of switch settings. Dispatchers refer to diagrams of the faulted lines and perform a reasoning process based on the design principles of the protective devices (circuit breakers, automatic switches) and practical experience. If it is not possible to locate the exact faulted section by reasoning alone, dispatchers may perform tests (opening switches and/or closing circuit breakers). The test actions are selected so as to restore the maximal number of customer substations if the actions are successful. However, if a test should fail, the EMS system will acquire more information about the fault and dispatchers can then continue the diagnosis. This reasoning process is time consuming and contains many opportunities for error. Hence, an expert system aid to speed this process and increase its reliability was judged useful by the Puget Power dispatchers.

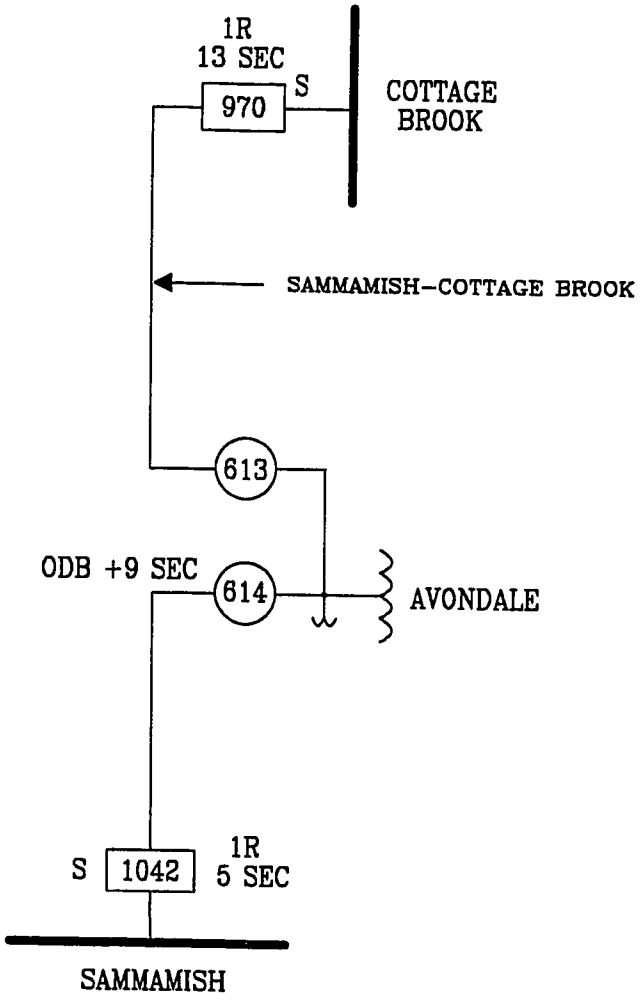


Figure 2–5 A Simple transmission line in the Puget Power system

Legend

1042	Circuit Breaker	S	Supervised
613 946	Switch	1R 5 SEC	One reclosure and its time setting
	Sensor	ODB +9 SEC	Open On Dead Bus Automatic switch time setting
	Under–sea cable		
NO	Normal Open		Substation

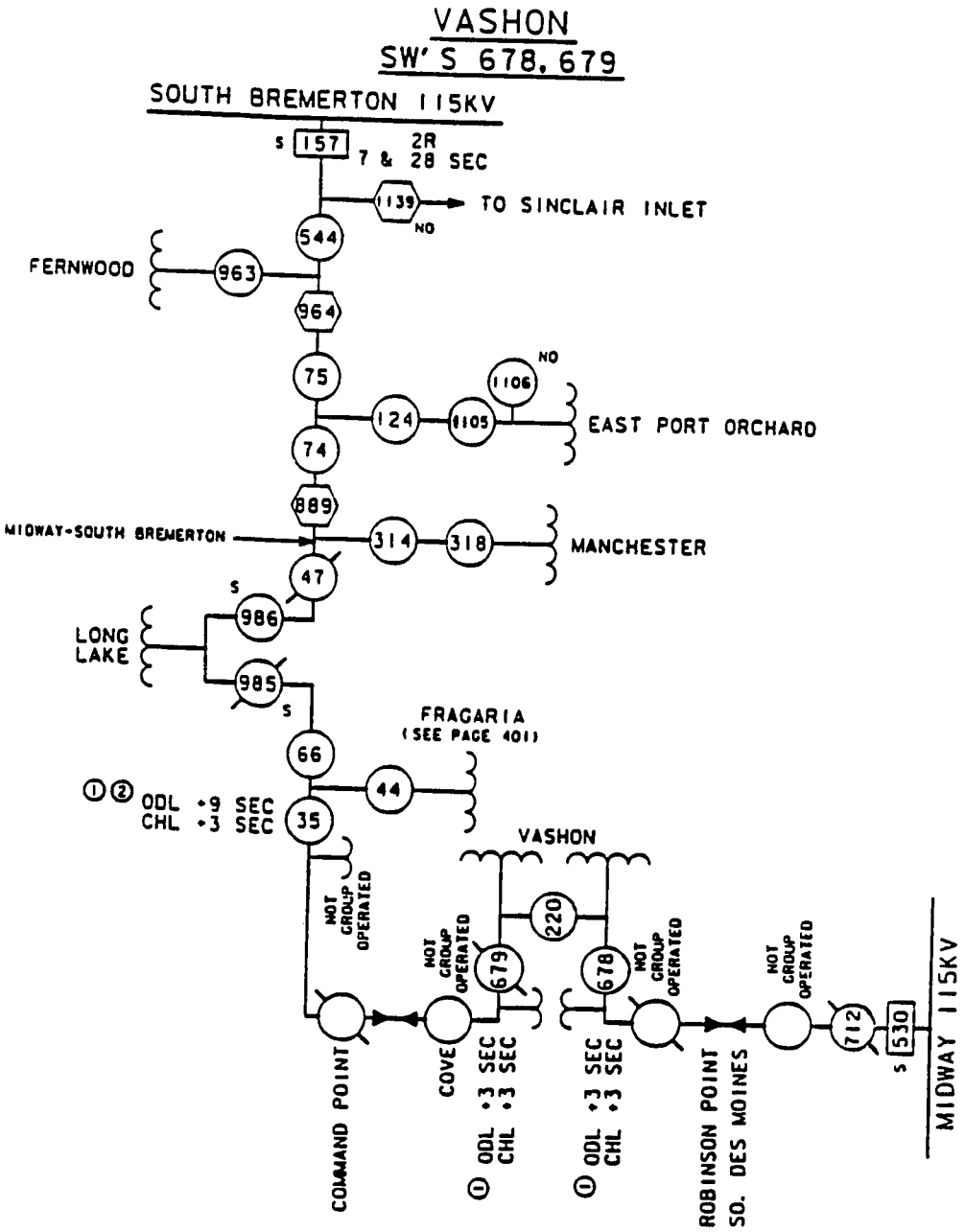


Figure 2-6 The SouthBremerton_Midway line

2.2.1 STRUCTURE OF CRAFT

The CRAFT system contains the same major parts as the VCES: a working memory, a rule base and an inference engine. The data structure of the working memory holds the information on line configuration, like that in Figures 2-5 and 2-6, while its knowledge base contains the rules used to determine fault location. The inference engine used in CRAFT is the same as that used in the VCES. The knowledge base and the inference engine of CRAFT permit the expert system to perform a reasoning process similar to an experienced dispatcher. It also provides operational guidelines for the dispatcher when restoring service by suggesting the proper sequence for opening and closing switches.

2.2.1.1 DATA STRUCTURE OF CRAFT

The CRAFT data structure is designed to model the system and to support the CRAFT rule base. The working memory contains the elements describing the line configuration. It also contains other necessary elements, such as tasks, alarms, and faults. Examples of element types defined for line and switching devices follows:

```
type line = element
(  id : symbol;              -- name of a branch of a line
   subject : symbol;         -- line identification number
   high_side : symbol;       -- name of the strong side of the line
   low_side : symbol;        -- name of the weak side of the line
   status : symbol;          -- status of the line
);
```

```
type device = element
(  id : symbol;              -- device name
   line : symbol;            -- line the device is on
   devicea : symbol;         -- name of the left neighbor device
   deviceb : symbol;         -- name of the right neighbor device
   class : symbol;           -- device type
   normal : symbol;          -- normally open or closed
   status : symbol;          -- current status, open or closed
   auto : symbol;            -- automatic or not?
   scada : symbol;           -- supervised or not?
   open_on : symbol;         -- logic for device to open
   close_on : symbol;        -- logic for device to close
   otime : integer;          -- delay time for open
   ctime1 : integer;         -- the first reclosure delay time
   ctime2 : integer;         -- the second reclosure delay time
   isolate : symbol;         -- substation to be isolated
   sensor : symbol;          -- sensor status, hot, dead or none
);
```

Note that the symbolic data type is widely used to describe objects. This is a common feature of expert systems in OPS83. But the OPS83 symbolic data type is incompatible with the character string type in FORTRAN. This problem must be solved when passing data between the working memory and a power system database (FORTRAN based).

2.2.1.2 RULE BASE OF CRAFT

Discussions with Puget Power dispatchers have identified the reasoning process and heuristics which are typically utilized in solving the fault isolation problem. The procedural information along with the empirical rules extracted from dispatchers form the knowledge base which consists of about 200 production rules [16]. A few assumptions are made to clarify the scope of the problem. These assumptions are also embedded in the rule base.

The important assumptions made in the CRAFT rule base are the following:

1) All power circuit breakers operate as designed and thus isolate a line fault to within a single line, i.e. the possibility of mis-coordination is not considered.

2) All other automatic switching devices function normally as well.

3) All faults are single line-section faults, i.e. bus faults and multiple-section faults on any particular line were not considered.

4) All switches and breakers have reached their final configuration, i.e. any automatic switching operations which are going to occur have already taken place.

5) Through the EMS or by communication with field personnel all device openings or closings are known to the dispatcher and updated in the system database.

These assumptions define the most common scenarios that Puget Power dispatchers face. More knowledge is certainly needed if any of these conditions is removed.

Certain techniques are employed in implementing the acquired knowledge. For example, the overall logical flow of CRAFT is to form a hypothesis of a line section fault and then validate and analyze this conjecture further through reasoning and/or physical testing of the power system. In addition to this framework, rules are organized under tasks. Each task may have subtasks. By successively creating tasks and subtasks the problem can be solved in steps similar to the process used by the dispatchers. Table 2-1 presents the task organization in the rule base [17].

Table 2-1 The task organization in the CRAFT rule base.

TASK 1:	Process Alarm
Subtasks:	Classify Problem
	Resolve Alarm
	Approve Proposed Actions
	Perform Bookkeeping
TASK 2:	Form Hypothesis
TASK 3:	Validate Hypothesis
Subtasks:	Gather Data
	Formulate Data
	Locate Fault
	Set up Testing
	Analyze Test Results
TASK 4:	Check Timing
TASK 5:	Test Line
TASK 6:	Suggest Solution

An example is given in the next section to illustrate the basic operation of CRAFT.

2.2.2 BASIC OPERATION OF CRAFT

The operation of CRAFT is initiated by alarms. The following scenario illustrates how CRAFT assists a dispatcher in analyzing alarms due to a fault on the line of Figure 2-6. Note that this discussion assumes CRAFT is integrated into a control center.

Assume that a fault appears between switches 47 and 986 on the line in Figure 2-6. The circuit breakers 157 and 530 trip at time t=0 and breaker 157 will have two reclosures, one at t=7 and the other at t=28 seconds. The automatic switches 679 and 678 will open after the line has been de-energized for 3 seconds (t=3). Breaker 157 fails its first reclosure at t=7 because it still "sees" the fault. At t=9, switch 35 opens because it senses that the line has been de-energized for 9 seconds. At t=28, breaker 157 fails the second reclosure and the entire line remains dead. At the control center, the dispatchers receive alarms from the supervised devices 157, 530 and, perhaps, phone calls from customers who lost power.

The alarms initiate CRAFT which retrieves the necessary data and analyzes the switch settings. Based on the intended operation of these switches, it attempts to identify the fault location. In this case, it cannot do so since the situation is ambiguous. The same set of alarms can be produced by other fault locations on the line. For example, a fault between 74 and 889 or between 66 and 985 would produce the same set of alarms.

Since the problem cannot yet be solved, CRAFT suggests a set of test actions:

OPEN 985 BY SUPERVISORY,
RECLOSE 157 BY SUPERVISORY.
YES/NO ? YES

When the dispatcher agrees, he/she implements the suggested actions. For this case, the reclosure fails because breaker 157 still "sees" the fault. CRAFT retrieves and analyzes the test results. Since the situation is still ambiguous, CRAFT suggests a second set of test actions:

OPEN 47
RECLOSE 157 BY SUPERVISORY.
YES/NO ? YES.

This time the reclosure is successful. CRAFT then concludes that the fault is located between 47 and 986 and suggests a sequence of actions to restore service to all customers:

OPEN 986 BY SUPERVISORY,
CLOSE 530 BY SUPERVISORY - CLEAR FIELD PERSONNEL,
CLOSE 679 MANUALLY,
CLOSE 985 BY SUPERVISORY - CLEAR FIELD PERSONNEL,
678 AND 35 WILL CLOSE AUTOMATICALLY.
AGREE (YES/NO) ? YES

2.2.3 A STAND-ALONE CRAFT TEST SYSTEM

The initial version of CRAFT was intended to evaluate and demonstrate the concept of this expert system [16]. Hence this system was developed as a "stand-alone" system, i.e. not implemented in a power system control center (Figure 2-7). Therefore it was necessary to develop a simulation of the power system behavior for the class of emergencies considered. This simulation replicates the operation of the automatic switches and circuit breakers in response to the status changes on a power line due to a fault and test actions. The simulation result is stored in the system database and then retrieved to the working memory by CRAFT as though it is coming from an actual power system. A relational information management system, UWRIM, was used to manage the system database and to provide for data communication between CRAFT and the simulator.

This test system is implemented on MicroVAXII. The simulator module is written in OPS83. In Figure 2-7, the solid lines between blocks stand for the data paths, while the dotted line stands for the control path.

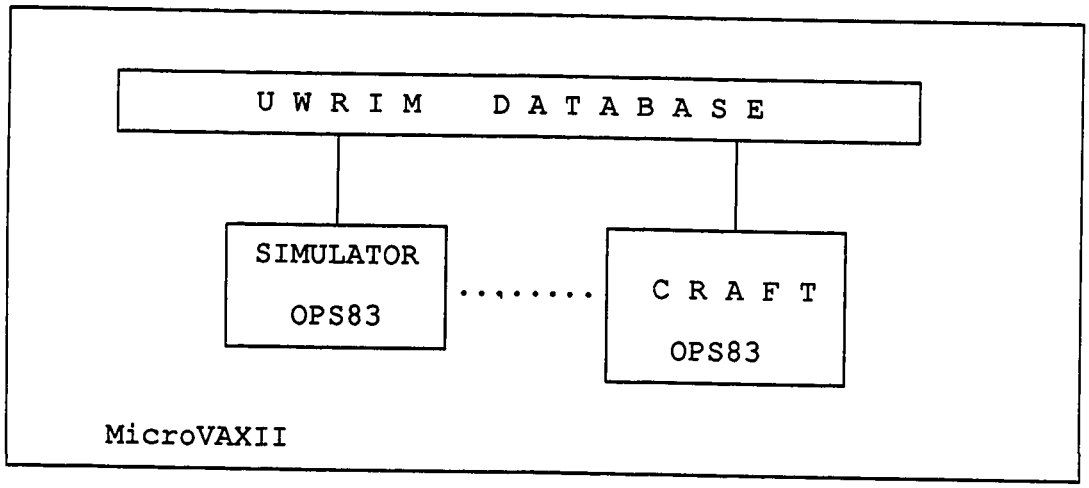


Figure 2-7 Stand-alone CRAFT test system

As in the stand-alone VCES system, a problem in passing data to the expert system arises from the different languages. The symbolic data type widely used in OPS83 can not map to the character string type in FORTRAN. An approach for communicating between the database and the expert system has been developed. The Appendix B shows how to map data types correctly between OPS83 and FORTRAN.

2.2.4 PERFORMANCE EVALUATION OF CRAFT

Since CRAFT is intended for an on-line system, correctness and time efficiency are of great importance. Dispatchers were relied on to evaluate the response of the expert system to various simulated fault scenarios on several sampled lines. The CPU time consumed in reaching a solution was measured to evaluate the efficiency. The CPU time required for each run of CRAFT was around 2-3 seconds on the MicroVAXII computer, with approximately 1-2 seconds of this time spent retrieving data from the database [6]. The performance of this demonstration system has been judged very satisfactory by the Puget Power dispatchers.

Note that for large power systems with many lines and devices the database access time will increase but, since the fault isolation problem is restricted to one line at a time, the time spent analyzing the fault will remain small. In other words, for large scale systems the time bottleneck is expected to be the database access.

2.2.5 DISCUSSION

Recall that the CRAFT rule base was built under a few important assumptions. Those assumptions, i.e. single line-section faults, reliable devices, available and reliable data, narrowed the problem scope and made it approachable by the available heuristics. When it is implemented in a control center, however, those assumptions may not always be valid. The situation may go beyond this scope due to uncertainties. Assistance under those conditions would be a great benefit.

As more heuristics are determined, some of the assumptions can be relaxed. For instance, the restriction on the availability of all device status can be lifted. In fact, an additional rule-based module, AUTOSTATUS, has been designed for that purpose and has been added to CRAFT to determine the actual status of the unsupervised automatic

switches [25]. In addition, if the power flow information, such as the status of substation taps and the line flows, is accessible, then the status of those unsupervised "manual" switches can also be determined. This is referred as the "PSEUDO POINT" problem [25].

2.3 APPENDED INTEGRATION OF CRAFT INTO A CONTROL CENTER

To give the dispatchers an opportunity to evaluate the utility of CRAFT in an actual operating situation, it is necessary to integrate it into an EMS center using either an appended or embedded approach. The diagram in Figure 2-8 and 2-9 illustrate the concepts of the two approaches. The embedded approach involves integrating CRAFT directly into the EMS system computer, database and displays. The appended approach involves separate computers, separate operator displays, separate databases and a data link for transmitting data between the EMS and CRAFT computers.

The appended approach was considered more practical for implementing CRAFT at the Puget Power system for several reasons. First, no production system language is currently available for the MODCOMP computer which is used in the Puget Power operating center. This eliminates the embedded approach but is no problem for the appended approach. Second, even if such a language existed, the potential for disrupting the power system normal operation makes it difficult to pursue the embedded approach. On the other hand, the appended approach has its own advantages: more computing capabilities, greater portability from one control center computer to another , and higher availability of the computer time for expert system development. This appended approach has some similarity with efforts to couple "AI" computers with standard machines running traditional analysis tools [34] although the motivation there is to gain the performance benefits of the special purpose machine.

Besides the CRAFT program, major software functions that are required for the appended system are: a SCADA interface on the EMS computer, a Datalink, and a Data interface on the MicroVAXII side. A detailed description of the conceptual design of those functions is available in [6]. The purpose of these functions is to provide the expert system with the necessary data and alarms from the EMS computer and to process data requests from CRAFT.

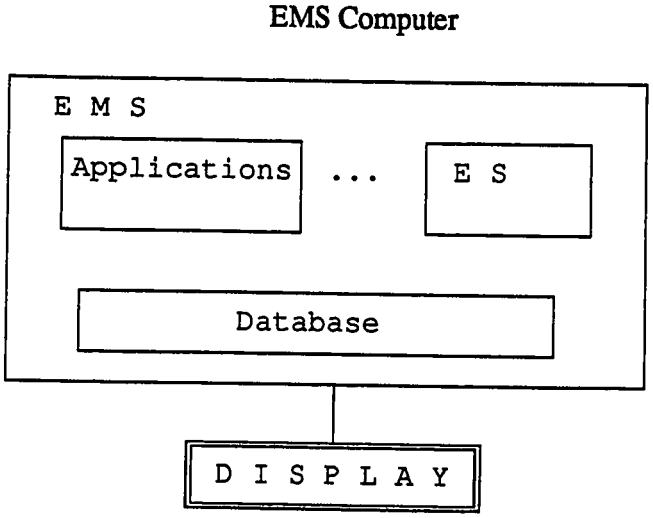


Figure 2-8 The embedded approach

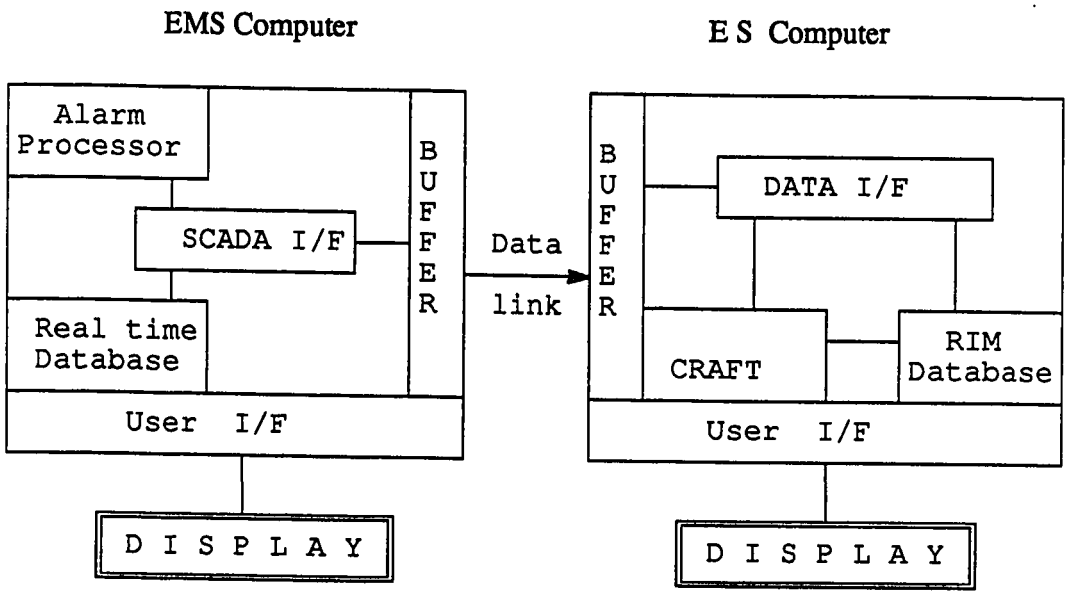


Figure 2-9 The Appended Approach

CRAFT has been installed in the Puget Power operating center using the appended approach. To maintain the integrity of the host system on this implementation stage, the data link is one-way only. The necessary data and alarms are sent from the EMS computer to the CRAFT computer. No data request or switch operation command is sent from the CRAFT computer to the main computer. Rather, when CRAFT suggested actions are approved by the dispatcher, the actions are carried out manually by the dispatcher. Hence, the control loop is actually closed by dispatcher intervention.

This appended approach has its limitations. CRAFT can not directly access the EMS database. The one-way datalink limits CRAFT to receive only the predefined data items. The CRAFT computer has to maintain its own database consistent with the EMS database. In addition, CRAFT can not take advantage of the software and utilities, such as the standard MMI facilities, provided by the EMS. The additional, required CRT and keyboard may crowd the console area which may cause confusion especially in emergency conditions.

These limitations may be relaxed or removed if the database integration or the full integration approach is implemented.

CHAPTER 3

DATABASE INTEGRATION APPROACH

The database integration (DBI) approach features a single computer, a unified database and standard displays. This approach was referred to as the embedded approach when embedding CRAFT into the DTS [26]. In some publications [10,19] the embedded approach is called the monolithic method in contrast to the distributed or appended method.

The database integration approach is expected to be more desirable compared to the appended approach. First, the dispatcher can interact with the expert system (ES) using the standard EMS interface, i.e. the standard displays, console and command set. Hence, the ES should "feel" natural to the dispatcher. Second, an unknown but expected benefit of DBI is that the performance (response time) of the expert system should be better because of the "tighter" coupling between the expert system and the system real-time database. As a result, the system data are directly accessible to the ES. For instance, the available power flow information may help to determine the status of some unsupervised devices. An additional consideration is that the implementation might be easier. For example, it is not necessary to build the data transfer link between two separate computers, a transfer that must be selective so the data link does not become clogged with a massive amount of irrelevant EMS system data.

The performance of an embedded system is difficult to predict and experiments on an actual system are almost impossible because of the likelihood of disrupting normal operation. Consequently, it is desirable to have an environment for developing an embedded system "off line". A Dispatcher Training Simulator (DTS) provides such an environment if it can be integrated with the ES. Such an integrated system will serve three purposes. First, it will provide a test system for performance evaluation and dispatcher acceptance in a realistic setting. Second, it will become a tool for training dispatchers in the use of the expert system and third, it will become a prototype for integrating an ES into EMS centers.

The immediate purpose for studying the embedded approach to CRAFT is to answer three basic questions:

1. Can an expert system that is built using an AI language tool (OPS83) be successfully integrated into a EMS system which uses standard, procedure based software for programs and data management?

2. Will the embedded expert system degrade the overall EMS system performance to an unsatisfactory level?

3. Does the embedded approach actually yield the expected advantages, i.e. greater dispatcher satisfaction due to the customary displays as well as faster response of the expert system?

CRAFT has been successfully integrated into the Dispatcher Training Simulator developed by the ESCA Corp. The DTS and CRAFT run in the VMS operating system on a DEC MicroVAXII. CRAFT is written in OPS83 and the DTS is written in FORTRAN. The successful integration basically answers "YES" to the first question posed above. In fact, this CRAFT-DTS system is one prototype for embedding other expert systems into an EMS. It can also be used to test the other questions asked above.

Several technical issues had to be solved to accomplish the integration of CRAFT into the DTS. Functionally, the issues can be classified as:

Modification and Extension of the DTS
- Modeling line fault effects
- Modeling automatic switches
- Modeling load taps and Puget Power lines

Integration of CRAFT
- Database extension
- Building displays for Puget Power lines
- Interface routines and translators
- Process coordination

The solutions to these issues demonstrate how an ES tool can be integrated with the DTS/EMS environment. However before discussing these technical integration issues, it is necessary to understand the organization of the DTS and the software environment called HABITAT. Section 3.1 presents the background of the HABITAT system and the basic structure of the DTS. Section 3.2 gives an overview of the CRAFT-DTS system. Section 3.3 concerns the technical problems of embedding CRAFT into the DTS. Section 3.4 presents tests and results. Evaluation and comments make up Section 3.5.

3.1 FEATURES OF THE DTS AND HABITAT

The ESCA dispatcher training simulator (DTS) and its HABITAT environment is used for simulating an EMS system. HABITAT is a software environment for development and implementation of engineering systems. It contains a database management system (DBMS), an application manager, a savecase manager, a user interface manager and utilities. The SCADA systems used in the power system control center and the DTS are both built using HABITAT.

As with any software system that is integrated through a database management system, the HABITAT database on the DTS is the central data reservoir for all the input data for the individual software modules and accumulates all the results from these modules. The database integration approach was accomplished by exploiting this central database. Extending the database to provide the direct data exchange between the DTS and the expert system process provided the basic mechanism for that integration. The ESCA HABITAT system provides utilities for man-machine interaction by supporting the graphics used in the displays and coupling those displays with the dynamic data. These utilities facilitate building customized displays for the expert system.

The DTS contains a power system model and an EMS model [35]. Figure 3-1 provides one view of a DTS where the items above the horizontal line represent the power system model and those below represent the EMS center. The solid paths on the diagram indicate the data path of the individual software modules to the database and the broken paths indicate the approximate sequence of execution of the separate blocks which are controlled by the process manager. Note that, the process manager is not the focus of the database integration approach, but it will be important in the full integration approach. The very general labels given to these blocks are intended to be only

suggestive of the type of functions performed. A more complete description of the DTS is provided in [32]. The power system and EMS models communicate through a central data base management system which is a component of the ESCA HABITAT system. In the DTS, the underlying database has separate divisions for the power system and EMS models. In an actual EMS center, such a database would still exist but contain only the division for the EMS. Hence, it is believed that the DTS provides a realistic EMS environment for this integration work.

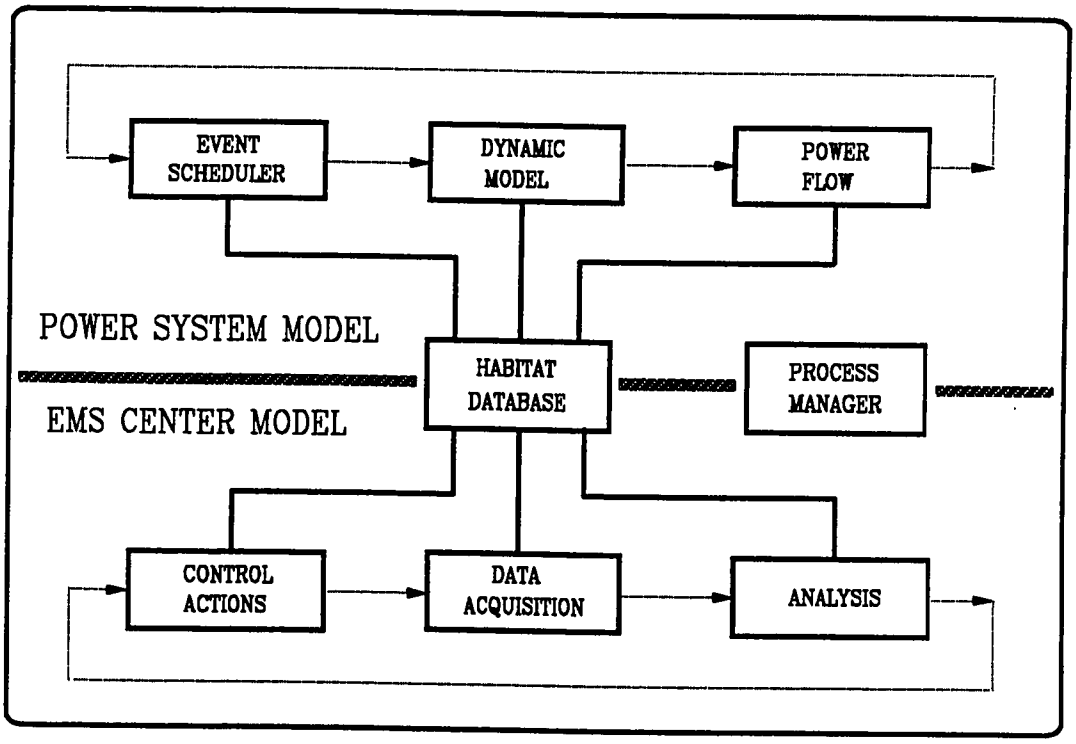


Figure 3-1 Basic structure of the DTS

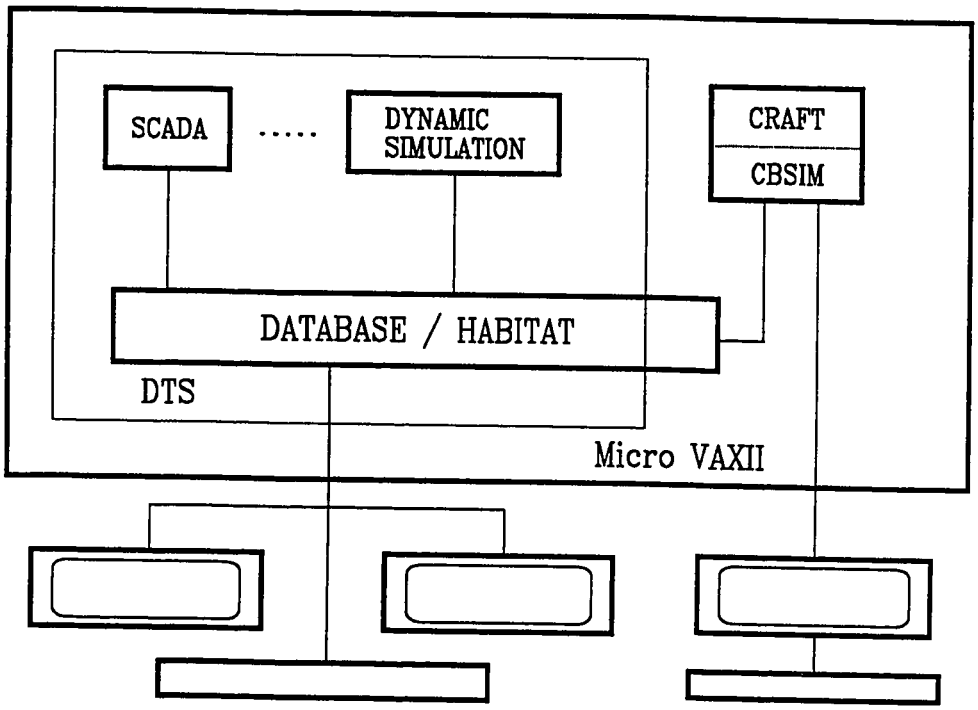


Figure 3-2 The CRAFT-DTS system

3.2 CRAFT-DTS SYSTEM

Figure 3-2 gives a block diagram of the CRAFT-DTS system. In this system, the DTS simulates the system power flow, automatic switching in response to a fault, data collection and display as well as managing the dispatcher interaction. CRAFT analyzes data to determine the fault location and advises the dispatcher on appropriate responses. We would also expect the DTS to simulate the circuit breaker response to faults. As will be described below, the version of the DTS we used does not have this capability so it was necessary to add this feature using an additional OPS83 module, CBSIM, the circuit breaker simulator. The result is an interesting example of using a program written in a production system language to supplement a standard procedural based program.

The basic means for interfacing the DTS and CRAFT is through the SCADA database. CRAFT can be added by making it an additional application module interacting with other applications via an extension of the DTS database. This database extension contains the new data items required by CRAFT.

The HABITAT DBMS is designed for real-time, on-line data storage and retrieval operations. This results in faster operation than previous versions of CRAFT. HABITAT also provides facilities for the user to design customized displays and interfaces. Therefore, the standard DTS displays can be supplemented with new displays to monitor CRAFT operations. These new displays can be built using the same user interface manager as is used for the DTS.

3.3 TECHNICAL ISSUES FOR EMBEDDING CRAFT INTO THE DTS

3.3.1 MODIFICATION AND EXTENSION OF DTS

3.3.1.1 MODELING LINE FAULT EFFECTS

The overcurrent protection system in the DTS was not designed to response to high speed events like faults. Hence no line or bus fault current protection system is available. The basic problem is that the currents are computed by a load flow routine [35,37] at an interval determined by the user. Four seconds is a common interval. Hence the DTS is not intended to simulate the transient currents due to a fault. As a result, the control of circuit breakers in response to faults must be provided to the DTS. Then the DTS can model the effects on power flow, automatic switches and SCADA operations.

Our solution is to use a supplemental program, Circuit Breaker Simulator (CBSIM), to determine what the circuit breaker response to faults ought to be and to send control signals to open or close the DTS circuit breakers. We have chosen to write this program in OPS83, the production language used for the expert system CRAFT. CBSIM is not an expert system in the usual sense because it does not depend on a knowledge base representing an expert's experience and behavior. Rather, it models a physical system, the types of models that standard procedural languages are usually used for. However, the behavior of this system is rather complex with a number of conditions and special

circumstances that must be adequately represented. We found OPS83 to be a very convenient language for this purpose by using rules to describe how the physical system functions. This is the same concept used for the system simulator in our stand-alone version of CRAFT [16]. We think this is an example of how the power of production system languages is likely to lead to their replacing some procedural language programming. The resultant system is the CRAFT-CBSIM-DTS system of Figure 3-2.

To control a circuit breaker's response, the location of a fault must be specified by the user at the start of a simulation session. Then both the initial operation of breakers and subsequent reclosure attempts are controlled by CBSIM. An example of a basic rule in CBSIM to guide a circuit breaker's (CB) response to a fault is:

> Assume all CBs are working properly.
> If a CB can 'see' a fault
> then
> the CB should be tripped and
> schedule an event to OPEN the CB on the DTS.

Figure 3-3 illustrates how CBSIM and CRAFT interact through the database. When a fault is specified, CBSIM maintains control until all automatic circuit breaker actions in response to the fault have taken place. Then control is passed to CRAFT for analysis. When testing is a part of the dispatcher response, control is passed back to CBSIM to manage any resulting circuit breaker operations. The example in Section 2.2 illustrates the operation of CRAFT. The reader is referred to [6,16] for a thorough discussion of the features of CRAFT. In the remaining discussion we will use the simple term CRAFT to refer to the combined CRAFT/CBSIM system.

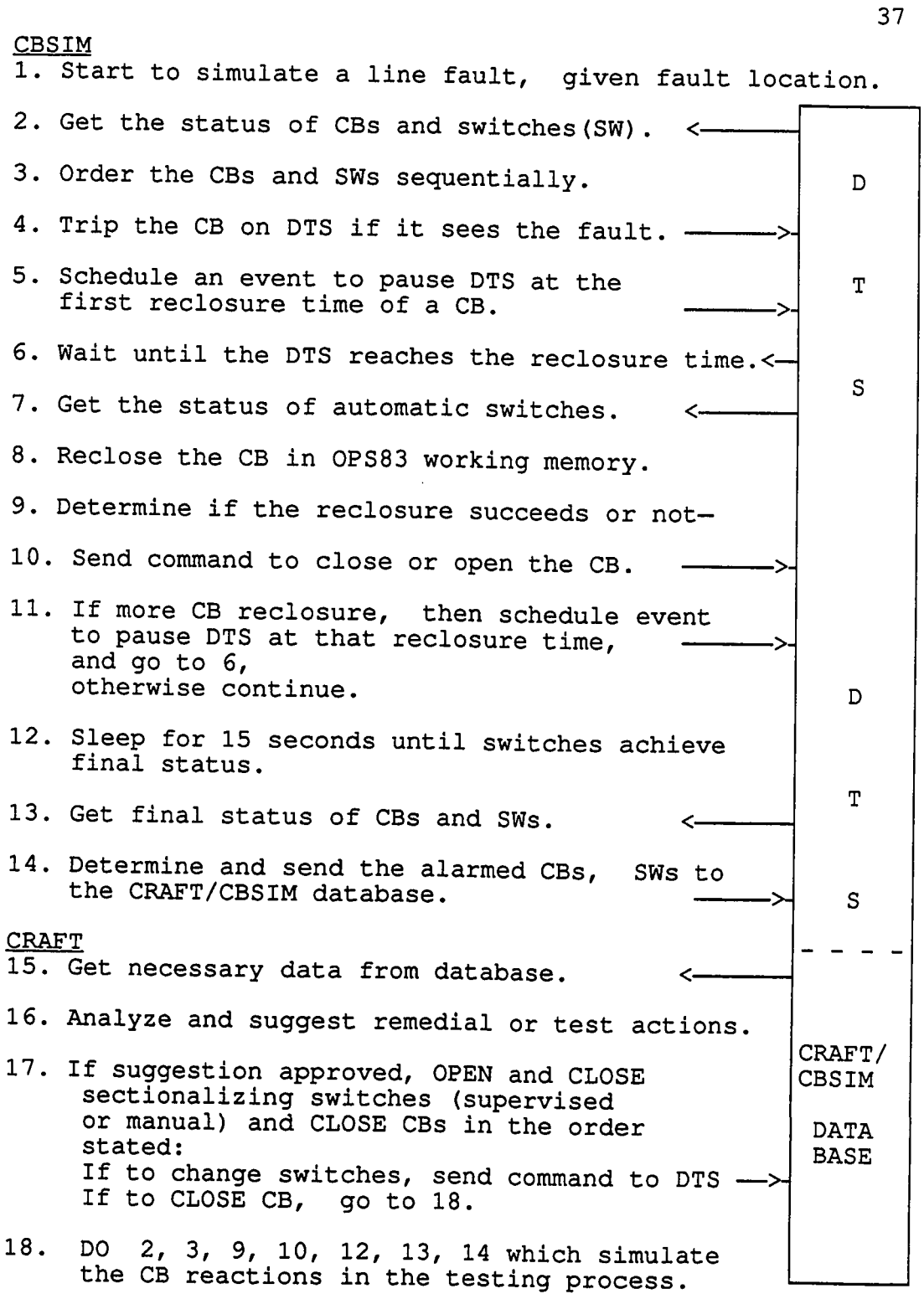


Figure 3-3 Data flow control in CRAFT-CBSIM-DTS system

3.3.1.2 MODELING AUTOMATIC SWITCHES

The customer restoration problem solved by CRAFT exists because of the use of automatic switches in the Puget Power system. However, these devices were not modeled in the DTS. Hence, the DTS must be modified to incorporate all the different types of automatic switches used by Puget Power. Table 3-1 lists the types of automatic switches modeled. Combinations of these switches, such as ODL+CHL, are also used by Puget Power to realize additional operating logic capabilities.

Table 3-1 A List of automatic switch types

Type	Name
ODB	Open on Dead Bus
ODL	Open on Dead Line
CDB	Close on Dead Bus
CDL	Close on Dead Line
CHL	Close on Hot Line
CHB	Close on Hot Bus
ATDB	Automatic Transfer on Dead Bus
ATDL	Automatic Transfer on Dead Line

The solution to modeling these automatic switches is to use circuit breakers and relays which are standard components in the DTS. The logic of two components can be combined in different ways to model the behavior of each type of automatic switch. Hence, an automatic switch model consists of a circuit breaker (or a logical switching device) and a voltage relay:

Autosw = CB + Voltage Relay

As a result, the Autosw model is represented by two records in the DTS database: a circuit breaker record and a voltage relay record. Figure 3-4 shows an ODB and its model in the DTS followed by the circuit breaker record and the voltage relay record.

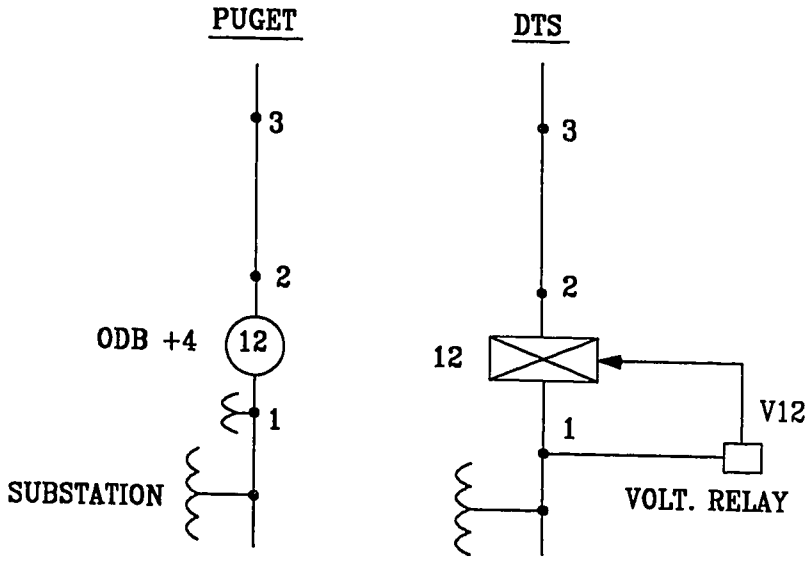


Figure 3-4 Modeling automatic switch ODB

Circuit Breaker record:

Name of CB:	12
From node:	1
To node:	2
Normal status:	closed

Voltage Relay record:

Name of VRY:	V12
Name of activated CB:	12
Name of sensed node:	1
Min activate voltage (p.u.):	0.60
Time delay before Min trip (sec):	4
CB action on Min volt violation:	open

In the model for an ODB (Open on Dead Bus) switch, a relay is assigned to a circuit breaker and is designated to sense a node. Note that a relay parameter called "Min activate voltage" is set to 0.6 p.u. volts. If the bus (node) voltage drops below 0.6 p.u. then the node is considered dead and the relay trips the associated CB. All the switches in Table 3-1 can be represented by similar combinations of circuit breakers coupled with voltage relay logic. The complete set of models is given in Appendix A.

3.3.1.3 MODELING LOAD TAPS AND PUGET POWER LINES

Another feature of Puget Power lines that CRAFT is concerned with is the numerous load taps. These, also, are not standard features of the DTS. However, they can be readily modeled in the DTS using a substation, a transformer and an attached load as shown in Figure 3-5. All of these are standard DTS components.

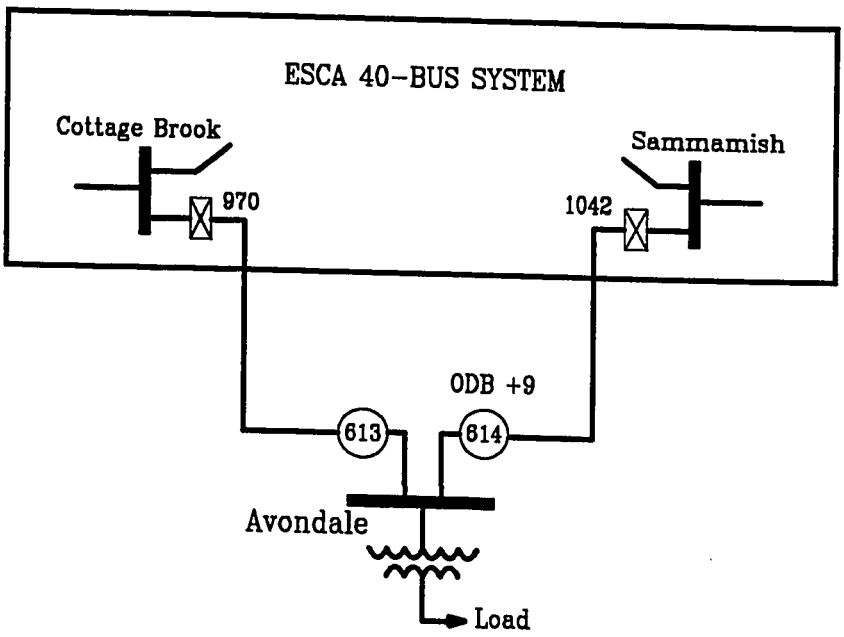


Figure 3-5 A Puget Power line added to the ESCA 40-bus system

Actually, Figure 3-5 indicates how to add an entire Puget Power line to the existing ESCA 40-bus system model for experimentation. Shown here is one of the simplest test lines. From the Puget perspective, this line contains one load tap (Avondale), one automatic switch (614), one manual switch (613), and two circuit breakers (970 and 1042). In terms of DTS components, this line consists of 3 substations, two line segments, 4 circuit breakers, a voltage relay, a transformer and a load to the transformer.

3.3.2 INTEGRATION OF CRAFT WITH DTS

3.3.2.1 DATABASE EXTENSION

The basic means for interfacing CRAFT with the DTS is through the HABITAT database. Figures 3-2 and 3-3 indicate that the database is extended to include the new data items required by CRAFT. The extended database partition consists of six records:

NAMES
LINES
DEVICE
SUBS
ALARMS
TESTCB

Each record is defined by a tuple (an ordered set of elements) of attributes, as given below. The record name and attribute name are in upper cases with the definition in lower case in parentheses.

NAMES:

NUM_NAMES	(line identification number)
NAME_NAMES	(line name)

LINES:

ID_LINES	(name of a branch of a line)
HISIDE_LINES	(name of the strong of the line)
LOWSIDE_LINES	(name of the weak side of the line)
REGION_LINES	(geographical region of the line)
STATUS_LINES	(status of the line)

DEVICE:

ID_DEVICE	(device name)
LINE_DEVICE	(line the device is on)
DEVICEA_DEVICE	(name of the left neighbor device)
DEVICEB_DEVICE	(name of the right neighbor device)
CLASS_DEVICE	(device type)
NORMAL_DEVICE	(normal open or closed)
SS_DEVICE	(pointer to CB record in DTS database)
AUTO_DEVICE	(automatic or not?)
SCADA_DEVICE	(supervised or not?)
OPENON_DEVICE	(logic for it to open)
CLOSEON_DEVICE	(logic for it to close)
OTIME_DEVICE	(delay time for open)
CTIME1_DEVICE	(first reclosure delay time)
CTIME2_DEVICE	(second reclosure delay time)
ISOLATE_DEVICE	(substation to be isolated)

SUBS:

ID_SUBS	(substation name)
LINE_SUBS	(line the subs is on)
DEVICEA_SUBS	(name of the left neighbor device)
DEVICEB_SUBS	(name of the right neighbor device)
STATUS_SUBS	(status of the subs, energized or not)

ALARMS:

DEVICE_ALARMS	(alarmed device name)
LINE_ALARMS	(line the device is on)
STATUS_ALARMS	(status of the alarm)

TESTCB:

DEVICE_TESTCB	(device to be tested)
ACTION_TESTCB	(test action, to open or to close)
STATUS_TESTCB	(status of the test action)

Note that the status of a device does not explicitly appear in the DEVICE record. Rather, this dynamic field is represented by a pointer SS_DEVICE which points to a circuit breaker record in the DTS database. Thus, the device status is uniquely determined in the DTS database. This extended database partition not only supports CRAFT, but also maintains the integrity of the DTS database by leaving it undisturbed.

3.3.2.2 BUILDING DISPLAYS FOR PUGET POWER LINES

Integrating CRAFT into the HABITAT database facilitates the development of customized displays for Puget Power lines. In particular, color, graphical line displays can be easily built using the HABITAT user interface manager and the existing standard picture elements. Any datum in the database can be designed to appear on a display. For example, the substation name, the device name, the real and reactive power flows and bus voltages can be shown on the display. Of course, the display can also be designed to reflect status changes of circuit breakers or switching devices.

Two levels of displays were built: the complete system one-line diagram and detailed displays of individual Puget Power lines. The system one-line diagram gives an overall view of the system including line flows and bus voltages. Each individual line displays the line configuration, status of switching devices and important associated data. The line display is very similar to the standard diagram in the Puget Power automatic switch book, with which the dispatchers are familiar. These Puget Power line displays supplement existing DTS displays for monitoring CRAFT and for user interactions.

3.3.2.3 INTERFACE ROUTINES AND TRANSLATORS

In addition to the extended database, it is necessary to provide for the movement of data between the HABITAT database and the working memory of the CRAFT/CBSIM OPS83 modules. The large dashed block in Figure 3-6 and Figure 3-7 contains FORTRAN

modules that accomplish this data flow and provide process control. The solid lines indicate the data paths while the dashed lines represent controls. The translators between the working memory and the FORTRAN routines are necessary for mapping data correctly between OPS83 and FORTRAN data types. These data flow modules and translators constitute the program interface. CRAFT determines which subroutine in this interface is to be called at any time during system operation.

A basic problem in developing the interface arises from the different data structures provided by OPS83 and FORTRAN. More specifically, symbolic (alphanumeric) data is incompatible between the two programming environments. This problem was initially encountered when UWRIM (FORTRAN based) was used to support CRAFT [6]. In FORTRAN, character strings of a fixed length provide the basis for symbolic data. OPS83 relies on variable length character strings called symbols. As a simple example, in FORTRAN, the switch status information 'closed' or 'open' could not be assigned to the same variable without adding blank characters as filler, i.e. 'closed' and 'open '. OPS83 requires no such character padding. Thus for name and other descriptive information, some data translation must be performed.

Two approaches have been identified to solve this problem. One is to use a temporary text file and write/read operations, a time consuming process. The other and more efficient approach is to use parameters and translators. The second approach is chosen due to its higher efficiency. Appendix B provides examples to illustrate how the parameters and translators map the data.

With the issue of compatibility resolved, data can pass between the OPS83 working memory and the HABITAT database by calling FORTRAN subroutines. According to the convention of terminology, the FORTRAN procedures are called 'subroutines' and the OPS83 procedures are called 'procedures'. A function is a special type of procedure or a subroutine that has a value associated with it. A remarkable difference between an OPS83 procedure and a FORTRAN subroutine is that FORTRAN subroutines can not call OPS83 procedure but the OPS83 procedures can call FORTRAN subroutines. Another difference between OPS83 and FORTRAN is that the OPS83 procedure name is case sensitive while FORTRAN is not. In the following discussion, OPS83 procedures and functions are indicated by names in lowercase while the FORTRAN subroutines are indicated by names in uppercase. Since our version of OPS83 (2.1) allows at most 6

parameters in the external procedures, the parameters in some interface routines (e.g. GETDEVICES), have to be array type to allow all necessary data items to be passed. The OPS83 procedures and the associated FORTRAN subroutines used in CRAFT to access the database are described below.

function get_alarms(&device, &line, &status):logical

The data type of all parameters is symbol. In this OPS83 function, a FORTRAN subroutine GETALARMS is called to get the name of an alarmed device, the line it is on and the status of the alarm. The record was defined previously in the database extension section. A working memory element is created based on the retrieved data. The function returns true (1b) if at least one alarm is obtained from the database. Otherwise it returns false (0b). Based on the value of this function, CRAFT determines whether to work on the problem or wait for alarms.

GETALARMS (ID_ALARMS, LINE, STATUS)

This FORTRAN subroutine gets new alarms from the CRAFT/CBSIM database. The data type of all parameters in the subroutine is character arrays. After a record of ALARMS is retrieved, the translators within the function translate these character arrays into symbols. Then an alarm working memory element is created. Appendix B gives examples to show how the translators work.

procedure get_lines(&line, &subject)

This OPS83 procedure is to get the record (LINES) for the alarmed line and to create a working memory element based on the retrieved data. The parameter &line is a symbol which gives the name of the alarmed line and &subject is the identification number of the line. If the line has branches, then all the branch records are obtained in this procedure call. The branches of a line are identified by the same subject (the identification number). First, the symbol &line is translated into ID_LINES, a character array of 20 characters. Then an external FORTRAN subroutine GETLINES is called:

GETLINES (ID_LINES, HISIDE, LOWSIDE, REGION, STATUS, FLAG)

For a given line ID_LINES, this routine accesses the database to get the name of both sides of the line, the geographical region and the status of the line. This

record was defined previously in the database extension section. The FLAG indicates success of the call.

procedure get_subs(&line, &subject)

This procedure is to get the records (SUBS) of all the substations on the alarmed line and to create working memory elements based on the retrieved data. The parameter &line is a symbol which is the name of the alarmed line. The parameter &subject is the identification number of the line. First, the symbol &line is translated to ID_LINES, a character array of 20 characters. hen an external FORTRAN subroutine GETSUBS is called:

GETSUBS (SUBS, ID_LINES, DEVICEA, DEVICEB, STATUS, FLAG)

For a given line ID_LINES, the routine gets a tuple of record SUBS from the database. The record was defined previously in the database extension section. The FLAG indicates success of the call. The character arrays retrieved are translated into symbols by the translators within the OPS83 procedure. A working memory element is created to store the record.

procedure get_devices(&line, &subject)

This procedure is to get the records (DEVICE) of all devices on the alarmed line and to create working memory elements. The parameter &line is a symbol which is the name of the alarmed line and &subject is the identification number of the line. First, the symbol &line is translated to ID_LINES, a character array of 20 characters. Then an external FORTRAN subroutine GETDEVICES is called:

GETDEVICES (ID_LINE, SUBS, ARRAY1, ARRAY2, ARRAY3, FLAG)

For a given line (or branch) ID_LINE, this routine retrieves the record of a switching device on the line from the database. The device status is retrieved by calling GETDTSCB using the device pointer SS_DEVICE:

GETDTSCB (SS_DEVICE, STATUS_DEVICE)

As mentioned above, the array data types are used to pack the data items into 6 parameters. Thus, ARRAY1 contains 3 integers, ARRAY2 contains 4x5 characters, ARRAY3 contains 12x5 characters. The retrieved character arrays are translated into symbols within the procedure get_devices. OPS83 working

memory elements are created to store the retrieved device data. If FLAG >= 0, then the retrieval is successful and the OPS83 procedure will call GETDEVICES again to get the next device on the line. When all devices on the line are retrieved, FLAG returns '-1' and the procedure is terminated.

procedure put_lines(&line, &status)

This procedure is to update the status of a line. First, the symbolic data &line and &status are translated into character arrays ID_LINES and STATUS. Then the following subroutine is called to update status of the line in the database:

PUTLINES(ID_LINES, STATUS)

Similarly, the procedure put_subs and subroutine PUTSUBS is used to update the status of a substation in the database:

procedure put_subs(&subs, &status)
PUTSUBS(ID_SUBS, STATUS)

procedure put_testcb(&device, &action)

This procedure is to send a new test circuit breaker event to the CRAFT/CBSIM database for CBSIM to carry out the test action. First, the symbolic data &device, &action are translated into character arrays DEVICE and ACTION. Then the subroutine PUTTESTCB is called to enter a new TESTCB record into the database:

PUTTESTCB(DEVICE, ACTION)

If the switching operation involved in an action does not reclose a circuit breaker, then this operation does not need CBSIM. Rather, it is carried out by procedure put_device(&device, &action) and subroutine PUTCB(DEVICE, ACTION).

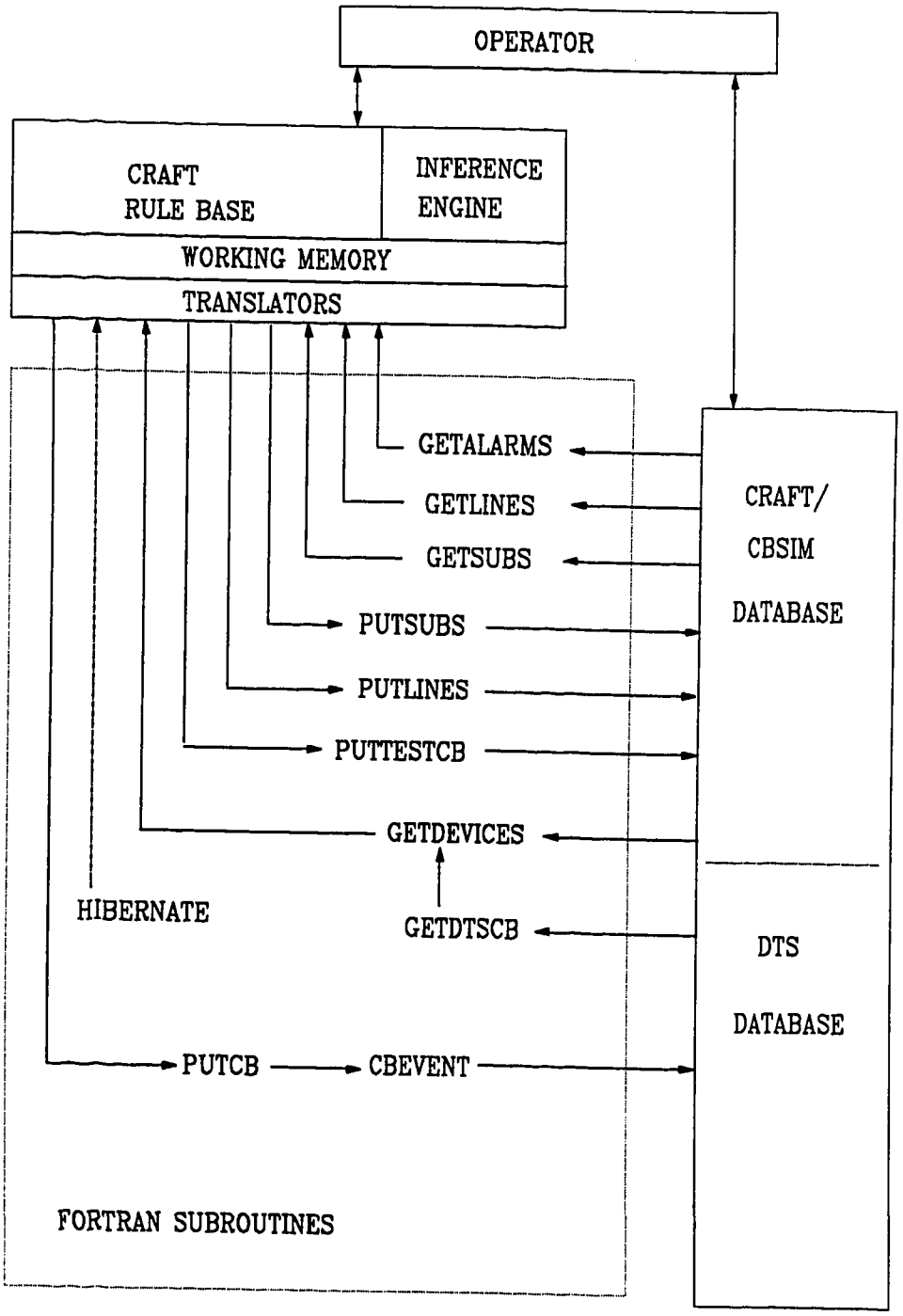


Figure 3-6 Interface between CRAFT and the DTS

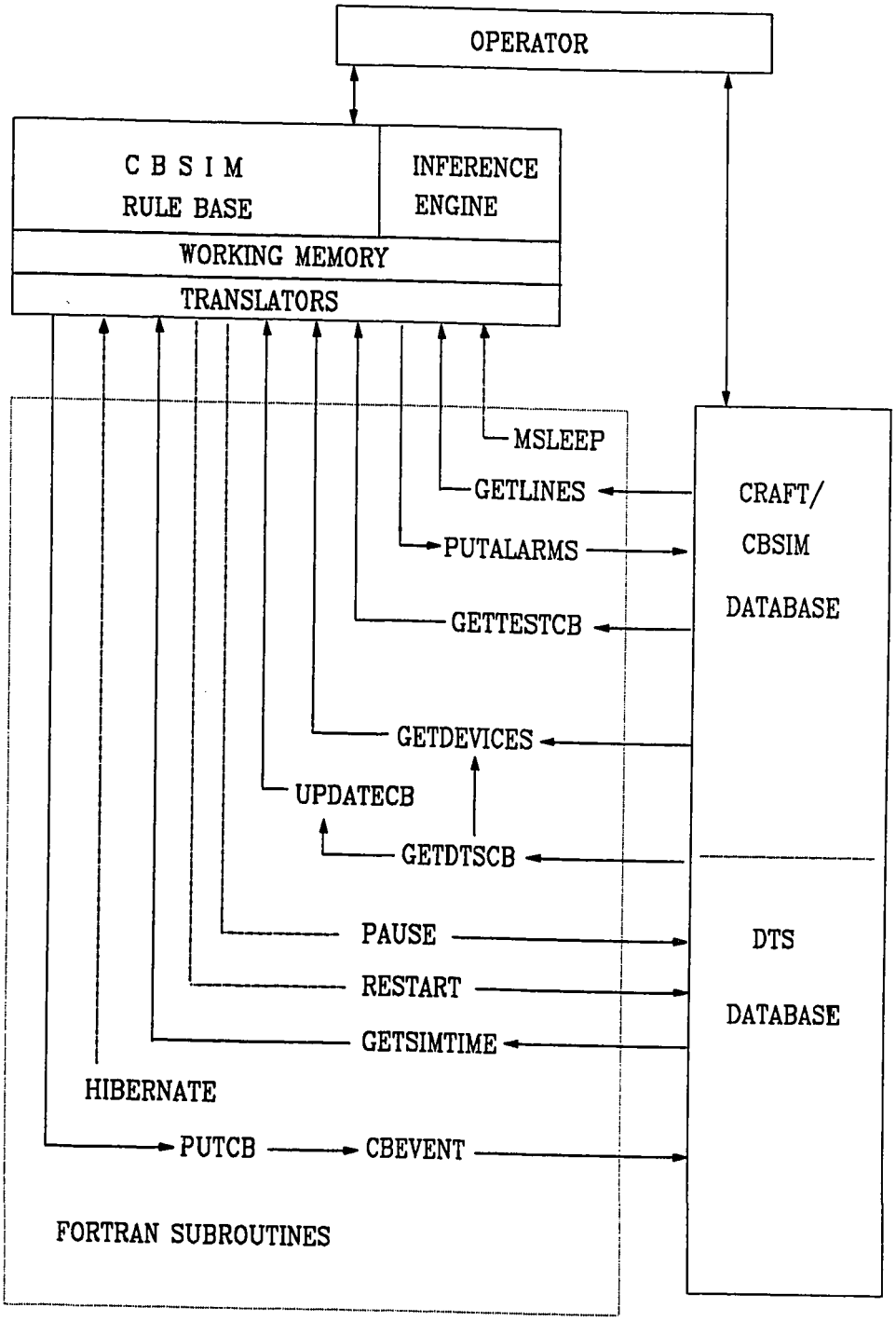


Figure 3-7 Interface between CBSIM and the DTS

procedure put_device(&device, &action)

First, the symbolic data &device and &action are translated into two character arrays DEVICE and ACTION. Then, these parameters are passed to subroutine PUTCB:

PUTCB(DEVICE, ACTION)

It accesses the CRAFT database to get the pointer of the DEVICE, SS_DEVICE, which points to the circuit breaker record in the DTS database. The pointer is used in the subroutine CBEVENT to schedule a switching event on the DTS:

CBEVENT(SS_DEVICE, ACTION)

The subroutine HIBERNATE is used for process control.

This routine is called by CRAFT to wait for the circuit breaker test results, or to wait for new alarms. As a result of this call, CRAFT hibernates until some other process activates it.

The following OPS83 procedures (lower case) and associated FORTRAN subroutines (upper case) used in CBSIM are the same as those discussed in the CRAFT interface section:

procedure get_lines(&line, &subject)
GETLINES (ID_LINES, HISIDE, LOWSIDE, REGION, STATUS, FLAG)
procedure get_devices (&line, &subject)
GETDEVICES (ID_LINE, SUBS, ARRAY1, ARRAY2, ARRAY3, FLAG)
GETDTSCB (SS_DEVICE, STATUS_DEVICE)
procedure put_device (&device, &action)
PUTCB(DEVICE, ACTION)
CBEVENT(SS_DEVICE, ACTION)

The new procedures and subroutines are as follows:

procedure put_alarms(&device, &line, &status)

This procedure puts alarm information into the CRAFT/CBSIM database. The data type of all parameters is Symbol. The values of these parameters are determined in CBSIM. The parameters are first translated into character arrays.

Then a FORTRAN subroutine PUTALARM is called to store alarms into the database.

PUTALARM (ID_ALARMS, LINE, STATUS)

All parameters in this subroutine are character arrays which can be assigned to variables of the same type and mapped to character strings by the FORTRAN EQUIVALENCE declaration. Then these character strings are entered into the database as a new record.

procedure updatacb(&device, &status)

This procedure is called within CBSIM at the reclosure time of a circuit breaker. It is to update the status of automatic switches in the working memory. The following two FORTRAN routines are called to access the database:

UPDATECB(DEVICE, STATUS)

It uses the pointer SS_DEVICE and calls GETDTSCB to obtain the device status:

GETDTSCB (SS_DEVICE, STATUS_DEVICE)

procedure get_testcb(&device, &action)

This procedure gets a new test circuit breaker event from the CRAFT/CBSIM database and creates a working memory element for CBSIM to carry out the test action. First, it calls the subroutine GETTESTCB:

GETTESTCB(DEVICE, ACTION, NEW)

When a new test record is retrieved, the status of the record in the database is reset to 'old'. The retrieved DEVICE and ACTION are character arrays which are translated into symbols, &device and &action and are assigned to a working memory element for CBSIM to work on.

The following subroutines are used for process control:

HIBERNATE

CBSIM calls this routine to hibernate itself after completing simulation of the circuit breakers' response to a fault or a test action.

PAUSEDTS(REALTIME)

Schedules a DTS event to pause the DTS at the simulation time REALTIME. The CBSIM rule base determines when to call this routine and the parameter REALTIME.

RESTARTDTS

This routine is called to continue the DTS as soon as the response of circuit breakers is determined.

GETSIMTIME(SIMTIME, STATUS)

This routine is called to obtain the DTS simulation time and to check the status of the DTS to see if it is paused or running.

MSLEEP(TIME)

This routine is called for the CBSIM to sleep for a duration of "TIME" and wake up automatically. This is necessary for synchronization of the system. For example, CBSIM must wait for the DTS to be paused at a circuit breaker's reclosure time.

The rule base of CBSIM determines when and which routine is called.

3.3.2.4 PROCESS COORDINATION

Coordination between the DTS, CRAFT and CBSIM is necessary for this experimental system to work properly. In an earlier version of CRAFT-DTS system, the CBSIM rule base was combined with the CRAFT rule base. Thus the coordination was between DTS and CRAFT. In the version discussed here, the CBSIM and CRAFT rule bases are separate. As a result, there are three processes in CRAFT-DTS: DTS, CBSIM and CRAFT. Figure 3-8 shows these processes and the flow of control. Data paths are not shown on the diagram for simplicity.

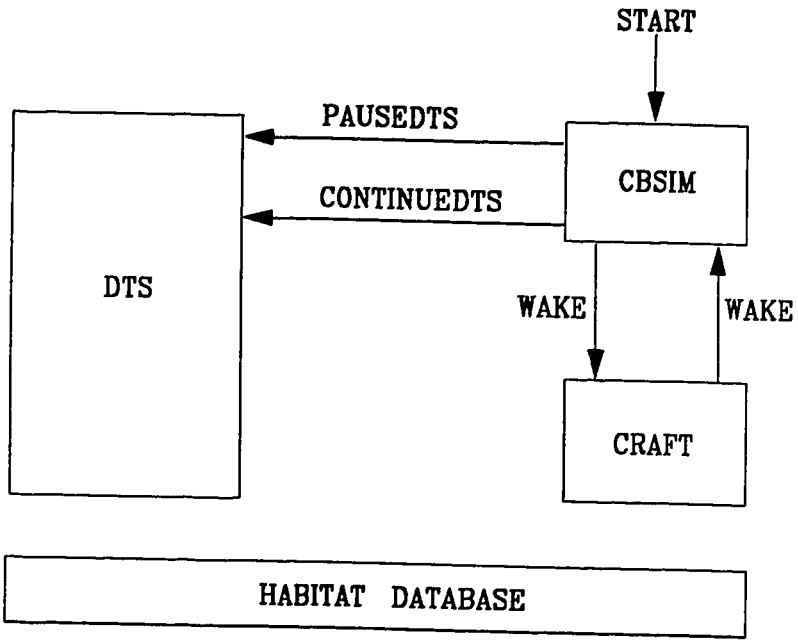


Figure 3-8 The Control flow in the CRAFT-DTS system

Since the DTS is to provide a realistic EMS center environment, it should run all the time, even when the dispatcher and/or CRAFT is solving problems. However, the DTS simulation should be slowed down or paused during the CBSIM simulation of the circuit breaker reactions to fault currents. In an actual power system, the circuit breaker response takes essentially no time (only a few cycles). To duplicate this fast process, it is necessary to pause the DTS simulation clock during CBSIM operation. In other words, we do not want the DTS to drift along while CBSIM is determining the circuit breaker response to a line fault. As soon as the circuit breaker response is determined, the CBSIM issues a command to continue the DTS. The schedule for the DTS to be paused and continued is determined in the rule base of CBSIM.

CBSIM also plays the role of line fault alarm monitor. It determines which alarms occur and activates CRAFT. Then CBSIM waits until called upon to simulate a circuit breaker test. When CRAFT wants to test a circuit breaker, it passes the test data to the database and activates CBSIM. Then CRAFT hibernates. When the test results are available,

CBSIM wakes up CRAFT to get the results and to continue its reasoning process. CBSIM then hibernates. The operation of CRAFT and CBSIM is mutually exclusive.

Overall control of these processes is necessary if this system is to run continuously. A deadlock condition could result if both CRAFT and CBSIM were waiting for each other, for example, if CBSIM was waiting for a circuit breaker test request and CRAFT was waiting for new device alarms. This problem is handled in the integrated system by sending a special test CB message from CRAFT to CBSIM when CRAFT reads no more alarms. CBSIM is activated upon receiving this message and exits. CBSIM can then be started manually as indicated in Figure 3-8.

3.4 TESTING AND RESULTS

Two stages of testing have been performed. In the first stage, an existing ESCA 40-bus system was used with single lines involving automatic switches inserted. In the second stage, the CRAFT-DTS system was tested on a subsystem model of the Puget Power system.

Figure 2-6 in Chapter 2 shows a typical transmission line in the Puget Power system, one of the most complex lines having many taps and switches. This line was inserted into the ESCA 40-bus system. It is also contained in the Puget subsystem modeled on the DTS.

3.4.1 STAGE 1 OF TESTING, AN ESCA 40-BUS SYSTEM

In separate experiments we have tested the CRAFT-DTS system on three lines which represent simple as well as the most complicated line configurations in the Puget Power network. The system has been tested for all fault locations on each of these lines and both CRAFT and the DTS responded properly. The three lines tested are CottageBrook_Sammamish (Figure 2-5) in Chapter 2, TalbotHill_Asbury (Figure C-1 in Appendix C) and SouthBremerton_Midway (Figure 2-6). In each test, one of these lines was inserted into the ESCA 40 bus model as shown in Figure 3-5.

Tables 3-2, 3-3 and Figure 3-9 present the results for the tests on the SouthBremerton_Midway line, the most complex of the test lines. Table 3-2 shows that

even for this line the total CPU time for CRAFT and CBSIM is very reasonable. This CPU time accounts only for the time used by the expert system and not for the DTS time.

Table 3-2 Results of testing on a Puget Power line

Test when line fault between devices:		CPU time (s) CBSIM+CRAFT
157	544	43.80
964	75	43.46
74	889	37.04
47	986	29.75
985	66	23.27
35	679	10.70
678	712	9.91
712	530	9.77

Figure 3-9 illustrates the interaction of CRAFT and CBSIM with the DTS. The activity shown is for the test of a fault between devices 47 and 986 on the SouthBremerton_Midway line. That is, the test corresponds to row 4 in Table 3-2. Total elapsed time to solve the problem is 160 seconds. This elapsed time represents "immediate" response by the dispatcher to the suggestions supplied by CRAFT. This 160 second interval to solve the problem is considered very reasonable by the Puget Power dispatchers. Note that the CPU time for CBSIM and CRAFT is about 30 seconds (Table 3-2). This CPU time is used for 3 CRAFT and 4 CBSIM executions as is seen from Figure 3-9. If the CPU time for CBSIM is excluded then the net CPU time for CRAFT executions, including database accesses, is about same order as the average CPU time (25.21 seconds for 3 executions of CRAFT) for the appended approach [43]. This equivalence of CPU time for CRAFT analysis is expected. However, the database access time may be different since the appended approach uses a RIM database and the other uses the extended HABITAT database.

Consistent with the concept that the DTS simulates the power system, we want it to be running all the time. From the two left hand columns in Figure 3-9 we see that this is the case except for the two pauses at t1 and t2. These pauses represent the time required for CBSIM to retrieve necessary data from the database, determine whether the reclosures of circuit breaker 157 were successful and send the open or close commands to the DTS. The pauses are very short (between 0.25 and 1.0 second) and guarantee that the DTS simulation remains "synchronized" with CBSIM. The "operation" and "activity" columns show the portion of time that CRAFT and CBSIM are running in parallel with the DTS. The "activity" column also shows what activity is occurring in each interval, "A" (analysis) indicates a CRAFT activity and "S" (simulation) indicates CBSIM activity.

It appears that there is little impact of the CRAFT system on the speed of the DTS operation suggesting that an EMS system may be able to tolerate CRAFT or other expert system applications without it becoming a burden on the EMS computer. Much more study of this problem is required before we can be very precise about this issue.

Table 3-3 indicates the CPU time required by some of the operations of the FORTRAN interface (Figure 3-6 and 3-7) when retrieving data for the SouthBremerton_Midway line. Tests on the initial implementation indicated that database access was requiring too much time and slowing the system response to unaccepted level. To illustrate the solution of the response time problem, refer to Table 3.3.

The first two rows of table 3-3 refer to specific routines, GETDTSCB/UPDATECB and PUTCB/CBEVENT, which are called frequently when managing the circuit breakers. The first column indicates the time required for a database retrieval operation when the database is opened (accessed) each time a retrieval is required. The second column shows the incremental time for a retrieval when the database remains open after the prior retrieval. Hence, considerable improvement was achieved by simply opening the database once and keeping it open for all retrievals. The third column shows the reduction in the incremental retrieval time when subscripts or pointers were used. The third row illustrates the cumulative effects of these different approaches. As indicated, retrieving the data for 15 circuit breakers required more than 7 seconds if the database had to be opened each time and considerably less if it remained open.

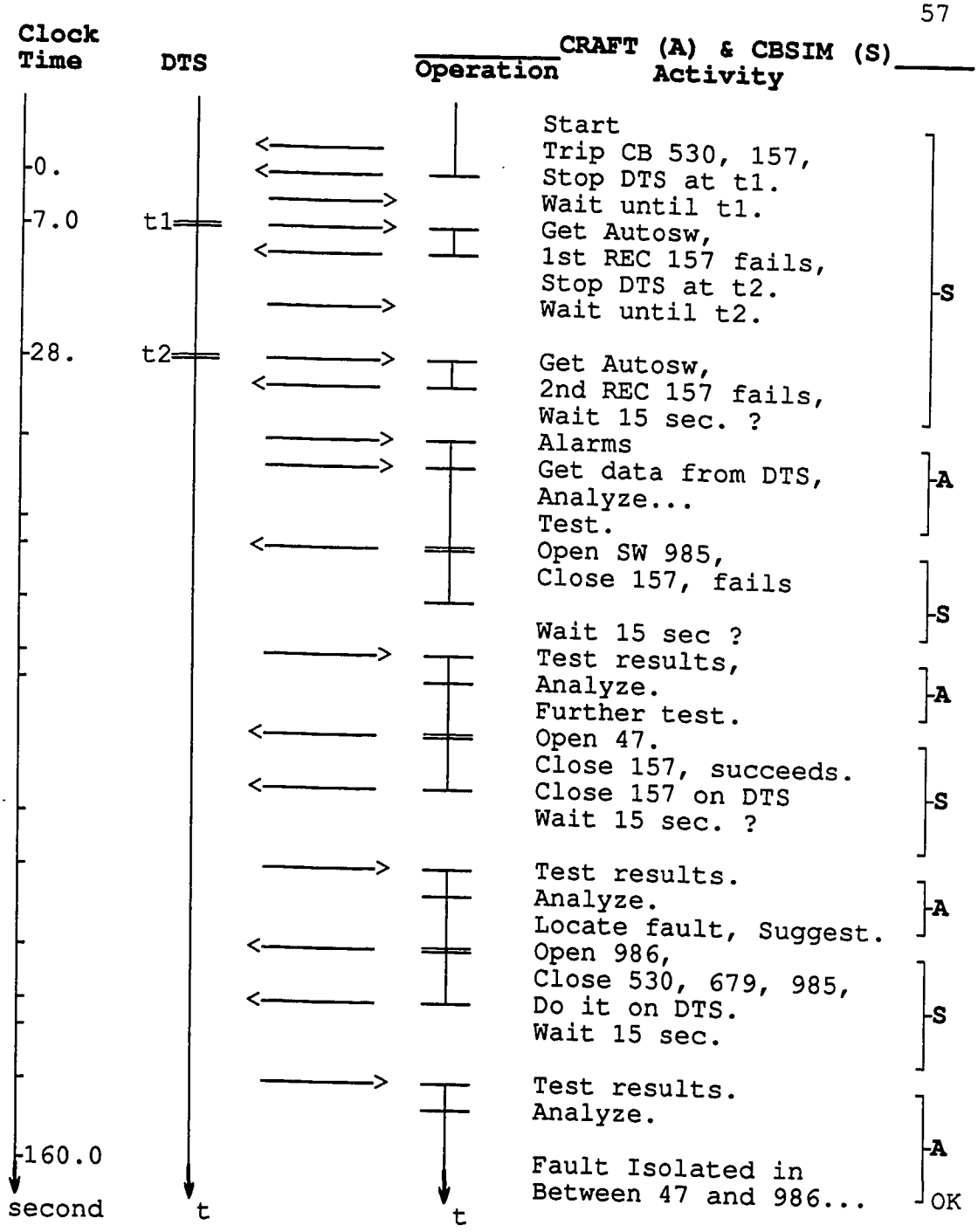


Figure 3-9 Sample Timing Diagram of the CRAFT-DTS System Operation

Line tested: SouthBremerton_Midway Fault location: 47-986

Note: pause duration is 0.33 sec. at t1 and 0.40 sec. at t2.

Table 3-3 Comparison of DB accessing time for different types of calls

Transmission line tested: SouthBremerton_Midway

Items to measure	CPU time (s)		
	with DB access overhead, No subsc.	without DB access overhead	
		No subsc.	subscript
call GETDTSCB/ UPDATECB	0.46	0.065	0.045
call PUTCB/ CBEVENT	0.50	0.11	0.050
transfer 15 CBs data into WM	7.26	1.21	0.98

Database access is a bottleneck in improving the efficiency of the CRAFT-DTS system. Efficient database access can be achieved by keeping all relevant databases open during the CRAFT solution process and by using pointers.

3.4.2 STAGE 2 OF TESTING, A Puget Power SUBSYSTEM MODEL

To give dispatchers a more realistic experience, a multi-line model of a portion of the Puget Power system has been built on the DTS. Each line in the model has automatic switches and corresponds to a line in the Puget Power system. This allows Puget Power dispatchers to experiment with the CRAFT-DTS system using a power system model that they are familiar with.

This Puget subsystem, as shown in Figure 3-10, consists of 6 substations, 8 lines and 4 generating units. Along each line, there are load transformer taps which are not shown on the diagram. This portion of the Puget Power system was modified at the boundaries so it would function as an independent system. Based on the full system power flow, a generation unit was added to a boundary bus of this subsystem if there was a net power flow into the bus from the outside area. An extra load was added at a boundary bus if the

net power flow was from the bus to the outside area. The effect of these added injections was to maintain line flows and bus voltages identical to those found on the full Puget Power system.

Extensive simulation was performed on this system to test the capability of the CRAFT-DTS system. The results show that the CRAFT-DTS system works satisfactorily. The DTS runs continuously except for several interruptions of very short duration for the simulation of circuit breaker responses. The impact on the DTS performance of the added expert system software is minimal. For the most complex two terminal line, SouthBremerton_Midway, the CRAFT solution time measured is about the same order as that tested on the ESCA 40-bus system. This is expected because the line fault problem is really localized to the individual line. The size of the DTS database is smaller than that of the ESCA 40-bus system while the size of the CRAFT database is only slightly different. The only common fields between the CRAFT database and the system database are those containing the status of the switching devices which are accessed by CRAFT using pointers. The use of pointers speeds up the database access.

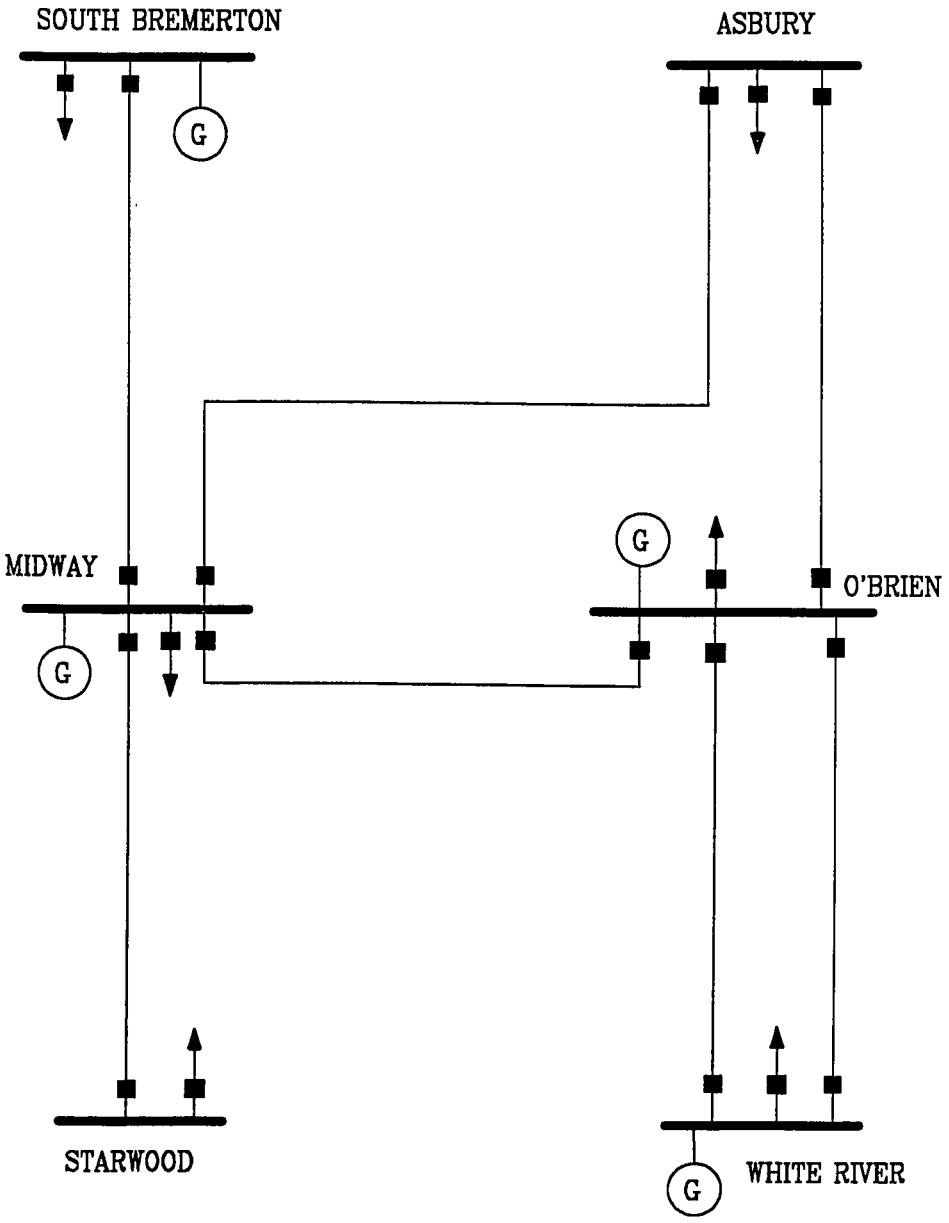


Figure 3-10 A Portion of the Puget Power system

3.5 REMARKS

This chapter presents the database integration approach for incorporating an expert system into the EMS center environment. The ESCA DTS and its HABITAT system are used for simulating the EMS environment and the power system. The EMS user interface manager can be used to build customized displays for CRAFT. In order to perform the integration, the DTS functionality was modified and extended to simulate the line fault effects and to model automatic switches. The DTS database was extended and interface routines were developed to support the CRAFT-DTS system.

The CRAFT system has been successfully integrated with the DTS using the database integration approach. This is a first step in demonstrating how such an expert system tool can be integrated into a control center on the EMS computer. The initial experiments indicate that CRAFT does not degrade DTS performance so we do not expect it to degrade the EMS system performance. This study provides a basic approach to enhancing EMS center functionality with expert systems and other AI tools.

This system is ready for dispatcher assessment. The power system model could be expanded if the dispatcher response is favorable. If this system evolves into a training tool, it should be extended to represent the entire Puget Power system of about 240 substation buses and 65 lines.

In the CRAFT-DTS system, CRAFT, CBSIM and the DTS are separate computational processes on the same computer. At present, these processes are organized in a way that the CBSIM schedules the pause and restart events of the DTS. CBSIM also determines when to activate CRAFT. When CRAFT is running, it may request CBSIM to manage circuit breaker test actions. Without modifying the DTS code, the DTS itself can not turn on CRAFT or CBSIM. For the control purposes, CRAFT and CBSIM still need two separate terminals for the user intervention. The full integration approach presented in Chapter 4 solves the coordination problem and improves the man-machine interface.

CHAPTER 4

FULL INTEGRATION APPROACH

The full integration (FI) approach features a single computer, a unified database, unified displays and a unified process control. In this approach, the expert system becomes an integral part of the EMS software and interacts naturally with the rest of the EMS modules. The process coordination problem that was present in the database integration approach is solved by the FI approach.

A process scheduler exists in an EMS center to control all of the EMS activities which are commonly written in conventional procedural languages such as FORTRAN. Since an expert system is different from the conventional procedural programs, this study was concerned with the basic questions:

1) Can the expert system be put under the control of this central process scheduler and how can this be done?

2) Does the full integration result in better performance than the database integration approach due to a unified control, displays and 'natural' interaction between the ES and the rest of EMS software?

The full integration study answers these questions. The ESCA DTS is used to simulate an EMS center environment. The expert system used in this integration work is the voltage control expert system (VCES).

VCES has been successfully integrated into the DTS software environment. To do so requires integration of both the data and the process control.

Similar to the DBI approach, certain technical issues related to modeling and integration had to be solved to realize the full VCES-DTS(EMS) system. These issues were:

1) Power system simulation
2) Database extension
3) Interface development
4) Control integration
5) Sensitivity computation
6) Man-machine interface integration

Some of these issues, such as items 1, 2, 3 and 6, are similar to those in the database integration work. For completeness, the solutions to all these issues are presented in the technical issues section. However, to understand the concept of full integration, it is necessary to understand the DTS process manager, PROCMAN, which is discussed in Section 4.1. Section 4.2 gives a general description of the VCES-DTS system. Testing and results are in Section 4.4

4.1 UNDERSTANDING THE DTS PROCESS MANAGER

It is important to understand the DTS process manager (PROCMAN) because it is the major challenge in achieving full integration. PROCMAN is a HABITAT utility for software system integration. It consists of a database, a user-callable interface and several tasks to enable users to establish and verify the PROCMAN database and to initialize and run the PROCMAN controlled system. The function of PROCMAN is to manage the activities of a group of interrelated VMS processes. PROCMAN must start up, sequence and schedule all processes. The use of PROCMAN within a system can be as simple as starting up all of the processes on one CPU, or as complex as sequencing the execution of many tasks on several machines.

A large real-time software system would be much easier to develop and maintain if one could pull all scheduling and synchronization information out of the code and put it into a database. The implementation of this concept is PROCMAN and its database. The database contains the control information of the processes to be coordinated. This concept facilitates the integration of a software system since changing the schedule of tasks or coordination among tasks can be accomplished by modifying the database. There

is no need to change the code of one module if changes are made to another. Hence, PROCMAN is database driven. The status of the database determines which process or program to run next. The system architecture is similar to the blackboard control architecture [39].

To emphasis the role played by the process manager, the basic structure of the DTS of Figure 4-1 is recalled here with the PROCMAN database indicated and the control paths added with broken lines. The solid paths on the diagram indicate data flow and the dotted paths indicate the sequence of execution of the separate blocks which is controlled by the process manager. Based on the status of each execution and of the PROCMAN database, PROCMAN determines which functional block will execute next. Note that, in the DTS, PROCMAN manages the activities in both the power system model and the EMS model. This PROCMAN is not an artifact. In an actual EMS center that uses HABITAT, PROCMAN exists for the EMS activities only. Hence, the integration of an expert system with the DTS PROCMAN is a prototype for integration with an actual EMS PROCMAN.

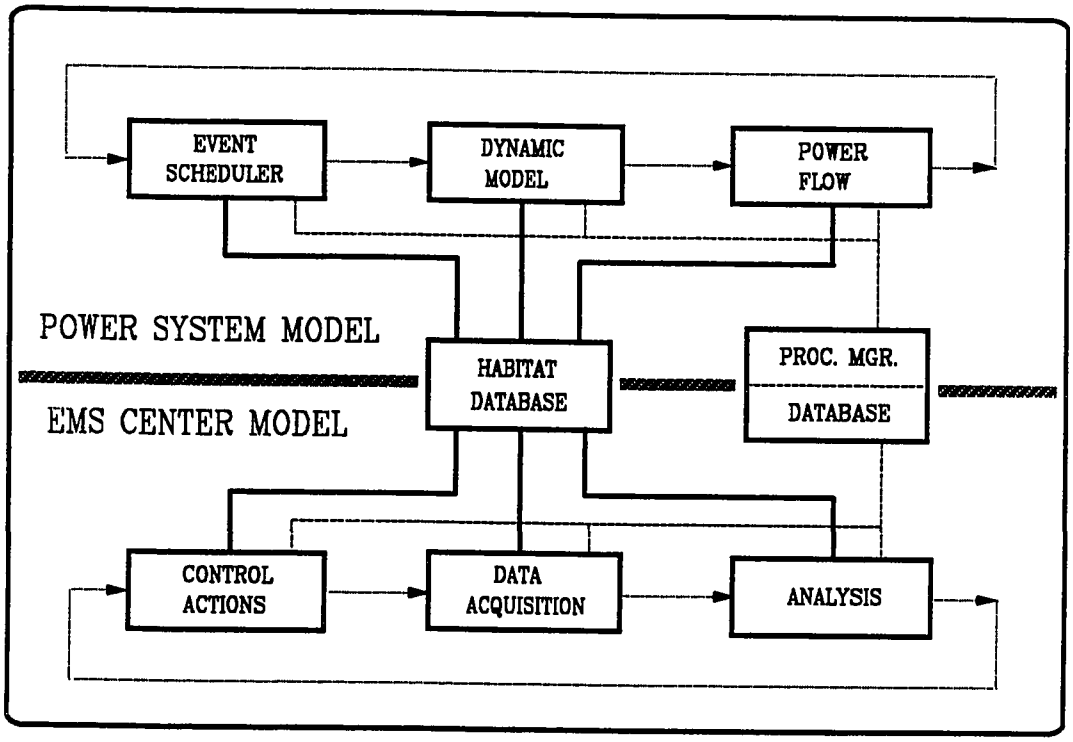


Figure 4-1 Process Manager in the DTS

Usually, the design of an integral software system starts with a control map which consists of task nodes, directional paths, condition flags, important attributes and labels. The PROCMAN database is a structured data representation of this control map. A simple control map is given in Figure 4-2 below to illustrate the basic concept of PROCMAN and the graphical representation of the database. For the detailed data structure of the PROCMAN database, the reader should refer to the PROCMAN Reference Manual [38]. There are three nodes (labeled by task1, task2, task3) and two paths on the map in Figure 4-2. A node represents the entrance point of a task. A node can be scheduled by assigning a schedule to it and/or by scheduling the path to the node. Two paths labeled task2 and task3 exit node 1. The label (1) or (2) of a path determines the sequence for that path to be evaluated by PROCMAN. This sequence is defined when the database is built. Thus, at node 1, path (1) should be evaluated first. Paths can be conditioned as well as scheduled . Both paths in Figure 4-2 are conditioned as is indicated by the condition flag near the path. When the control is at node 1, task1 is carried out. Upon completion of task1, path (1) is evaluated first. If the condition flag 1 is true, then path (1) is turned on and control is passed onto node 2. Note that path (2) has an attribute 'P' which stands for 'parallel'. Parallel paths can be executed simultaneously if conditions on both paths are true. Hence, path (2) will be evaluated regardless of the sequence of the path. If the condition flag 2 is true, then the path is executed and control passes onto node 3. If path (2) is not a parallel path and if path (1) is turned on, then no evaluation will be performed on path (2).

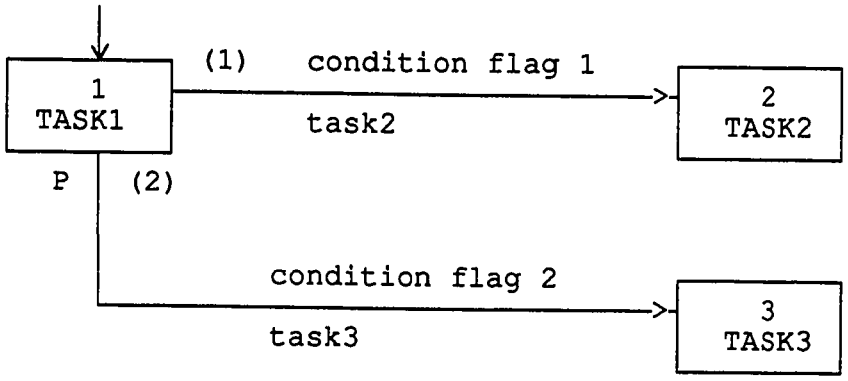


Figure 4-2 A portion of the control map

To place a new application under the control of PROCMAN, the major steps that must be accomplished are:

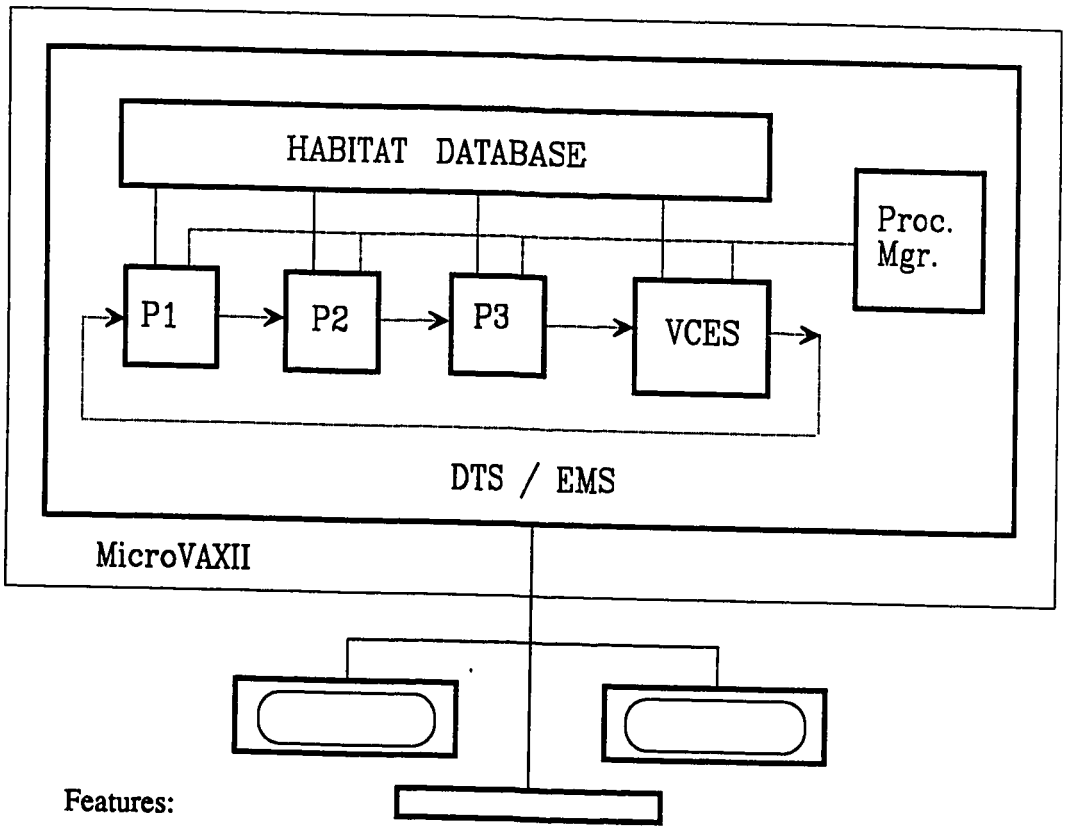
1) Identify the necessary control information (e.g. sequence, schedule, cause, conditions, interactions).

2) Design and modify the PROCMAN control map (e.g. insert new nodes, paths, condition flags, schedules, labels).

3) Implement the design by modifying the PROCMAN database.

4) Use the user callable PROCMAN routines to interface the new application with PROCMAN.

5) Test



Features:
Single Computer
Unified Database
Unified Process Coordination
Unified Displays

Figure 4-3 The full integration of an ES into EMS centers

4.2 VCES-DTS SYSTEM

In the VCES-DTS system as shown in Figure 4-3, the VCES is integrated with the DTS database and displays as well as with the process manager. The P1, P2 and P3 blocks represent DTS or EMS software modules. As the figure suggests, the VCES becomes simply another of the DTS/EMS software modules. Communication among these modules is through the HABITAT database. All these modules are run under the unified control of the process manager. As a result, the VCES interacts naturally with the other software modules. The user can monitor and interact with the system via the standard console displays, keyboard and other cursor moving devices.

4.3 TECHNICAL ISSUES FOR FULL INTEGRATION

4.3.1 POWER SYSTEM SIMULATION

To make the results of the full integration effort comparable with the simulation results on the stand-alone system, the same IEEE 30-bus system has been modeled in the DTS. The step up transformer model is used for the generation unit to couple the power to the network. The load tap changer (LTC) is physically modeled by the standard LTC type step down transformer. Since the primary and secondary sides of the transformer are treated as different buses in the DTS power flow program, an increase in bus number in the DTS model is expected. Thus, the 10 transformers in the system, 6 step up transformers for the generation units and 4 LTC step down transformers, create 10 additional buses. As a result, the original 30 bus system becomes a 40 bus system when modeled in the DTS.

The control outputs from the VCES to the power system are in the form of scheduled DTS events which the dispatchers are familiar with. The events include: the switching event to turn on or off a capacitor bank, the transformer event to move a tap position and the generation event to schedule its generation voltage.

4.3.2 DATABASE EXTENSION

The basic means for interfacing the VCES with the DTS is through the database. As with the CRAFT-DTS system, the DTS database is extended to support the VCES. This extended portion is called the VCES database. It consists of 4 records:

CONTRL
NCTRL
BUSES
SENS

These records are basically relational. Each record is defined by a number of attributes given below. The record names and attribute names are in upper case with the definitions in lower case in parentheses.

CONTRL:

BS_CONTRL	(the pointer to the bus in DTS database)
CLASS_CONTRL	(the type of the control, e.g. tap)
STATUS_CONTRL	(the control is energized or not)
SETTING_CONTRL	(the current setting of the control)
STEPS_CONTRL	(the total number of the discrete steps)
UPPER_CONTRL	(the upper limit of the control)
LOWER_CONTRL	(the lower limit of the control)

NCTRL:

NTAP_NCTRL	(the total number of the load tap changers)
NGEN_NCTRL	(the number of the active generation units)
NRC_NCTRL	(the number of the capacitor banks)

BUS:

BS_BUS	(the pointer to the bus in DTS database)
TYPE_BUS	(the type of the bus, PV or PQ bus)
VOLTAGE_BUS	(the voltage magnitude of the bus)
UPPER_BUS	(the upper limit of the bus voltage)
LOWER_BUS	(the lower limit of the bus voltage)
STATUS_BUS	(the bus is energized or not)
Q_DEMAND_BUS	(the reactive power demand at the bus)
P$SENS_BUS	(the hierarchy pointer to record SENS)

SENS:

P$BUS_SENS	(the parent pointer to record BUS)
I$CONTRL_SENS	(the pointer to record CONTRL)
SENSITIVITY_SENS	(the sensitivity of the BUS voltage to the CONTRL)

The data structures in the VCES database are organized to ease the mapping of the data to the corresponding data items in the VCES working memory. Among these data items, SENSITIVITY_SENS is the only new data which is not directly available from the DTS database. Some data, such as bus type and status, are available but are scattered in the DTS database. Some data, such as the upper and lower limits, are also available but need to be transformed to fit the scale predefined in the VCES rule base. Pointers are used in the VCES database to establish links to the existing data in the DTS database. The use of pointers avoids duplication of data and eases data maintenance.

4.3.3 INTERFACE DEVELOPMENT

A data interface is necessary to transfer data between the VCES and the database. This interface consists of FORTRAN routines and OPS83 translators. The dashed block in Figure 4-4 contains the FORTRAN routines. Based on their function, these routines can be classified into four types:

1) To prepare the VCES database, e.g., VCES_DB_INIT.
2) To perform computation based on the DTS database and send the results to the VCES database, e.g., COMPUTE_SENS_MATRIX.
3) To get data from database, e.g., GET_BUSES.
4) To send control actions to the event database, e.g., RC_EVENT, TAP_EVENT.

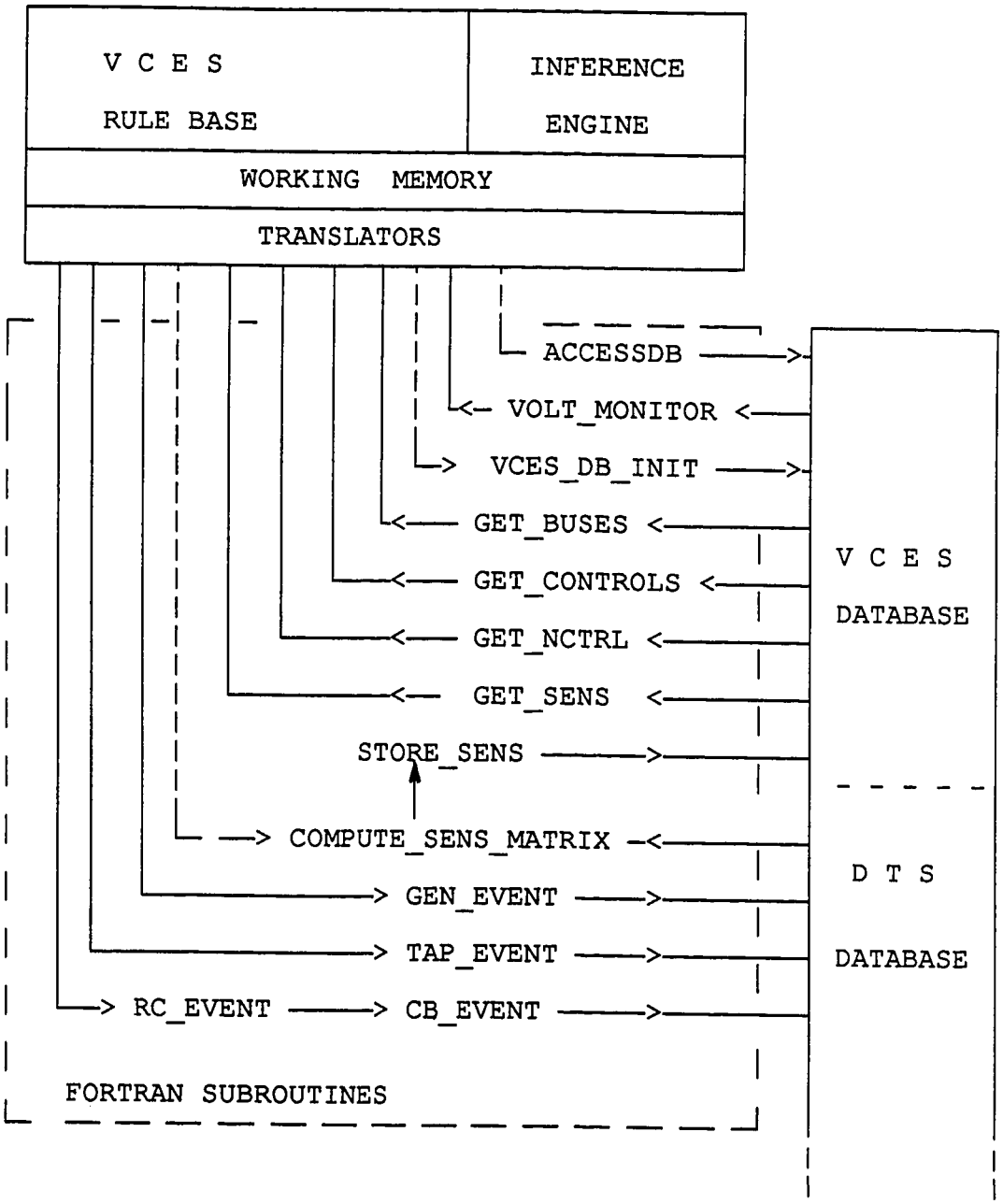


Figure 4-4 Interface routines and translators

For types 1 and 2, no data is actually passed between the VCES and the database. Hence, no translator is needed. For types 3 and 4, the translators on the VCES' side are necessary for correctly mapping the symbolic data type in OPS83 to the character string type in FORTRAN.

Appendix B gives examples of data mapping and of how the translators work.

A list of the interface routines with a brief explanation of their functions follows:

ACCESSDB

This subroutine opens all necessary databases and partitions for the rest of the data interface routines. It is called at the very beginning of the main program.

VOLT_MONITOR (&violation)

This subroutine accesses the EMS database to monitor the bus voltage. The parameter &violation gives the total number of troubled load buses. If any voltage violation is detected then the VCES continues. Otherwise, it quits.

VCES_DB_INIT

This subroutine is called when a voltage problem is detected. The routine initializes the VCES database. It accesses the DTS database to gather the necessary data and puts it into the VCES database. Then, the COMPUTE_SENS_MATRIX routine is called.

COMPUTE_SENS_MATRIX

The sensitivity of bus voltages to three types of controls, the generation bus voltage, the tap changer and the capacitor bank, is computed in this routine. It accesses the DTS database, performs calculations and stores the results into the VCES database.

procedure get_system_state();

This OPS83 procedure is called at the beginning of the execution of the VCES rule base to retrieve all data necessary to solve the problem. In this procedure, the four FORTRAN subroutines are called to get data from the database:

GET_BUSES (bus#, type, volt_array, status, qd, flag)

The volt_array contains bus voltage and its upper and lower limits. The array data type is used to contain all the data items to be passed because OPS83 allows at most 6 parameters in external procedures.

GET_NCTRL(ngen, ntap, ncap)

This routine gets the total number of active generating units, taps and capacitor banks in the system.

GET_CONTRLS (id_array, class, status, set_array, step, flag)

The id_array contains control number and bus number.

The set_array contains the control setting and its upper and lower limits.

GET_SENS (bus#, ctrl#, sensitivity)

This subroutine gets the sensitivity matrix.

RC_EVENT (location, setting)

Based on the location and setting specified, this routine determines which capacitor should be used and which CB should close. It then issues a CB event request:

REQUEST CHANGE (cbopen, time, subscript_CB, 'evcl')

TAP_EVENT (location, setting)

Based on the location and setting specified, this routine determines which transformer tap should move to a new position. It then issues a transformer tap event request:

REQUEST CHANGE (xftap, time, subscript_tap, setting)

GEN_EVENT (location, setting)

Based on the location and setting specified, this routine determines which generator unit's regulation schedule should change to a new setting. It then schedules the maximum and minimum target regulation events:

REQUEST CHANGE (regskmn, time, subscript_regsk, setting)

REQUEST CHANGE (regskmx, time, subscript_regsk, setting)

Parameter 'regskmn' indicates the minimum regulation schedule and 'regskmx' indicates the maximum regulation schedule at the time specified by the parameter 'time'. The 'subscript_regsk' is a pointer pointing to a regulation schedule in the database.

4.3.4 CONTROL INTEGRATION

The focus of the full integration approach is to put the VCES under the control of the process manager. As stated previously, Figure 4-1 and 4-3 illustrate the basic concept. The implementation involves modifying the PROCMAN database and using the PROCMAN interface routines. To proceed, it is necessary to identify the control information, i.e. when and under what conditions the VCES is activated or is on standby. Then a graphical representation of the control logic, the control map, is used to design the control integration. The PROCMAN database is a structured data representation of this control map which is the key to this integration.

Figure 4-5 shows a portion of the DTS control map with the new nodes, new paths and condition flag inserted. The new nodes are the VCES MONITOR, VCES and the COPY DATA. The new paths are as shown between the SCANNER and the MONITOR and between the MONITOR and the VCES. The new condition flag is the VCES-SOLVING which is used to condition the path to the COPYDATA node and the path to the VCES MONITOR node.

The upper part of Figure 4-5 represents the power system model. It contains a continuous simulation loop. There are three tasks in the model: Events, Dynamics, and Power flow. The node LoadEvents belongs to task Events. The nodes Dyn-loop, Eccess-f and PF-relays belong to task Dynamics and the nodes PF-prepare and PF-solve belong to task Power flow.

The task Events is responsible for loading events into the simulation event queue prior to the time when the event is scheduled to happen. Also it is responsible for the savecase events.

The node Dyn-loop is the dynamic loop node within the simulation loop. All prime mover dynamic models are included in this node.

At the node PF-prepare, power flow data is prepared. The topology processing is included in the node. Whenever there is a frequency trip or a PF-relays trip, the topology of the network must be reprocessed and power flow data prepared.

If no excessive load-generation mismatch (flag 'not excessf' true) is detected in the power flow input routine, then the path 31 will turn on. It automatically waits for execution if the PF-solve node is busy.

Control is passed to node Excess-f if the power flow data preparation routine detects an excessive load-generation mismatch. New island frequencies are calculated and frequency relay logic is executed to try to detect any frequency relay trips prior to solving the power flow. If a trip is found, then a path back to the power flow input node is taken. Otherwise, control is passed to PF-solve where the Powerflow is computed.

At the node PF-relays, using the state just calculated in the power flow, all voltage, and overcurrent relays are checked. There are 4 paths out of this node. PROCMAN evaluates the paths in the sequence defined in the database. The number beside each path is the sequence number of the path. Thus, path 26 is evaluated first. If a relay is tripped, then path 26 is turned on and control is passed back to the power flow input node. In this case, paths 27 and 28 will not be taken because they are not parallel paths. Though path 29 is a parallel path, it is not taken because the condition flag is false. The attribute 'parallel' means the path can run simultaneously if the imposed conditions are true. If no relay trip is detected, then path 27 is evaluated. If flag 'load-event' is true and neither a 'pause' nor a 'savecase' event is scheduled, then PROCMAN will take path 27 and load the next batch of events into the simulation event queue. If there are no events to load, then path 28 is taken. The attribute 'autowait' on path 28 means the path will wait automatically when the node 'Dyn-loop' is busy and will execute the node when it is available. In case either path 27 or 28 executes, path 29 will execute because it is a

parallel path and the condition 'no pftrip' is true. This parallel path models the fact that the SCADA system runs simultaneously with the power system.

The operation at node "COPY DATA" is to provide an updated copy of the network and power flow data for the sensitivity computation in the VCES. Upon the completion of the power flow solution, PROCMAN will perform this database copy operation if the VCES is NOT in solution process.

The lower part of the Figure 4-5 represents the EMS model. It consists of the SCADA data scanner, automatic generation control(AGC), the DTS control functions PAUSE-DTS and CONTINUE-DTS, command interpreter, SCADA man-machine interface (MMI), and the new functions VCES and MONITOR.

If no relay trip is detected at the PF-relays node, then the parallel path 29 will activate the SCANNER node where the power flow database is scanned. The analog values from the just completed power flow solution are transferred into the SCADA database. Thus, this node simulates the data acquisition part of the SCADA system. Most SCADA related events such as Analog/Digital conversion, limits checking and bad data filtering are within this node.

There are two paths that exit node SCANNER. One is to node AGC (the Automatic Generation Control). The other is to node MONITOR. At node AGC, the DTSAGC program is run to perform the EMS functions such as monitoring AGC, economic dispatch, scheduling generations and transactions. At node MONITOR, the system voltage profile is monitored.

The DTS operation controls and the EMS man-machine interface are represented by the four nodes: PAUSE-DTS, CONTINUE-DTS, SCADA MMI and COMMAND INTERPRET.

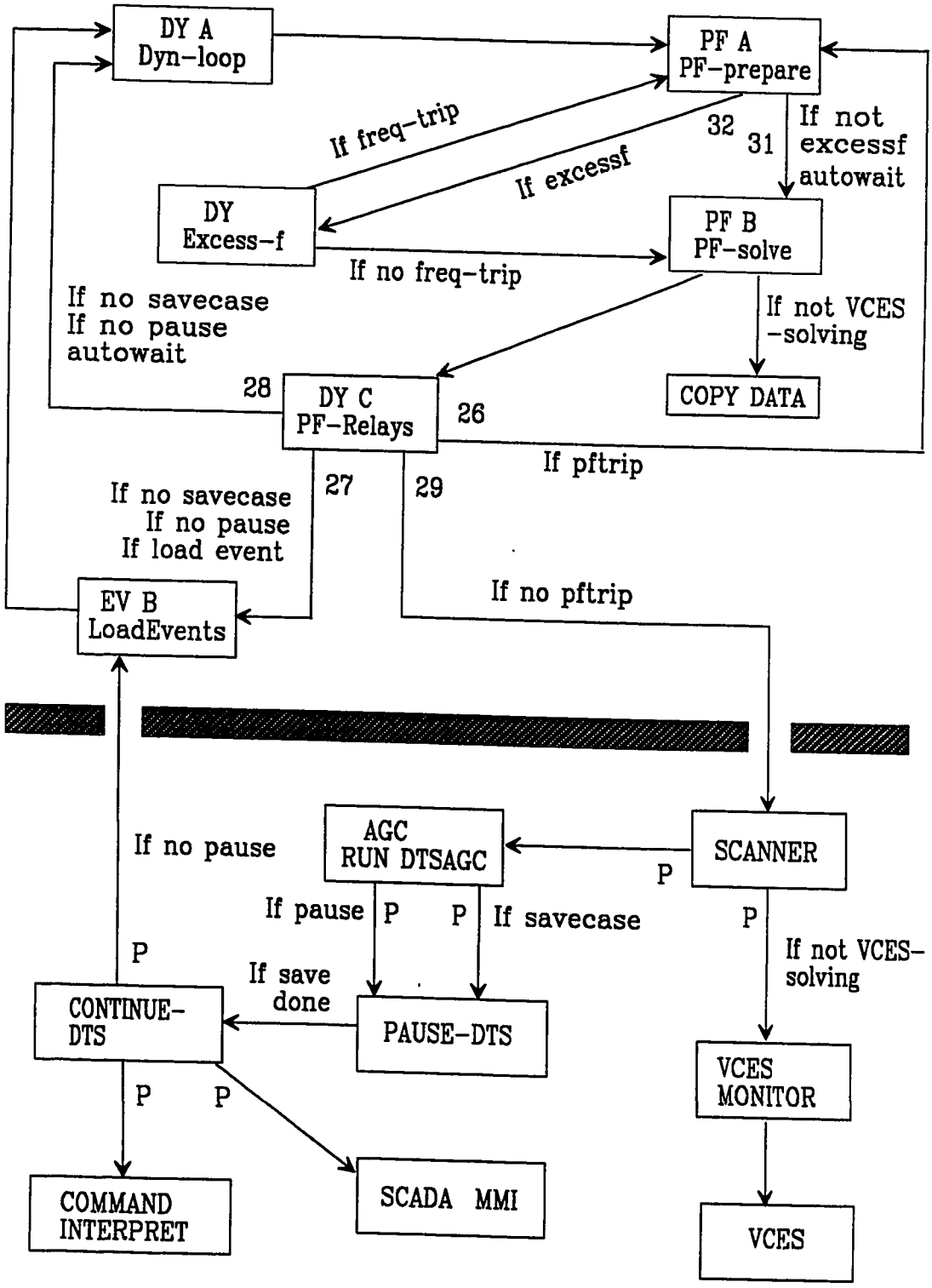


Figure 4-5 A portion of the DTS control map

Two parallel paths go from the AGC node to the PAUSE-DTS node. One is conditioned by 'if pause', another path is conditioned by 'if savecase'. If either of the conditions are true, then control would be passed onto PAUSE-DTS upon the completion of the AGC. The current status of the power system simulation and the data acquisition results will be saved for study and continuation. The path from PAUSE-DTS to CONTINUE-DTS is conditioned by 'if save done', which is to assure that the wanted Savecase is saved before the DTS continues. A manual start will reset the condition "pause" to false and continue running the DTS.

Three parallel paths that exit node CONTINUE-DTS are shown. One goes to node 'LoadEvents' to continue the power system simulation. The other goes to node 'SCADA MMI' to activate the SCADA man-machine interface functions. Another leads to node 'COMMAND INTERPRET' to start to accept and interpret the control commands for execution.

Figure 4-6 illustrates the operations at VCES MONITOR and VCES nodes under PROCMAN control. Logically, the VCES should run when the data acquisition task is completed. It has the same priority as the other EMS tasks such as AGC. Hence, the path to the monitor has the attribute of 'parallel'. This path is conditioned by 'IF NOT VCES-SOLVING'. In this implementation, it is assumed that no new voltage problem occurs during the VCES solution process. Thus, the path to the monitor is not executed whenever the VCES is solving a voltage problem. As indicated in Figure 4-6, the VCES is activated if a voltage problem is detected by the monitor. If no voltage problem exists, then the VCES returns to standby status and the condition flag "VCES-SOLVING" is reset to FALSE. Only in the standby status, i.e. NOT VCES-SOLVING, can the control path go through to the VCES.

Note that the flag VCES-SOLVING is set to TRUE immediately after the control enters the VCES. This flag is reset to FALSE after the solution is completed. This flag TRUE tells the PROCMAN to save the copy of the databases used by the VCES application. Then, the VCES continues to find which voltages are out of range and uses sensitivity analysis to find a set of controls that should solve the problem. Hence, a set of "control actions", in Figure 4-1 and Figure 4-6, is established.

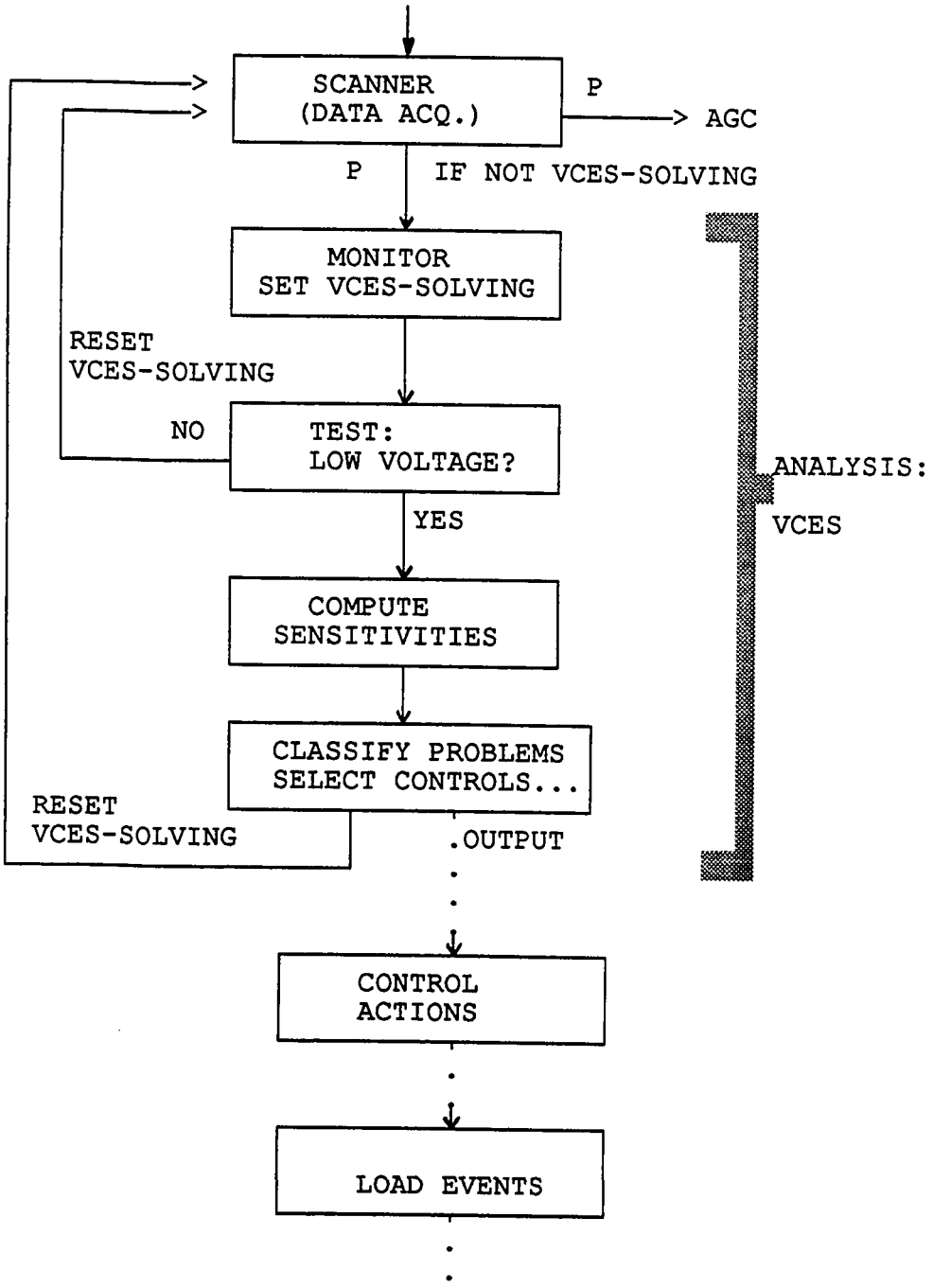


Figure 4-6 Additional Operations under the Procman control

The PROCMAN then implements the VCES solution at a time that "coordinates" with the power system simulation. In the case of the DTS, the control actions output from the VCES are sent to the event database. At node "LoadEvents", events are scheduled according to the specified simulation time and then loaded prior to the next simulation cycle. In other words, just as is the case when an operator is solving a problem, the power system simulation continues to run while the VCES is solving the problem.

When modifying the PROCMAN database, six PROCMAN_user routines [38] are used to interact with PROCMAN and its database. These FORTRAN routines (or functions) can be called from the VCES main program using the same approach as was used for the data interface in Section 4.3.4. The following four PROCMAN routines are used in this integration:

PROCMAN_DEFINE ('VCES')
PROCMAN_NODE (path, node)
PROCMAN_SETFLAG ('VCES_SOLVING')
PROCMAN_RESETFLAG ('VCES_SOLVING')

Appendix D discusses the use of theses routines and how to call them from the VCES main program.

4.3.5 SENSITIVITY COMPUTATION

Sensitivity information is essential to the VCES, as discussed in Section 2.1.2 and in [8], to select controls which will be most effective. Furthermore, the quantitative effects of each control are determined so that an ensemble of control actions to completely solve the problem can be estimated from the sensitivities. All this sensitivity information can be assembled from data already available from the DTS. Consequently, one significant part of the integration of the VCES is to extract the sensitivity information from the DTS power flow results and transfer it to the OPS83 code.

The sensitivity of V_i to C_j is defined as

$$\frac{dV_i / V_i}{dC_j / CN_j} \quad \text{or} \quad \frac{dV_i}{dC_j} * \frac{CN_j}{V_i} \qquad (4.3\text{-}1)$$

where V_i = voltage at bus i, i=1,..., N, N is the total bus number, C_j = the jth control, j=1,..., L, L is the total number of controls and CN_j = the normalization factor for control j. For generation buses, CN_j = the generation bus voltage. For transformer taps, CN_j = 1.0, the nominal kv (per-unit) of the tap. For the capacitor banks, CN_j = 1 per-unit base power. The quantity dV_i / V_i = the percentage of the voltage variation at bus i and dC_j / CN_j = the normalized variation of control j.

4.3.5.1 SENSITIVITY TO CAPACITOR BANKS

The sensitivity of a bus voltage to a capacitor bank is defined as:

$$\frac{dV_i / V_i}{dQ_{cj}} \qquad (4.3\text{-}2)$$

where V_i is the bus voltage at bus i, and Q_{cj} is the value of the capacitor bank at bus j. This sensitivity can be derived from the result of the system power flow computation.

The relationship between the state and control variables can be established from the power flow computation. This computation is based on the differential relationship:

$$[J]\, d\underline{X} = d\underline{Y} \qquad (4.3\text{-}3)$$

where $\underline{X}$ is the column array of the state variables, i.e. $\underline{V}$, the voltage magnitudes of the load buses and $\underline{a}$, all voltage angles. $\underline{Y}$ is the column array of the system control variables, i.e. the real and reactive power injections. [J] is the Jacobian matrix.

Switching on or off a capacitor bank is equivalent to a change in the reactive power injection at the bus where the bank is connected. Suppose the only reactive power injection change was an increment of dQ_{cj} at bus j. Then $d\underline{Y}$ can be expressed as:

$$d\underline{Y} = [0\ 0\ ..\ dQ_{cj}\ ..\ 0\ 0\ ]t$$

where the dQ_{cj} is in the position for bus j. Dividing both sides of Eq. (4.3-3) by dQ_{cj}, we obtain

$$[J]\, d\underline{X}/dQ_{cj} = d\underline{Y}/dQ_{cj} \qquad (4.3\text{-}4)$$

where $d\underline{Y}/dQ_{cj} = [0\ 0\ ..\ 1\ ..\ 0\ 0\ ]^t$. The solution of Eq. (4.3-4), $d\underline{X}/dQ_{cj}$, contains the sensitivity information of all the state variables to this reactive variation. For example, dV_i/dQ_{cj} is contained in $d\underline{X}/dQ_{cj}$. The sensitivity defined in Eq.(4.3-2) can be obtained by dividing dV_i / dQ_{cj} by V_i.

The sensitivity computation is straightforward for the capacitor bank. However, additional data manipulation is needed to obtain the sensitivities to the generation voltages and to the tap changers.

4.3.5.2 SENSITIVITY TO GENERATION VOLTAGES

The sensitivity of a bus voltage to a generation voltage is defined as:

$$\frac{dV_i / V_i}{dV_j / V_j} \qquad (4.3\text{-}5)$$

where V_i is the bus voltage, and V_j is the generation bus voltage.

The difficulty in obtaining this sensitivity is that V_j is fixed by the automatic voltage regulation (AVR) and dV_j=0. Hence, dV_j does not appear in Eq.(4.3-3). To make dV_j appear in Eq. (4.3-3), technically, the power flow computation is carried out with that generation unit AVR turned off but with the others unchanged. As a result, a new Eq. (4.3-3) with a new Jacobian is obtained.

Assume that the only reactive power injection to be changed was a 1 p.u. increase at bus j. Then $d\underline{Y}$ can be expressed as:

$$d\underline{Y} = [0\ 0\ ..\ 1\ ..\ 0\ 0\ ]^t$$

where 1 is in the position for bus j. The solution of Eq. (4.3-3), $d\underline{X}$, is an array of variations of all the state variables to this reactive control. Hence, it includes the vectors ($d\underline{V} / dy_j$) and ($d\underline{a} / dy_j$), and the scalar (dV_j / dy_j), so we can compute ($d\underline{V} / dV_j$) by the chain rule:

$$\frac{d\underline{V}}{dV_j} = \frac{d\underline{V} / dy_j}{dV_j / dy_j} \qquad (4.3\text{-}6)$$

For bus i, Eq.(4.3-6) gives (dV_i / dV_j), which, after multiplied by the normalization factor (dV_j / V_i), is the sensitivity defined in Eq. (4.3-5).

Hence, the procedure for computing the sensitivity is:

1. Set the generation unit AVR off, so its bus voltage V_j is variable.
2. Run the power flow to obtain a new Jacobian.
3. Set the reactive injection dy_j to 1 where bus j is the generation bus.
4. Solve Eq. (4.3-3) for the vector ($d\underline{V} / dy_j$).
5. Apply the chain rule of Eq. (4.3-5).
6. Normalize the result by a factor of (V_j / V_i).
7. Reset the AVR on.

4.3.5.3 SENSITIVITY TO TRANSFORMER TAPS

The voltage sensitivity to a tap is defined as

$$\frac{dV_i / V_i}{d(STEP)_j / 1.0} \qquad (4.3\text{-}7)$$

where V_i is the voltage at bus i, and $d(STEP)_j$ is the step size of the tap change at bus j. Note that the term $d(STEP)_j$ does not explicitly appear in Eq. (4.3-3). But the effect of the tap change is to change the reactive power injection at that bus. Based on this understanding, the key to the sensitivity computation is to determine the reactive power variation dy_j caused by a step change of the tap. The value of dy_j can then be put into the right-hand side of Eq. (4.3-3) and it can be solved for $d\underline{X}$ which contains $d\underline{V}$, the variation of load bus voltages to the tap change.

The procedure for computing the sensitivity is:

1) Based on the tap transformer model used, compute the variation dy_j of reactive power at bus j due to a unit step change of the tap. Appendix E gives the model of the two-winding tap transformer used on the DTS and the approach to the computation of this dy_j.
2) Put this dy_j into Eq. (4.3-3) and solve for $d\underline{X}$ which contains the voltage variation array $d\underline{V}$.
3) The sensitivity defined in (4.3-7) can be obtained by multiplying dV_i with a normalization factor $(1.0 / V_i)$.

4.3.6 MAN-MACHINE INTERFACE INTEGRATION

The operator interface with the VCES was made consistent with all other operator-DTS interactions. Using the HABITAT user interface manager, the system one-line diagram of Figure 4-7 was built to display the system operation. Important data, such as the line flows, bus voltages and status of circuit breakers were designed to appear on the display. Since the VCES is an integral part of the DTS, the control actions, initiated by the VCES such as adding capacitors, changing transformer taps and scheduling the generation voltages, are observable through the standard DTS displays. In other words, few new displays were necessary. However, where it was important, the HABITAT user interface manager and user interaction utilities enabled the user to easily build displays which were added to the regular DTS displays. Of course, those data items to be displayed must exist in the database because the HABITAT displays are database driven. For instance, the DTS message library was extended to include the messages created during VCES operation and a VCES message display was built for communication with the user. The operators can call up the new display using the standard commands. An advantage over the appended or database integration approaches is that all the operator interaction takes place through the standard console with no additional screens or keyboards.

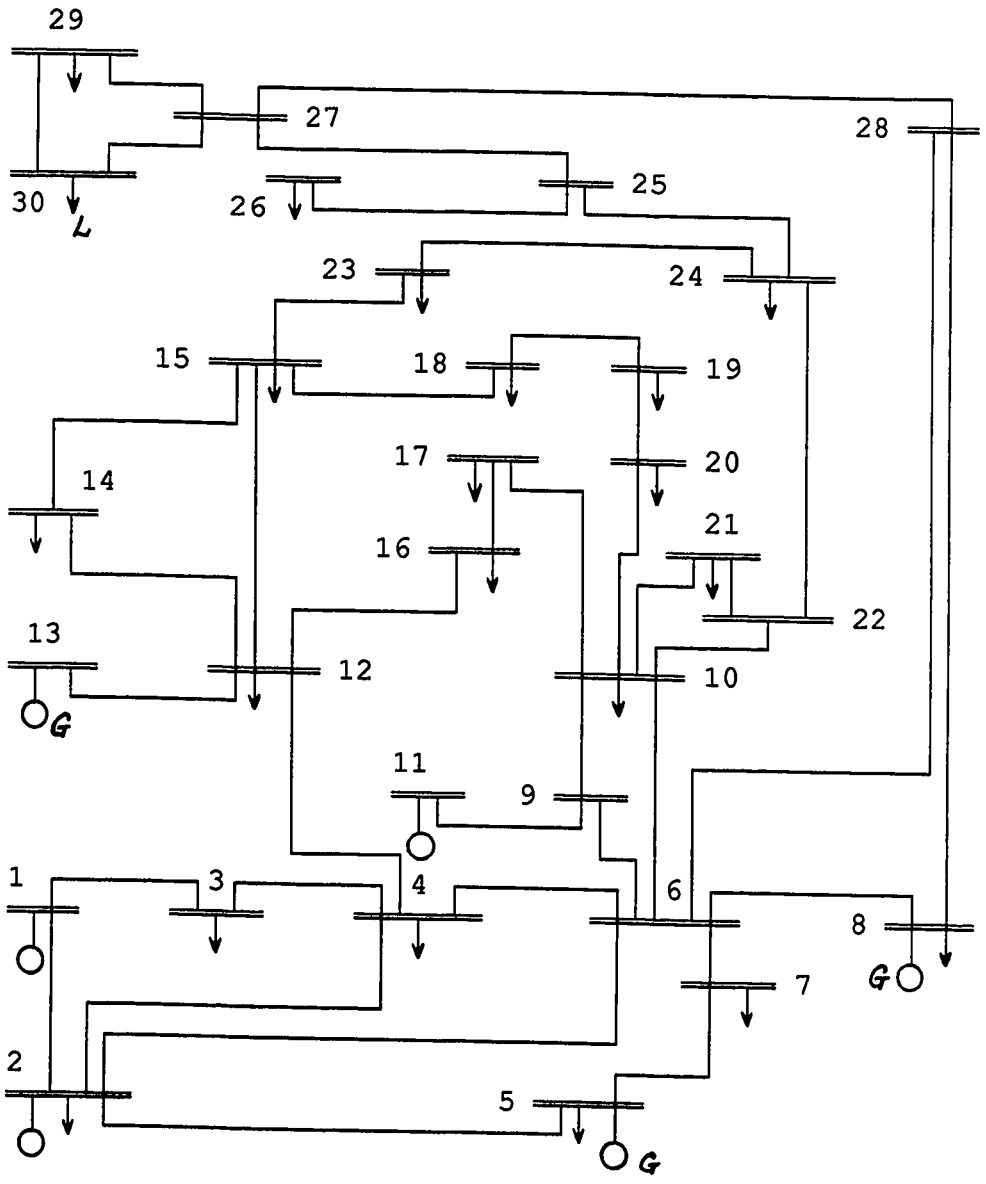


Figure 4-7 IEEE 30 bus system

4.4 TESTING AND RESULTS

Testing was performed on the IEEE 30-bus test system of Figure 4-7. The range of acceptable voltages is 0.95 to 1.05 p.u. The controls consist of voltage settings on the generators at buses 1, 2, 5, 8, 11 and 13, capacitor banks at buses 10, 19, 24 and 29 with maximum capacity of 10 MVar and tap changing transformers at buses 4, 6, 16 and 26 with a range of 0.90 to 1.10 p.u. The power flow solution of the basic system was the same as that of the stand-alone system. For the same contingencies, the resulting voltage problems were also the same. This was expected, because the same network configuration and same transmission model was used.

The VCES appears as one of the EMS functions. It is available whenever there is a voltage problem and provides the solution right away. The corrected voltage profile can be observed from a DTS console when the power flow routine updates the system state after the control actions. If one were watching the DTS displays from the time the disturbance occurs until the solution is implemented, the voltage would be seen to drop. Then the switches which brought the capacitor banks on line would be seen to close and the voltage would return to acceptable values. The user may want to call up the VCES message display to obtain more information about the operation of the VCES such as when it was activated, what buses had low voltages and the VCES proposed solution. The test results indicate that the integration was successful.

Two examples are given to demonstrate the system operation. The results agree with that from the stand-alone system. The first example was created by a change in load. The second example was created by a line outage and the loss of a generation unit. For each case, the pre-disturbance and the post-disturbance as well as the final voltage profiles obtained after the execution of the control actions are given. The VCES control actions are also listed.

Table 4-1 shows the bus voltages for example 1. Since the low voltage limit is 0.95 p.u., we see that buses 29 and 30 are low voltage violators as indicated by the underlined voltage values 0.908 and 0.912. The VCES solution is to change two capacitor banks at buses 29 and 24 to their maximum of 10 MVar. We see from the final column of Table 4-1 that, after implementing this solution, the voltage problem is corrected.

Table 4-2 shows the bus voltages for example 2. We see that 16 buses are out of the acceptable range as indicated by the underlined voltage values. The solution prescribed by the VCES is to change three capacitor banks, those at buses 19, 24 and 29. In all cases, these are to be moved to their maximum of 10 MVar. We see from the final column of table 4-2 that, after implementing this solution, all voltages are within acceptable bounds.

Example 1, Heavy loading at bus 29

A load of (0.05 + j 0.102) p.u. is added at bus 29. It causes two voltage violations at bus 29 and bus 30.

The VCES control actions: RC29 moved to 0.1
RC24 moved to 0.1

Table 4-1 Voltage profiles for Example 1

Bus No.	Bus Type	Voltages		
		Initial	Outage	Final
1	Gen.	1.060	1.060	1.060
2	Gen.	1.045	1.045	1.045
3	+	1.027	1.025	1.028
4		1.021	1.017	1.020
5	Gen.	1.010	1.010	1.010
6		1.013	1.010	1.021
7		1.001	1.003	1.005
8	Gen.	1.010	1.010	1.010
9		1.027	1.020	1.029
10		1.008	0.996	1.010
11	Gen.	1.080	1.080	1.080
12		1.025	1.024	1.033
13	Gen.	1.080	1.080	1.080
14		1.007	1.007	1.017
15		1.002	1.000	1.010
16		0.998	0.996	1.005
17		0.999	0.992	1.003
18		0.985	0.981	0.993
19		0.978	0.975	0.987
20		0.981	0.977	0.989
21		0.998	0.986	1.002
22		0.999	0.986	1.003
23		0.993	0.985	1.007
24		0.991	0.977	1.008
25		0.993	0.961	1.004
26		0.979	0.952	0.991
27		1.000	0.964	1.004
28		1.010	1.005	1.012
29		0.985	0.908	0.988
30		0.980	0.913	0.978

+ Buses with unspecified type are load buses.

Example 2, Line from bus 27 to bus 28 tripped, a generation unit at bus 13 tripped

The VCES control actions:

RC29 moved to 0.1
RC24 moved to 0.1
RC19 moved to 0.1

Table 4-2 Voltage profiles for Example 2

Bus No.	Bus Type	Voltages		
		Initial	Outage	Final
1	Gen.	1.060	1.060	1.060
2	Gen.	1.045	1.045	1.045
3	+	1.027	1.017	1.023
4		1.021	1.008	1.014
5	Gen.	1.010	1.010	1.010
6		1.013	1.006	1.011
7		1.001	0.996	0.999
8	Gen.	1.010	1.010	1.010
9		1.027	1.005	1.024
10		1.008	0.997	1.004
11	Gen.	1.080	1.080	1.080
12		1.025	0.945	0.980
13	*Gen.	1.080	0.945	0.980
14		1.007	0.929	0.969
15		1.002	0.925	0.971
16		0.998	0.933	0.970
17		0.999	0.950	0.988
18		0.985	0.919	0.972
19		0.978	0.920	0.977
20		0.981	0.927	0.980
21		0.998	0.950	0.995
22		0.999	0.949	0.996
23		0.993	0.916	0.974
24		0.991	0.914	0.998
25		0.993	0.870	0.982
26		0.979	0.855	0.968
27		1.000	0.852	0.987
28		1.010	1.008	1.012
29		0.985	0.835	1.002
30		0.980	0.826	0.981

\+ Buses with unspecified type are load buses.

*Gen. The bus type changes to load bus due to unit tripping.

CHAPTER 5

CONCLUSION:
COMPARISON AND EVALUATION OF THE THREE APPROACHES

Table 5-1 gives a comparison of the three approaches for integrating an ES into an EMS center. The comparison is based on the structure of the integrated system such as, the number of computers, the number of operator displays and keyboards, and the number of databases involved. It is also based on the means for integration, the requirements for the EMS computer and whether the ES can access the EMS software.

Table 5-1 Comparison of the three approaches for integrating expert systems into an EMS

	Approaches		
Items	Appended	DBI	FI
Computers	Two	One	One
Databases	Two	One	One
Basic means for integration	Datalink	Data base	DB+PROCMAN
ES on host comp.	No	Yes	Yes
Expandable host	Not needed	Yes	Yes
keyboard	Two	Two	One
Displays	Separate	Separate	Unified
Process control	Separate	Separate	Unified
ES ACCESS EMS S/W	Via datalink	Via database	Yes

The appended approach features two computers, two databases, a datalink, separate CRT displays and separate keyboards. The datalink transfers the necessary data between the two computers. A communication protocal must be established between the two computers. A two-way datalink would allow the ES to access the EMS database and to issue the control commands from the ES to the EMS computer. For the present implementation of CRAFT, however, the datalink is one-way from the EMS computer

to the ES computer, i.e. the ES can only receive alarms and predefined data sent from the EMS computer. This limited datalink determines that separate displays and keyboards are used for the man-machine interface and for the control purposes. It is possible to use a single console screen and a single keyboard to access different computers if an efficient network and the "multi-window" display facilities are available in the appended (or the "distributed") system. The comparison with this advanced network system is not shown on Table 5-1. An advantage of the appended approach over the two direct integration approaches is that it is the least disruptive to normal system operation. Furthermore, the main computer need not be extended to support the expert system. The only modification on the main computer is to add a data interface to transfer the necessary alarm and data to the datalink. However, the appended computer can not directly access the EMS real time database due to the one-way datalink. As a result, the ES computer has to maintain its own database to be consistent with that of the EMS computer. In addition, due to the limitation of the datalink, the expert system on the appended computer can not use the EMS software facilities such as the standard user interface manager for building customized displays.

In contrast to the appended approach, the database integration (DBI) and the full integration (FI) approaches avoid the appended computer and the datalink. Only one computer, the host EMS computer, and only one database, the EMS database, are used to support both EMS software and the expert system. As a result, the database is directly accessible to the ES. The system configuration is simpler. With the use of a real time database management system, without the limitation of the datalink and without the communication overhead, the ES system response will be faster. Since the expert system is installed on the EMS host computer, the host computer must be able to support the expert system.

An important advantage of the direct integration approaches is that the EMS/DTS console displays can be extended for the ES. Thus, the user can interact with the ES via the standard EMS displays and command set.

The database integration approach provides a prototype for the basic integration of expert systems into EMS centers. It allows the expert system to adequately receive data and communicate the results back to the EMS system. The difficulty with that approach is that the expert system remains a separate computational process from the other tasks

under the EMS/DTS process control. As a result, the DBI approach requires a separate display and keyboard for the user to control and interact with the ES. The additional display hardware and keyboard increase the complexity of the operator console, and it, in turn, increases the possibility of human error. More importantly, the coordination of the expert system with the EMS software is more awkward making the operation less flexible. The result is that one reserves the expert system set-ups for only a few activities rather than developing them whenever they would be useful. Using the DBI approach, the ES can access the EMS utility library, but the access to other main programs is only through the database.

The full integration approach removes these limitations. An expert system is integrated with the EMS on both the data and control levels. The database provides the media for data communication among all software modules and the process manager maintains control over the expert system in exactly the same way it controls all other software in the EMS. Thus, an expert system becomes simply one of the applications in the EMS software package and interacts with the rest of the software in a natural way. This feature gives complete flexibility in employing the expert system so that it is available whenever appropriate. The importance of this fact should not be underemphasized. With complete availability of rule based programming, one can begin to imagine that some standard EMS operations may be better programmed in a rule based or in a hybrid manner rather than in a procedural manner.

The other major advantage of full integration is that the operator uses all the power and familiarity of the standard EMS man-machine interaction utilities. First, this fact means simply that the customary keyboard and display hardware are suitable for the user interface. More importantly, it means that the same user interface utilities can now develop new displays specifically for the expert system operation which are called up by the operator in the standard manner.

Having claimed all these advantages for full integration, we should mention that there may yet be a role for the database integrated approach in an EMS center during development. The database integration is simpler to perform and may be useful for development and debugging, especially if the expert system is being ported from a separate development environment. However, we believe that full integration is necessary for the production environment.

CHAPTER 6

FUTURE IMPLEMENTATION OF EXPERT SYSTEMS IN EMS CENTERS

This study provides basic results to lead the way for integrating a new class of software tools, e.g. expert systems, into EMS centers. Many other AI tools (not just expert systems) with different language tools can also be integrated into EMS centers using the approach developed here. In the following sections, possibilities for future implementation of ES's into EMS centers are discussed. First, a plan to fully integrate CRAFT with a DTS is presented. Then a generalized idea of directly integrating a generic expert system into an EMS center is discussed. Finally a guideline for selecting the software environment is provided.

6.1 FULL INTEGRATION OF CRAFT INTO THE DTS

Chapter 3 presents the integration of CRAFT with the DTS using the database integration approach. It has demonstrated the feasibility of directly embedding an expert system into EMS centers. However, CRAFT remains as a separate computational process from the DTS. A separate display and keyboard are required for user interaction with CRAFT and there is a potential problem of coordinating CRAFT with the DTS. The principle of full integration can be applied to solve these problems. It is necessary to integrate CRAFT into DTS on both data and control levels. As a result, CRAFT "tightly" couples and more naturally interacts with the DTS software.

Since the basic problems related to the data integration of CRAFT with the DTS have been solved, further discussion will focus on the control integration. That is the control integration of CRAFT and CBSIM with the rest of the DTS software modules using the DTS process manager. An implementation plan for this integration is listed below:

1) Define CRAFT and CBSIM as two separated tasks within the DTS.

2) Determine the control information required for CRAFT and CBSIM.

3) Modify the PROCMAN database to include the control information of the new tasks. This modification involves inserting records of new tasks, nodes, paths, conditional flags and schedules into the database.

4) Use the user callable PROCMAN routines within CRAFT and CBSIM to communicate with the PROCMAN database and with PROCMAN.

5) Design the message database and build displays for the user to interact with CBSIM and CRAFT.

Note that, the DTS contains a power system model and an EMS model. Logically, CRAFT should be added to the EMS model because it is an EMS function. CBSIM should be added to the power system model because it simulates the response of power system components (circuit breakers) to line faults. Figure 6-1 gives a conceptual design. CBSIM is inserted into the power system model and CRAFT is added to the EMS model. The HABITAT database is extended to support them and the process manager controls all the activities on the DTS. For simplicity, the data paths from CRAFT to the database and the control paths from PROCMAN to CRAFT are not shown in the diagram.

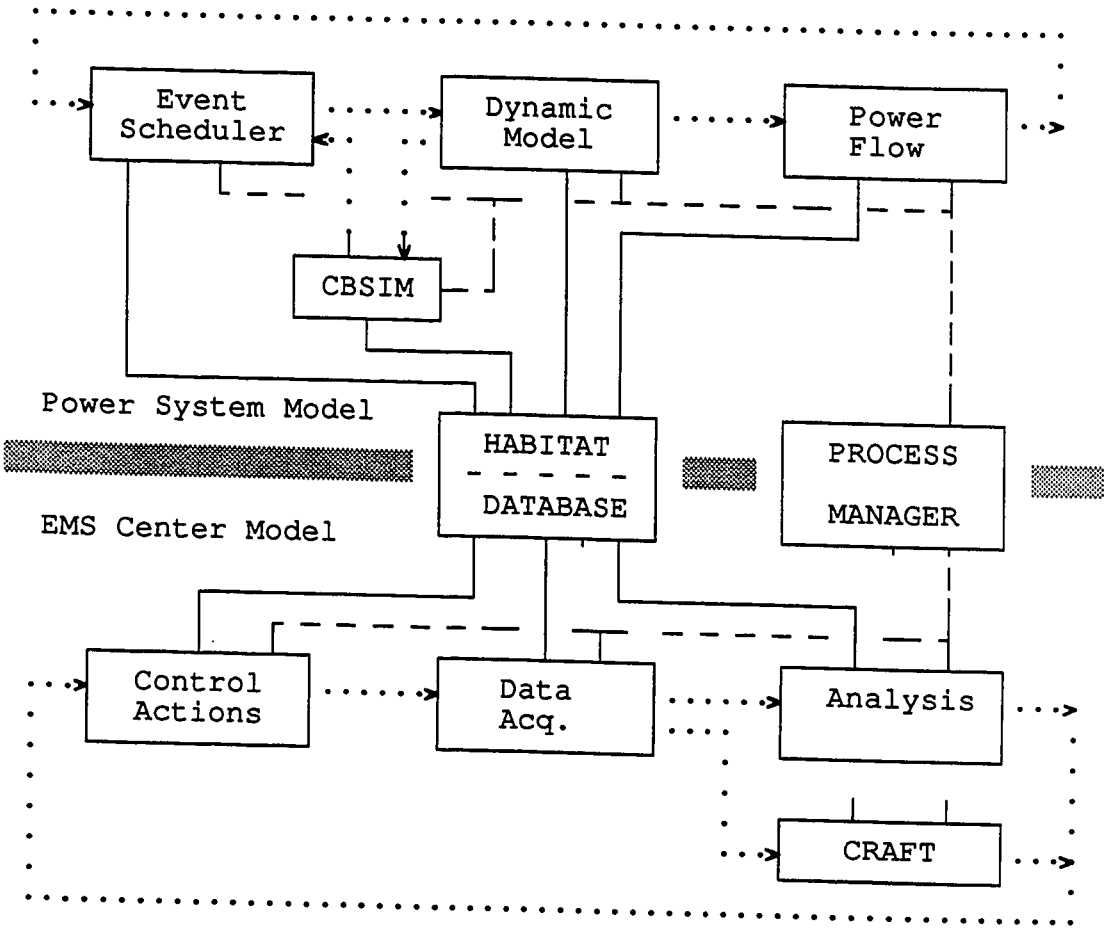


Figure 6-1 Conceptual design of the full integration of CRAFT and the CBSIM with the DTS

Before using the PROCMAN map to plan the control integration, the control information of the new processes must be identified. That is, we must identify the sequence and synchronization requirements of the added modules with the existing applications.

The conditions under which the circuit breaker simulator CBSIM should be activated are any of the following:

1) A manual start to simulate line fault effects.
2) The first reclosure of a circuit breaker if it was previously tripped due to line fault.
3) An additional reclosure of a circuit breaker if the previous reclosure failed.
4) A request for testing reclosure of a circuit breaker.

The conditions for CRAFT to be activated are the combination of:

1) The existence of device alarms
2) CRAFT is not busy.

The condition for CRAFT to wait is either of:

1) Test actions are suggested
2) Remedial actions are posted on the screen for approval.

The condition for CRAFT to continue its analysis is:

1) When the actions are completed and the system stabilized.
Note that, fifteen seconds are commonly needed for stabilization after a test reclosure of a circuit breaker.

After identification of the control information for the interrelated processes, the PROCMAN map can be used to plan and design the integration. Figure 6-2 shows a portion of the system control map. It is similar to the control map in Figure 4-6 for the VCES-DTS system. The difference is the added node of CBSIM and the associated control paths. The functional description of other blocks is found in Section 4.

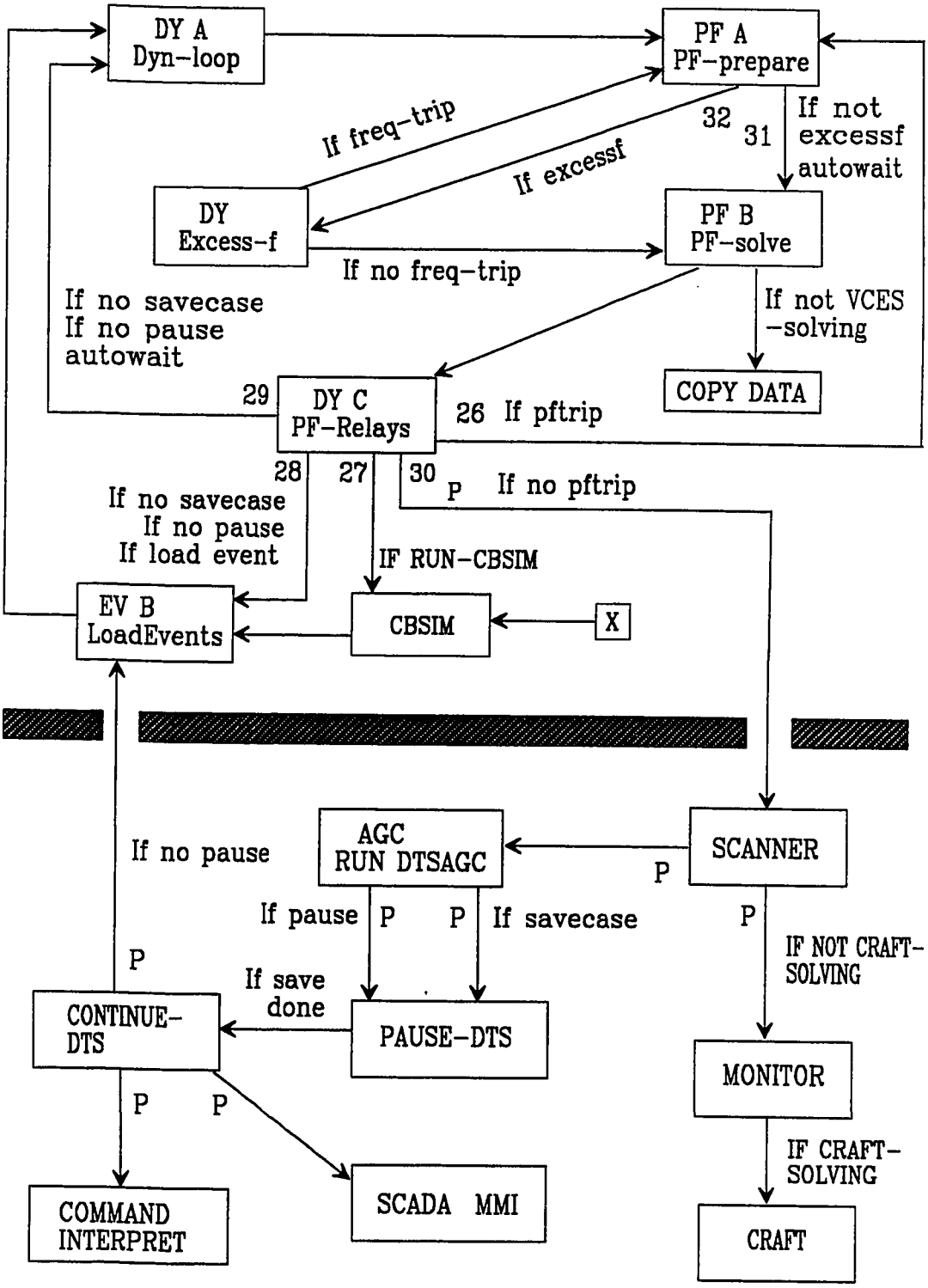


Figure 6-2 A close view of a portion of the modified DTS control map

On the DTS, a user can set up a 'line fault event' by specifying the fault location and the fault time. At the event time, CBSIM is activated to simulate the circuit breakers' first response to the fault. If a tripped circuit breaker has a reclosure, then a schedule is set up for CBSIM to run at that reclosure time to determine the success of the reclosure. This schedule can be set up by calling PROCMAN_SETSKED, a user callable PROCMAN subroutine. The output of the CBSIM may be an event to trip a circuit breaker. This event is sent to the events database and is loaded by LoadEvents before the next simulation cycle.

On the EMS side, the SCANNER performs the data acquisition function. CRAFT is normally in stand-by and is activated by the device alarm MONITOR. When the CRAFT suggested remedial or test actions are approved by the dispatcher, the actions not involving circuit breaker reclosures are send to the event database for execution while the circuit breaker reclosure event is sent to the CBSIM database for simulation and the condition flag 'RUN-CBSIM' is set to true which allows the path to the CBSIM to be turned on.

Customized displays should be built for manual control and user interface. At the time when the user intervention is needed, an associated display should be sent to the console CRT screen. The display should provide data to inform the user and to provide fields for the user to enter commands. The displays should be fully integrated with the existing DTS displays. As a result, only the standard console displays and keyboard are used for the user interface.

6.2 INTEGRATION OF GENERIC EXPERT SYSTEM INTO AN EMS

Actually the importance of the full integration of the VCES and CRAFT into an energy management system (EMS) environment goes much beyond the problems they solve. An EMS center has many tools to aid the operator in one of his/her major tasks: the solving of network management problems. The VCES or CRAFT is such a tool but is developed using a knowledge base or non-procedural approach. The successful integration of these expert systems into an EMS center opens the way for many other tools and operator aids that are also developed using new programming approaches. Hence, much of the value of the examples is to show the way for a new class of extensions of an EMS center's capabilities. In the following discussion it is assumed that the host computer system

allows such an extension.

Integrating a generic expert system fully into the EMS center requires database integration and control integration. The database integration is on the data level and the control integration is on the control level. Figure 6-3 illustrates this generalized concept. Note that the control database is emphasized here by separating it from the domain database. As the PROCMAN database, this generalized control database only contains the control information. The domain database provides data communication media for all the EMS applications which are controlled by the process manager. The blocks labeled P1, P2, P3 stand for existing application programs in the EMS. The block labeled ES is an expert system. As a result, an ES becomes a part of the EMS function.

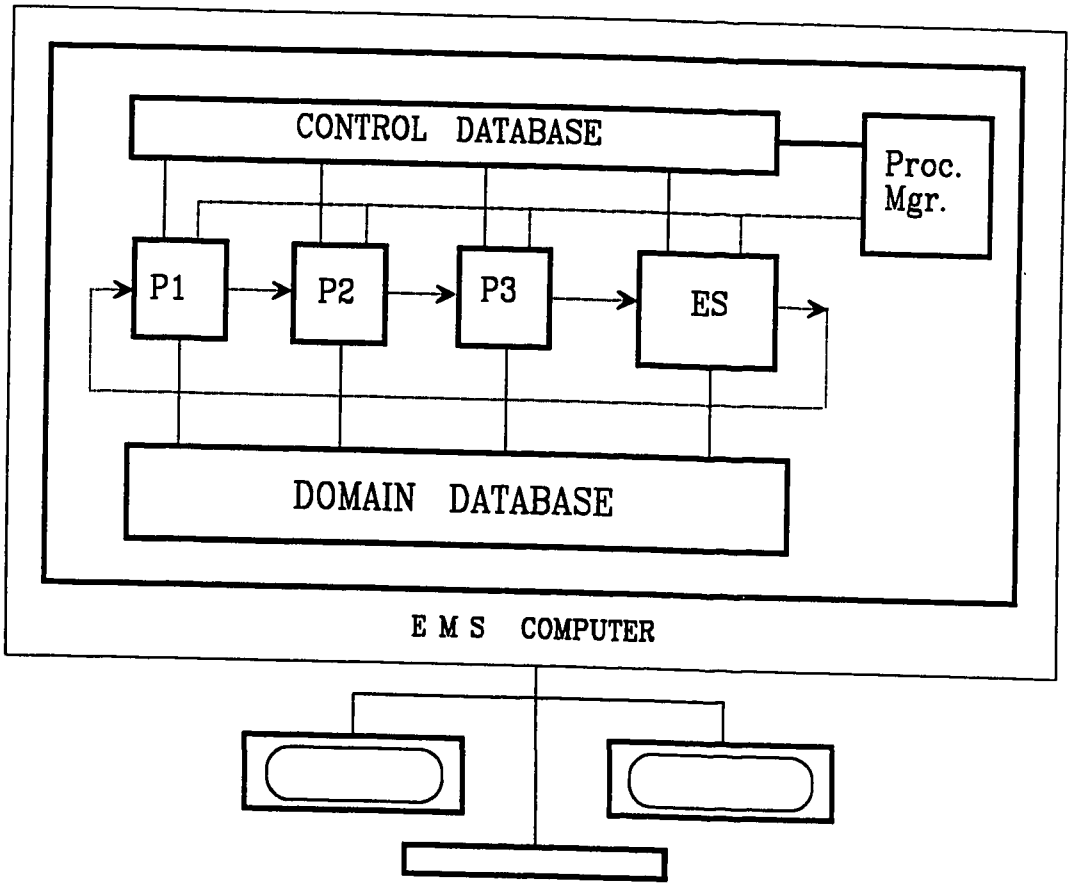


Figure 6-3 Integration of a generic expert system into an EMS center

1) DATABASE INTEGRATION

A database management system is usually used to support an EMS system. An expert system commonly has a working memory to support the knowledge base and the inference engine. The database integration involves extending the EMS system database and building the data interface between the database and the working memory. The database extension is made by appending a database partition to the system database. This database partition contains new data items and pointers. The new data items are needed by the ES but do not appear in the original database. The pointers are used to gain access to the existing data in the system database. Using pointers avoids duplication of data and speeds up the access to the database.

Before extending the database, it is necessary to identify the input data and the output data of the expert system. What data items are available in the existing database and what data items are not available must also be identified. Of course, one should understand the data structure of the system database and of the working memory. Since the system database supports all activities in the system, it is usually big. The understanding of the data definition, data structure and the conventions used in the data organization will help one to locate the data and to use it correctly. The working memory data structure, on the other hand, supports the expert system application. The data structure may not be fully compatible with that of the system database due to differences in languages. The understanding of the data helps one to map data correctly between the working memory and the system database and to develop the data interface.

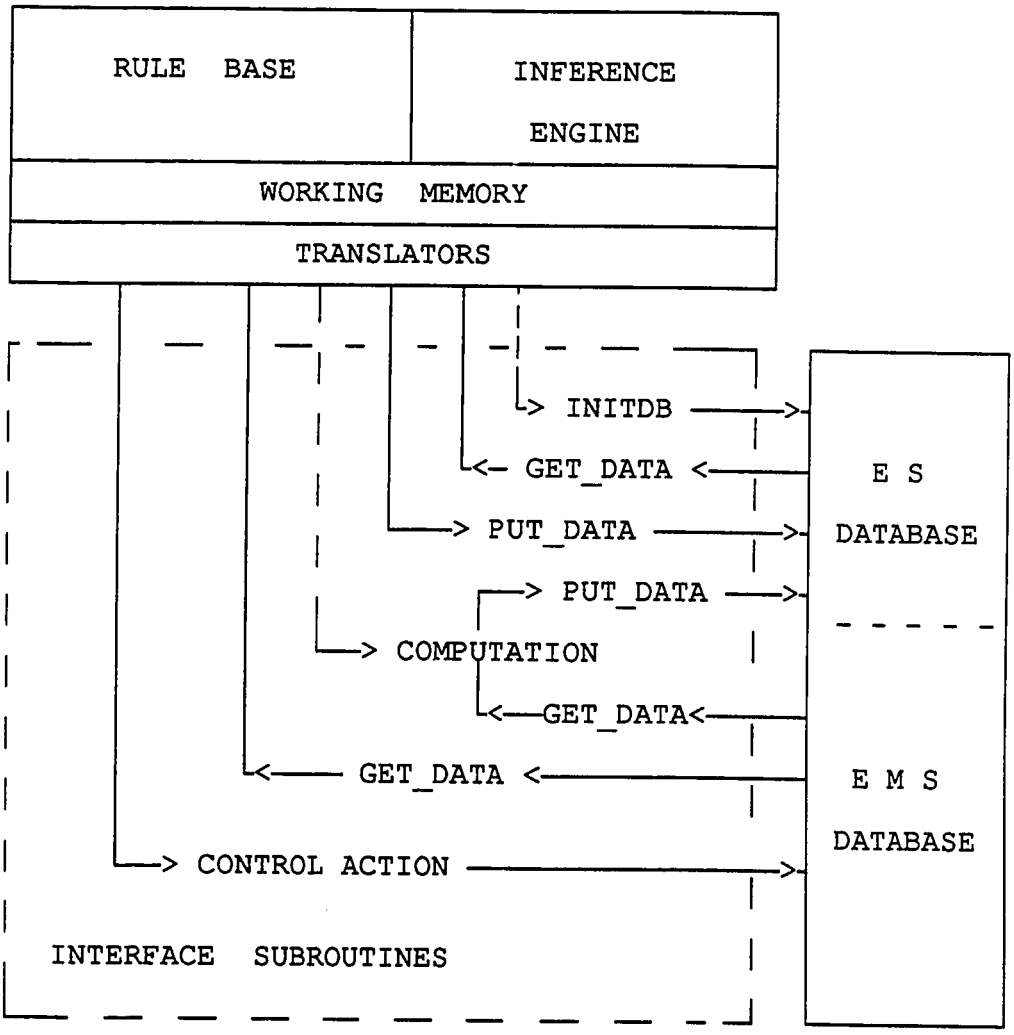


Figure 6-4 Data interface procedures and translators

The interface between the database and the working memory consists of subroutines and translators. The translators are contained in the OPS83 procedures. The interface subroutines in the dashed block of Figure 6-4 may be classified into four types based on the activity performed:

1) Open and initialize the ES database.
2) Perform computation or data transformation and then put the results into the ES database.
3) Get data from the database.
4) Put data into the database or put control action into the database.

The first type of interface routine is called at the beginning of the ES main program to open databases and to initialize the extended ES database. No data transfer from the ES working memory to the database in the routine. The second type of interface routine performs computation on or transformation of EMS data for the ES and places it in the ES database. This second type routine is called in the ES program, but no data is passed between the ES working memory and the database. The last two types of interface routines involve passing data between the database and the working memory. The importance of correctly mapping data can not be over emphasized. The translator plays a key role in the data mapping between different programming languages because the data structure provided in the expert system may not be compatible with that in conventional programming languages, such as FORTRAN.

2) CONTROL INTEGRATION

Applications in modern EMS systems are commonly organized as scheduled tasks (or processes). To maintain an orderly schedule, the tasks are usually controlled by a task scheduler, or a process manager. A database is usually used to support the process manager. This database is called the control database because it maintains the control information such as the sequence and synchronization information of the processes. Based on the status of this database, the process manager determines what process to execute next. The execution of a process may interact with the control database by setting or resetting certain bits in the database thereby communicating with the process manager and affecting the control flow. This control mechanism is referred to as 'database driven'. The control integration of an expert system with EMS software can be

accomplished by exploiting this control database and the process manager on the host system.

Similar to the database integration, the control integration involves extending and modifying the control database and using interface routines to couple to this database. However, the control database extension is not made by appending a new database partition to the control database. Rather, the extension and modification is directly made on the central control database. New control information is inserted into the control database.

The design of the control integration usually starts with a control map which depicts the relationships between tasks. The map consists of nodes, directional paths, condition flags and labels. The control database is actually a structured data representation of the control map.

The lower part of Figure 6-5 shows a modified EMS control map with an expert system added. A node is the entrance point of a task. Nodes are represented by blocks on the control map. The label in a block names an EMS function. For instance, the block SCANNER represents the data acquisition or state estimation function. The block ANALYSIS represents an EMS analysis function, such as economic dispatch or contingency analysis. The MONITOR and ES blocks, with the associated paths, represent the added functions. The control actions from the ANALYSIS or ES are carried out at the CONTROL ACTION node.

Paths from a node are evaluated in a sequence defined when the control database is established. Note that there are two paths out from SCANNER. The character 'P' near a path stands for parallel. The parallel attribute overrides the sequence. It means that the path to the MONITOR is evaluated simultaneously with the other path. A path can be conditioned and scheduled. The value of a condition can be set or reset by other processes. If the conditions imposed on the path are all true then the path is turned on and the control is passed onto the node MONITOR.

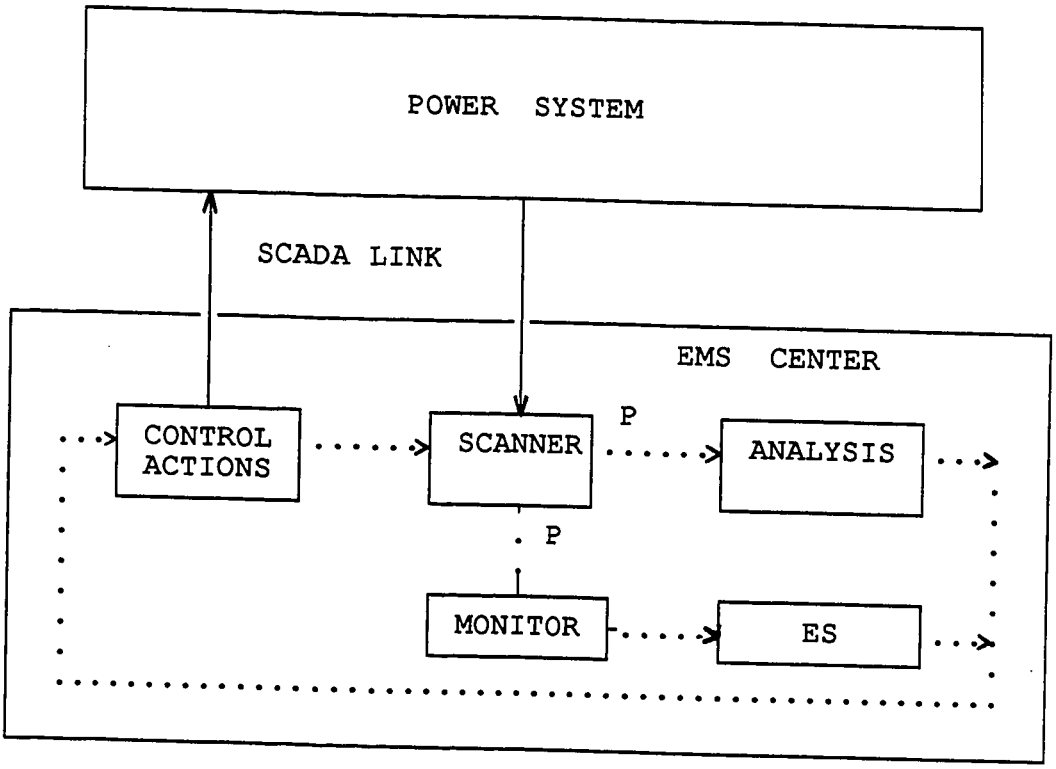


Figure 6-5 Control map for process coordination

To put an expert system under the control of a process manager, the control information of the ES must be identified and put into the control database. Since an ES is designed to solve a specific problem, the existence of the specific problem is such a control signal, the one necessary to turn on the ES. Hence, a monitor is used to detect the problem. The path from the SCANNER node to the MONITOR node means that this monitor is turned on after the data acquisition task.

Figure 6-6 illustrates the control flow in the integrated portion SCANNER-MONITOR-ES. At node "SCANNER", the data acquisition task is performed. There are two parallel paths, out of this block and a conditional flag 'NOT ES-SOLVING' imposed on the path to the MONITOR. When "SCANNER" completes its task, if the condition flag 'NOT ES-SOLVING' is TRUE, then the path to the monitor node is turned on and control enters the MONITOR node. As soon as the control reaches the MONITOR node, the flag "ES-SOLVING" is set to TRUE. This setting closes the path from SCANNER to MONITOR. The monitor module then accesses the relevant database to determine the existence of the specific problem or scenario that the ES should deal with. If the problem does not exist, the condition flag "ES-SOLVING" is reset to FALSE. It thus tells the PROCMAN that the ES is not in a solving mode. If, however, the problem exists, then control is passed to the ES and it will run. The ES may access the database, analyze and interact with the user and reach a solution. The solution may be a judgement with an explanation displayed to the user or it may suggest remedial actions to be sent to the event database for execution. The ES task will end with resetting the flag "ES-SOLVING" to FALSE. This tells the PROCMAN that the ES task is finished and can be restarted.

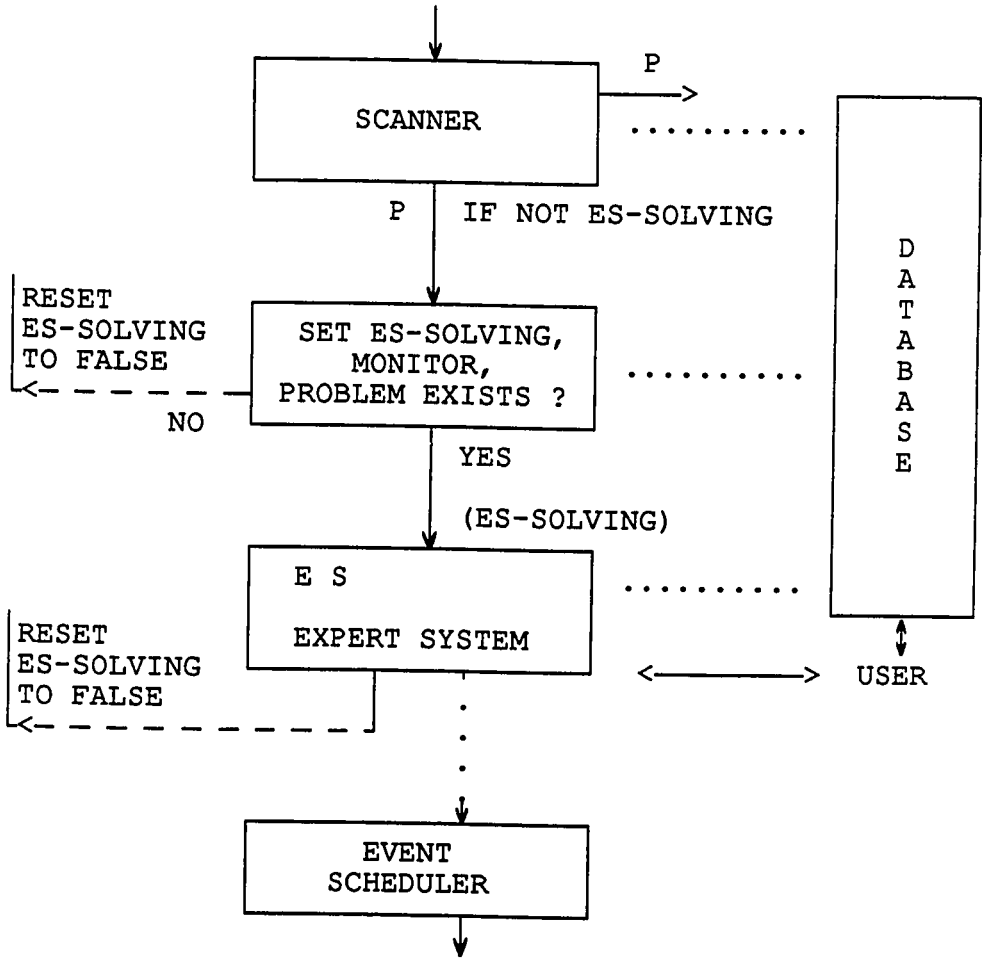


Figure 6-6 Control flow of the ES

6.3 GUIDELINES FOR SELECTING THE HOST COMPUTER SYSTEM

Expandability is required for the host computer system to incorporate the new functionalities. Some utilities with consoles and systems in place find it difficult, sometimes impossible, to take advantage of new technology. Some systems have closed architecture which prevents new software from being installed effectively. Others are driven exclusively by CPUs which are operating at or near their limits so that there is no processing power available for new applications. Still other configurations can accept new software but must be run on dedicated hardware and the operator consoles quickly become crowded with a confusion of terminals and different types of keyboards. The specific requirements of the host system for embedding an ES are:

1) The CPU resource of the host system must be available for the new applications.

2) The host system must support both conventional and expert system languages.

3) The system database must be extendable and accessible by the expert system.

4) The system software should have modular structure that allows new applications to be added.

5) The control information of software modules should be separated and maintained in a database. This facilitates system integration and maintenance.

6) Displays and user interfaces should be extendable so that customized displays and user interfaces can be added to support the ES application.

7) A simulation environment of the power system behavior of interest is needed for development.

As a summary, the implementation of the embedded approach requires an expandable and compatible hardware and software base to accommodate both the conventional and the expert system programs. A modern dispatcher training simulator is just such a base.

References

[1] F. Hayes-Roth, D. Waterman and D. Lenat, Building Expert Systems, Addison-Wesley Co., 1983.

[2] B. F. Wollenberg, "Feasibility Study for an Energy Management System Intelligent Alarm Processor", IEEE Transactions on Power Systems, Vol. PWRS-1, No. 2, May 1986, pp. 241-247.

[3] An International Survey of the Present Status and the Perspective of Expert Systems on Power System Analysis and Techniques, CIGRE SC 38 WG 02 TF 07 Report, March 1988.

[4] Proceedings of Symposium on Expert System Applications to Power Systems, Stockholm-Helsinki, August 1988.

[5] Proceedings of Symposium on Expert System Applications to Power Systems, Seattle, July 1988.

[6] C. C. Liu and M. J. Damborg, CRAFT: An On-line Operational Expert Systems for Customer Restoration And Fault Testing, EPRI RP-2944-1 Final Report, 1989.

[7] C. C. Liu and T. Dillon, "State-of-the-Art of Expert System Applications to Power Systems," Int. Journal Electric Power and Energy Systems, Oct. 1989.

[8] C. Liu and K. Tomsovic, "An Expert System Assisting Decision-Making of Reactive Power/Voltage Control", IEEE Transactions on Power Systems, Vol. PWRS-1, No. 3, Aug. 1986, pp. 195-201.

[9] T. Sakaguchi et al. "Prospects of Expert Systems in Power System Operation", Proc. 9th Power Systems Computation Conference, Lisbon, Portugal, 1987.

[10] R. D. Christie and S. N. Talukdar, "Expert Systems for On Line Security Assessment A Preliminary Design", PICA Conference, Montreal, 1987, pp. 114-119.

[11] E. A. Feigenbaum, "Knowledge Engineering for the 1980's," Computer Science Department, Stanford University, 1982.

[12] E.H. Shortliffe, Computer-based Medical Consultation: MYCIN. Elsevier, New York. 1976.

[13] F. Hayes-Roth and Victor R. Lesser. "Focus of Attention on the Hearsay-II Speech Understanding System," Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Aug. 1977, pp. 27-35.

[14] Alistair D. C. Holden, "Adaptive Methods in Speech Understanding Systems," Proceedings of IEEE International Conference on Cybernetics & Society, 1982.

[15] J. McDermott, "R1: A Rule Based Configurer of Computer Systems", Artificial Intelligence, Vol. 19, 1982, pp. 39-88.

[16] K.L. Tomsovic, C.C. Liu, P. Ackerman and S. Pope, "An Expert System as a Dispatcher's Aid for the Isolation of Line Section Faults," IEEE Trans. on Power Delivery, Sept. 1986.

[17] K. Tomsovic, "Development of Expert Systems as On-Line Operational Aids", PhD Thesis, Department of Electrical Engineering, University of Washington, 1987.

[18] K. Komai, T. Sakaguchi and S. Takeda, "Power System Fault Diagnosis with an Expert System Enhanced by the General Problem Solving Method", IASTED Conference, Bozeman, Montana (USA), Aug. 1986.

[19] E. Cardozo and S. N. Talukdar, "A Distributed Expert System For Fault Diagnosis", PICA Conference, Montreal, 1987, pp. 101-106.

[20] N. Wada, et. al., "A Real Time Expert System for Power System Fault Analysis", IASTED Conference, Bozeman, Montana(USA), Aug. 1986.

[21] S. Mokhtari, J. Singh and B. Wollenberg, "A Unit Commitment Expert System", IEEE PICA conference, Montreal, 1987, pp. 400-405.

[22] C. C. Liu, S. J. Lee and S. S. Venkata, "An Expert System Operational Aid for Restoration and Loss Reduction of Distributed Systems", PICA Conference, 1987, Montreal, pp. 79-85.

[23] C.C. Liu, H. Marathe and K.L. Tomsovic, "A Voltage Control Expert System and Its Performance Evaluation," to appear as a chapter in Expert System Applications to Power Systems, Eds. T. Dillon and M. Laughton.

[24] J. Kawakami and S. Tamura, "An Expert System for Voltage-Var Scheduling," Proc. 1987 PSCC Conference, Portugal, pp. 702-707.

[25] H.Y. Marathe, C.C. Liu, M.S. Tsai, R.G. Rogers and J.M. Maurer, "An On-Line Operational Expert System with Data Validation Capabilities," Proc. 1989 PICA Conference, Seattle, May 1989.

[26] M.J. Damborg and Ming Chen, "An Example of Integrating an Expert System into a Control Center Using A Dispatcher Training Simulator," Proc. Symposium of Expert Systems Application to Power Systems, Stockholm-Helsinki, Aug. 1988.

[27] M. Chen, M.J. Damborg and T.M. Athay, "Full Integration of an Expert System into an Energy Management System Using A Dispatcher Training Simulator," Proc. Symposium of Expert Systems Application to Power Systems, Seattle, Jul. 1989. pp. 474-479.

[28] C. Forgy, "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, Vol. 19, No. 1, 1982, pp. 17-37.

[29] M. J. Damborg and S.S. Venkata, Specification of Computer Aided Design of Transmission Protection Systems, Final report EL-3337, PR 1764-6, EPRI, Jan. 1984.

[30] C.J. Date, An Introduction to Database Systems, New York: Addison-Wesley, Vol. 1, 4th Edition, 1984

[31] C. Liu, K. Tomsovic and S. Zhang, "Efficiency of Expert Systems As On-Line Operating Aids", Proc. 9th Power Systems Computation Conference, Lisbon, Portugal, 1987.

[32] C.C. Liu, H. Marathe and K.L. Tomsovic, "A Voltage Control Expert System and Its Performance Evaluation," to appear as a chapter in Expert System Applications to Power Systems, Eds. T. Dillon and M. Laughton.

[33] K. R. C. Mamandur, "Emergency Adjustments to VAR Control Variables to Alleviate Over-Voltages, Under-Voltages and Generator Var Limit Violations," IEEE Transactions on Power Apparatus and Systems, Vol. Pas-101, No. 5, pp. 1040-1047, May 1982.

[34] A. Botnen, "Integrating an AI-Frontend with Existing Numeric Software," Proc. Symposium of Expert Systems Application to Power Systems, Stockholm-Helsinki, Aug. 1988.

[35] R. Podmore, J.C. Giri, M.P. Goreuberg, J.P. Britton, N.M. Peterson, "An Advanced Dispatcher Training Simulator," IEEE Trans. on Power Apparatus and Systems, Vol. PAS-106, No. 1, Jan. 1982, pp. 17-25.

[36] ESCA Corp., HABITAT User's Guide, May, 1987.

[37] ESCA Corp., DTS User's Guide, November, 1985.

[38] ESCA Corp., PROCMAN Reference Manual, 1989.

[39] B. Hayes-Roth, "A Blackboard Architecture for Control," Artificial Intelligence, Vol. 26 1985. pp. 251-321.

[40] T. Sakaguchi and K. Matsumoto, "Development of a Knowledge based System for Power System Restoration", IEEE Transactions on Power Apparatus and Systems, Vol. PAS-102, No. 2, Feb. 1983, pp. 320-329.

[41] B. Stott and O. Alsac, "Optimal Load Flow with Steady-State Security," IEEE Trans. on Power Apparatus and Systems, Vol. PAS-93, No. 3, May/June 1974, pp. 745-751.

[42] P. Stoa and G. Aakvik, "An Architecture for Flexible Control of Problem Solving Behavior in a Hybrid System," Proc. Symposium of Expert Systems Application to Power Systems, Stockholm-Helsinki, Aug. 1988

[43] C. C. Liu and M. J. Damborg, Development of Expert Systems as On-line Power System Operational Aids, EPRI EL-5635 Project 1999-9 Final Report, Feb. 1988.

[44] B. C. Schwab, K Hemmaplardh, S. A. Sackett, C. M. Kitto, D.J. Inglis, C. D. Meier, "A Developed Database Management System for Operation and Planning Applications," IEEE Trans. on Power Apparatus and Systems, Vol. Pas-104, No. 7, Jul. 1985.

[45] Display Design for Dispatch Control Centers in Electric Utilities: Handbook, Final

Report EPRI EL-4960, Westinghouse Research and Development Center, March 1987.

[46] C. L. Forgy, The OPS83 Report, System Version 2.1, Computer Science Department, Carnegie-Mellon University, Oct. 1984

[47] C. L. Forgy, OPS83 User's Manual, Computer Science Department, Carnegie-Mellon University, 1983

[48] C.C. Liu, S.M. Wang, L. Wong, H. Marathe, M. Lauby, "A Self-learning Expert System for Voltage Control of Power System," Proc. of Second Symposium of Expert Systems Application to Power Systems, Seattle, Jul. 1989. pp. 462-468.

[49] L. Barruncho, J.P.S. Paiva, C.C. Liu, "Voltage/Var Control Optimization and Knowledge Oriented Approach," Proc. of Second Symposium of Expert Systems Application to Power Systems, Seattle, Jul. 1989. pp. 453-457.

[50] K. Tomsovic, L. Barruncho, "Petri Net Representation as an Evaluation Method for a Voltage Control Expert System," Proc. of Second Symposium of Expert Systems Application to Power Systems, Seattle, Jul. 1989. pp. 410-415.

[51] A.H. Shoop, S. Silverman, B. Ramesh, "Consolidated Edison System Operation Computer Control System (SOCCS) Alarm Advisor," Proc. of Second Symposium of Expert Systems Application to Power Systems, Seattle, Jul. 1989. pp. 84-88.

[52] J. Zaborszky, G. Huang and K. W. Lu, "A Textured Model for Computationally Efficient Reactive Power Control and Management," Paper No. 845SM 517-7, IEEE-pes Summer Meeting, July 1984.

Appendix A

Modeling automatic switches and their combinations in the DTS

All the automatic switches listed on page 43 are presented here in the form in which they appear in the Puget Power Automatic Switch Book and in the form of the DTS equivalents. The data records that appear in the DTS database are also shown.

1) Model ODB (Open on Dead Bus) switch

CB record:

Name of CB	: 12
From node	: 1
To node	: 2
Normal status	: closed

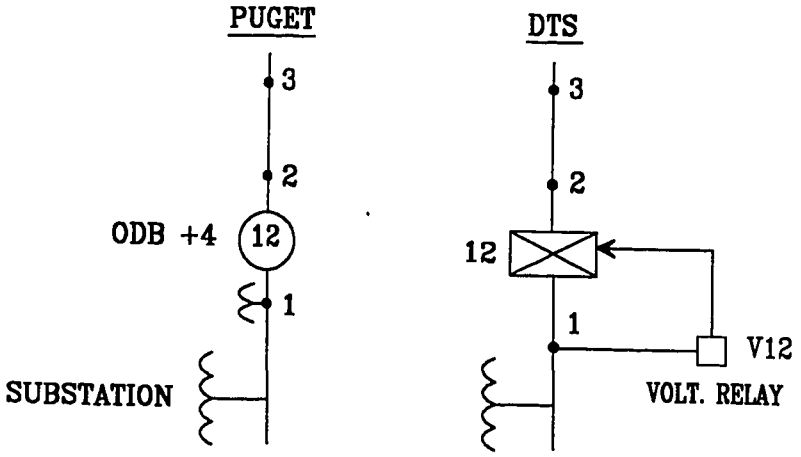


Figure A-1 Modeling ODB

Voltage Relay record:

Name of VRY	: V12
Name of activated CB	: 12
Name of sensed node	: 1
Min activate voltage (p.u.)	: 0.60
Time delay before Min trip (sec)	: 4
volt violation	: open

To construct an ODB (Open on Dead Bus) switch model, a relay is assigned to a circuit breaker and is defined to sense a node. Note that, a relay parameter called "Min activate voltage" is set to 0.6 p.u. If the bus (node) voltage drops below 0.6 p.u. then the node is considered "dead" and the relay trips the associated CB.

2) Model ODL (Open on Dead Line) switch

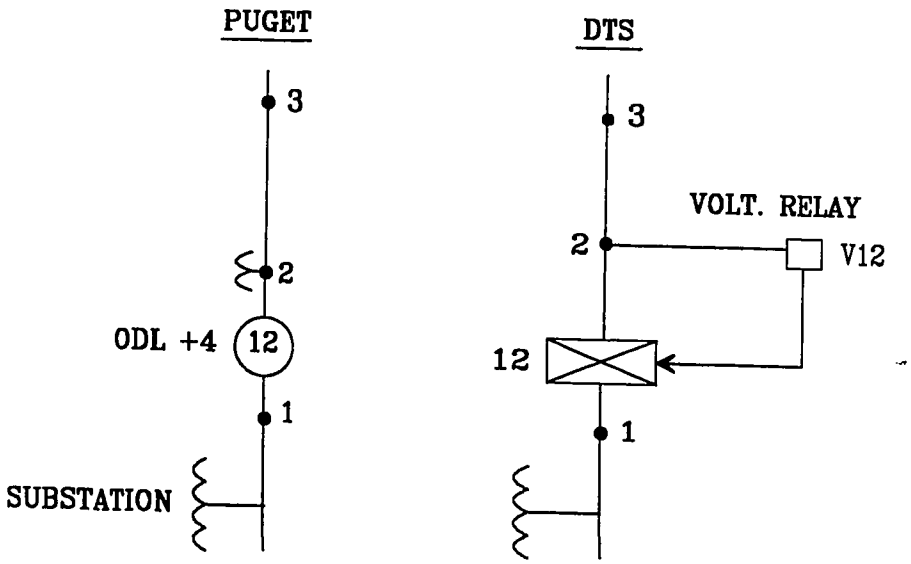


Figure A-2 Modeling ODL

CB record:

Name of CB	: 12
From node	: 1
To node	: 2
Normal status	: closed

Voltage Relay record:

Name of VRY	: V12
Name of activated CB	: 12
Name of sensed node	: 2
Min activate voltage (p.u.)	: 0.60
Time delay before Min trip (sec)	: 4
CB action on Min volt violation	: open

The difference between this model and the ODB model is that the location of the sensed node is on the line side rather than the bus side.

3) Model CDL (Close on Dead Line) or CDB (Close on Dead Bus) switches

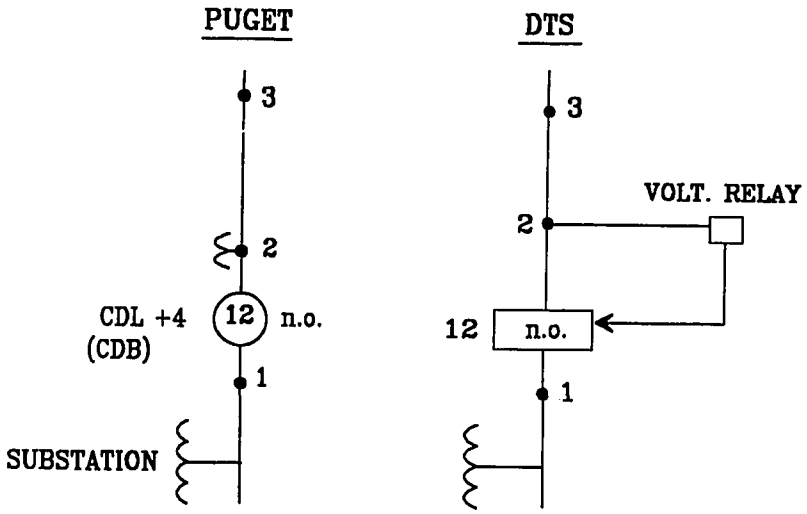


Figure A-3 Modeling CDL (CDB)

CB record:

Name of CB	: 12
From node	: 1
To node	: 2
Normal status	: open

Voltage Relay record:

Name of VRY	: V12
Name of activated CB	: 12
Name of sensed node	: 2
Min activate voltage (p.u.)	: 0.60
Time delay before Min trip (sec)	: 4
CB action on Min volt violation	: close

Switch 12 is normally open. the sensor is at node 2. The relay will trip to close the switch if node 2 is sensed dead for 4 seconds. In the case of ODL, node 2 is on the line side. If node 2 is on the bus side then this figure models the CDB switch.

4) Model CHB (Close on HOT Bus) or CHL (Close on Hot Line) switches

CB record:

Name of CB	: 12
From node	: 1
To node	: 2
Normal status	: open

Voltage Relay record:

Name of VRY	: V12
Name of activated CB	: 12
Name of sensed node	: 1
Max activate voltage (p.u.)	: 0.80
Time delay before Max trip (sec)	: 4
CB action on Max volt violation	: close

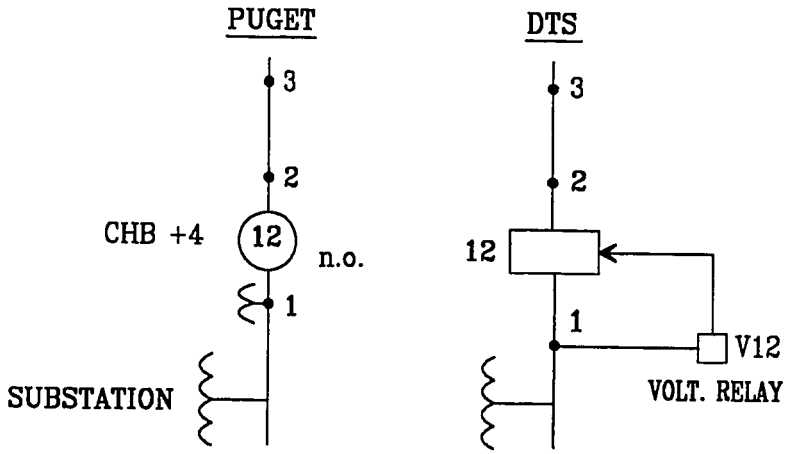


Figure A-4 Modeling CHB (CHL)

Switch 12 is normally open. the sensor is at node 1. The relay will trip to close the switch if node 1 is sensed dead for 4 seconds. In the case of CHB, node 1 is a bus node. If node 1 is on the line side then this figure models the CHL switch.

5) Model the combination of automatic switches

ODL+4	Open on Dead Line
CHL+2	Close on Hot Line

CB record:

Name of CB	: 12
From node	: 1
To node	: 2
Normal status	: closed

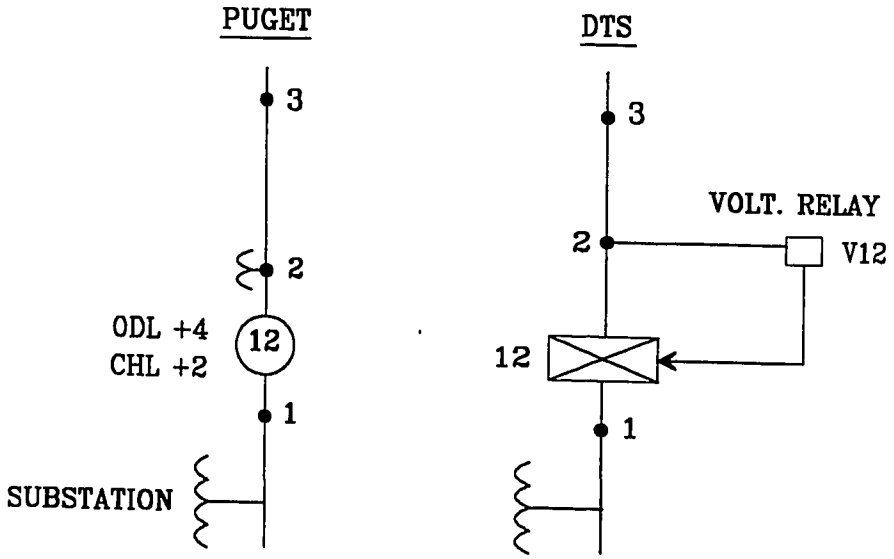


Figure A-5 Modeling ODL + CHL

Voltage Relay record:

Name of VRY	: V12
Name of activated CB	: 12
Name of sensed node	: 2
Min activate voltage (p.u.)	: 0.60
Time delay before Min trip (sec)	: 4
CB action on Min volt violation	: open
Max activate voltage (p.u.)	: 0.80
Time delay before Max trip (sec)	: 2
CB action on Max volt violation	: close

Circuit breaker CB12 is normally closed. If the voltage at node 2 drops below 0.6 for 4 seconds then the relay V12 will trip CB12 to open. When the node voltage is above 0.8 for 2 seconds then the relay will reclose the CB.

6) Model the combination of automatic switches

ODB+4	Open on Dead Bus
CHB+2	Close on Hot Bus

CB record:

Name of CB	: 12
From node	: 1
To node	: 2
Normal status	: closed

Voltage Relay record:

Name of VRY	: V12
Name of activated CB	: 12
Name of sensed node	: 1
Min activate voltage (p.u.)	: 0.60
Time delay before Min trip (sec)	: 4
CB action on Min volt violation	: open
Max activate voltage (p.u.)	: 0.80
Time delay before Max trip (sec)	: 2
CB action on Max volt violation	: close

This type of switch is basically the same as ODL+CHL. The only difference is that the location of the sensed node is on the bus side rather than the line side.

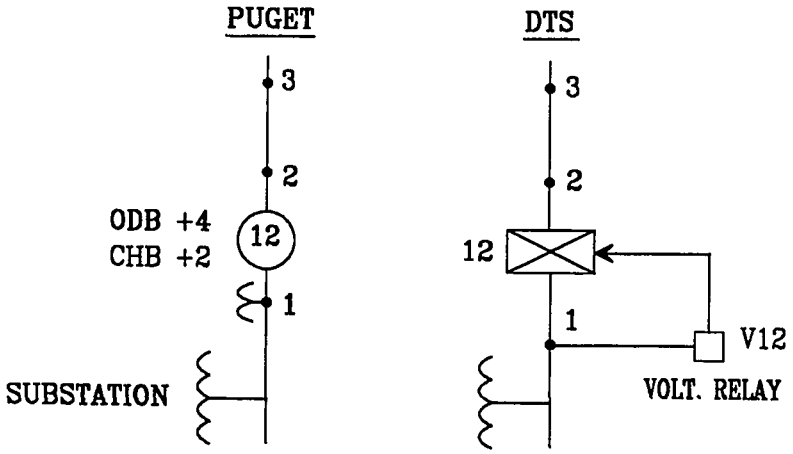


Figure A-6 Modeling ODB + CHB

7) Model ATDB (Automatic Transfer on Dead Bus)

This pair of switches is modeled by two combination switches ODB and CDB and 4 records. The bus node sensed is node 2. If node 2 is dead for 2 seconds, then there is an automatic transfer switch between CB 23 and CB 12. CB 12 will close after CB 23 opens. Power to node 2 is then supplied from the source on the left side.

CB record:		CB record:	
Name of CB	: 12	Name of CB	: 23
From node	: 1	From node	: 2
To node	: 2	To node	: 3
Normal status	: open	Normal status	: closed

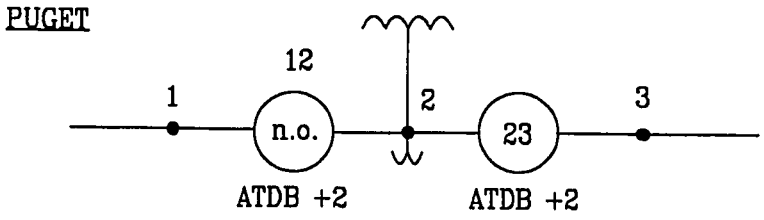


Figure A-7A ATDB in Puget System

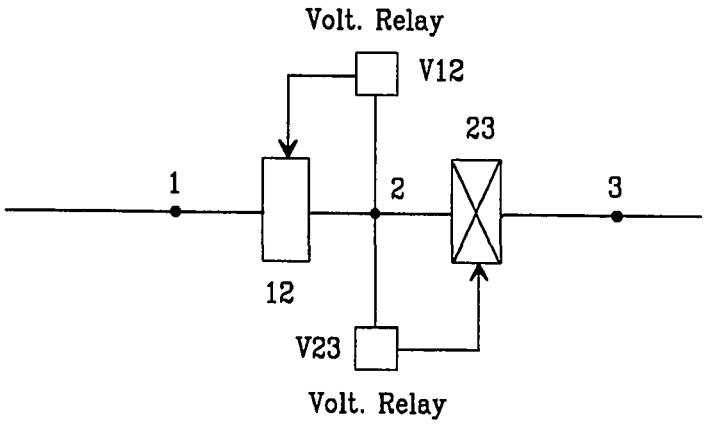


Figure A-7B Modeling ATDB

Voltage Relay record:

Name of VRY	: V12
Name of activated CB	: 12
Name of sensed node	: 2
Min activate voltage (p.u.)	: 0.60
Time delay before Min trip (sec)	: 2.0
CB action on Min volt violation	: close

Voltage Relay record:

Name of VRY	: V23
Name of activated CB	: 23
Name of sensed node	: 2
Min activate voltage (p.u.)	: 0.60
Time delay before Min trip (sec)	: 2.0
CB action on Min volt violation	: open

8) Model ATDL (Automatic Transfer on Dead Line) switch

The model of an ATDL switch is similar to that for ATDB. The only difference is that the sensed node is defined on the line side, node 3 in the following diagram.

The pair of ATDL switches is modeled by two combination switches (NC + ODL) and (NO + CDL) and 4 records. The node sensed is node 3. If node 3 is dead for 2 seconds, then there is an automatic transfer switch between CB 23 and CB 12. CB 12 will close while CB 23 opens. Power to node 2 is then supplied from the source on the left side.

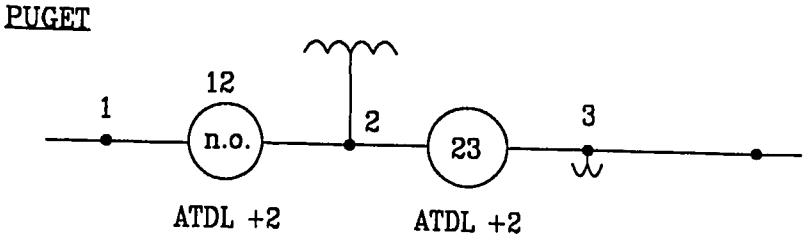


Figure A-8A ATDL in Puget System

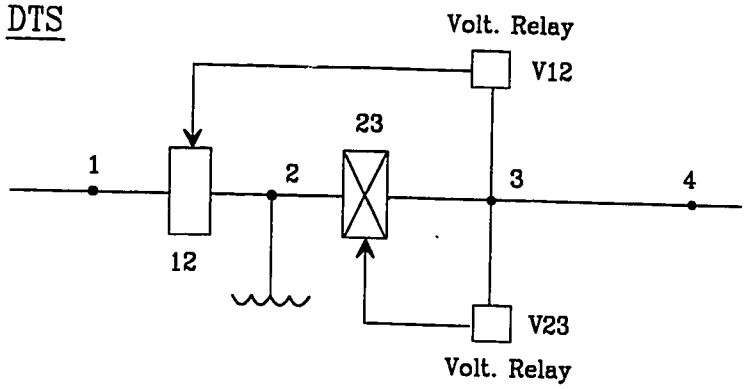


Figure A-8B Modeling ATDL

CB record:

Name of CB	: 12
From node	: 1
To node	: 2
Normal status	: open

CB record:

Name of CB	: 23
From node	: 2
To node	: 3
Normal status	: closed

Voltage Relay record:

Name of VRY	: V12
Name of activated CB	: 12
Name of sensed node	: 3
Min activate voltage (p.u.)	: 0.60
Time delay before Min trip (sec)	: 2.0
CB action on Min volt violation	: close

Voltage Relay record:

Name of VRY	: V23
Name of activated CB	: 23
Name of sensed node	: 3
Min activate voltage (p.u.)	: 0.60
Time delay before Min trip (sec)	: 2.0
CB action on Min volt violation	: open

9) Note that, a special design called the "lock out" operation logic is implemented in the Puget Power system for (ODL+ CHL) switches to avoid reclosing onto a fault. The scenario is:

IF switch type is (ODL + CHL)
and if it tripped at time t1,
and reclosed at time t2
and tripped again at time t3
and if (t3-t2) < specified time, say 10 seconds,

THEN
the switch is open and "locked out",
the automatic function is deactivated until
manual reset.

But IF (t3-t2) > lockout time,

THEN
no lockout.

This logic can not be modeled using the DTS. However, it can be implemented by using a rule based program coupled with the DTS. The lockout function is not implemented in the CRAFT-DTS system.

Appendix B

Mapping data between OPS83 and FORTRAN

Programs written in OPS83 can call procedures written in other languages, such as FORTRAN, Pascal or C. To pass data correctly, the data representations in the routines coded in different languages must agree. As the standard data representations such as integer, real, character, or character array are defined in OPS83, these data types can be passed over directly by assigning it to the arguments. This feature facilitates OPS83 program interface to a database which is accessible by other conventional languages. However, the data type 'symbol'(alphanumeric string with varying length) used in OPS83 does not agree with the fixed length character string type in FORTRAN. Without data transformation, symbolic data can not be passed between OPS83 and FORTRAN.

Matching data types

The following table illustrates the data type correspondence between OPS83 and FORTRAN. It shows how to transform the data representation for integer arrays, real arrays and character arrays in order to pass data directly via the parameters in OPS83 procedures to FORTRAN subroutines.

OPS83	FORTRAN
type int_array=array(6:integer);	INTEGER INT(6)
type real_array=array(6:real);	REAL REAL(6)
type CHA20=array(20:char);	LOGICAL *1 X1(20)
type CHA4=array(4:char);	LOGICAL *1 CH(4)
type CHA42=array(2:CHA4);	LOGICAL *1 X2(4,2)

Note that the character arrays in OPS83 are equivalent to arrays of 1 byte logical type data in FORTRAN.

Transforming data types

A data type transformation is necessary for correctly mapping symbols in OPS83 to character arrays in FORTRAN. Using the two functions (function 'cvsymbol' and function 'name') provided in OPS83, it is possible to convert a symbol into a character string with 127 characters and vice versa. By assigning the long string to a short string, it can be mapped to the character array type defined in the FORTRAN routine. A simple program is given to illustrate how the two functions work.

In the following OPS83 code, the function 'name' converts a symbol &line into a name type data &nline which consists of 127 characters, then the first 20 characters are assigned to the character array &n20. The &n20 array is the device ID which is passed in the subroutine GET_DATA to get the status of the device.

```
procedure start ( )
{
type char20=array(20:char);  -- data structure declare

-- Before a subroutine could be used in an OPS83 program,
-- it must be  declared.
-- The FORTRAN subroutine must be declared as an external
-- procedure, with a specification 'out'.
-- 'out' signifies that value of that variable may be
-- changed in the subroutine.


external procedure GET_DATA(&id:out char20, &st:out char20);

-- For a given device,  GET_DATA gets the status of the
-- device from the database.
```

```
local   -- declare local variables
        &i : integer,                   -- variables are started with "&"
        &j : integer,
        &line : symbol,
        &nline : name,                  -- name is an array of 127 chars
        &n20 : char20,
        &status: char20,
        &L : logical,
        &ns: name,
        &ss : symbol;

&line=|NAME_DEVICE|;                    -- assignment statement

-- The function 'name' has two parameters.
-- It converts symbol &line into 'name' typed data &nline.
-- Integer &i gives the total number of characters in
-- the symbol &line.

&i = name(&nline, &line);

-- Then the name typed data &nline is reassigned to &n20,
-- an array of 20 characters.

for &j = (1 to &i)
   &n20[&j] = &nline[&j];

for &j = (&i to 20)
   &n20[&j] = ' ';

-- For a given device,  GET_DATA gets status of the device
-- from the database.

call GET_DATA(&n20, &status);
```

```
-- the following code converts the character
-- array &status into a symbol &ss.
-- The 'for' loop converts the array with 20 character
-- &status into &ns, a 'name' typed data:

for &j = (1 to 20)
   if ( &status[&j] <> ' ')
    &ns[&j] = &status[&j]
   else
    &ns[&j] = '\0';  -- '\0' means blank filled in &ns

&L = cvsymbol(&ss, &ns);

-- function cvsymbol converts name typed data &ns
-- into a symbol typed data &ss. When cvsymbol sees the
-- first blank '\0', it knows that the string ends there.
-- If the conversion is successful then &L is true (1b).
-- Otherwise &L is false (0b).
-- &ss is the status of the device.
};  -- end procedure
```

Discussion

OPS83 passes parameters by reference; that is, it passes the addresses of the parameters rather than the values of the parameters. This is the same convention that many other languages such as FORTRAN use.

In order to successfully use routines written in other languages one must ensure that the linkage conventions used by the two languages are compatible. In general, OPS83 always uses the same linkage convention as C on the same machine. Hence OPS83 can always call C routines (and vice versa) as well as other languages that are compatible with C. The object codes must be linked to the OPS83 real time library, C real time library and database access routine library to form an executable file.

Appendix C

A Puget Power Line: TalbotHill_Asbury

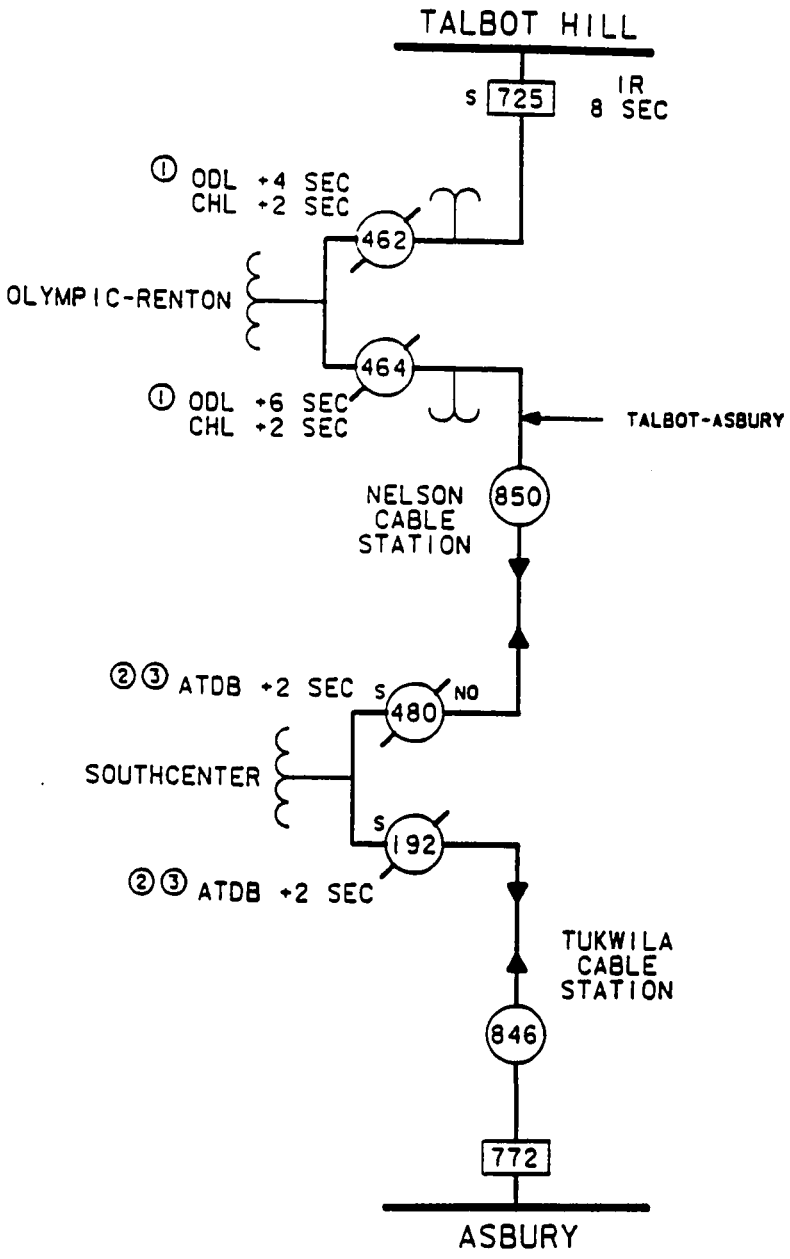


Figure C-1. A Puget Power Line: TalbotHill_Asbury

APPENDIX D

Examples of using the PROCMAN routines

In programs running in the HABITAT environment, it is possible to control the execution of other programs. For example, you can start other programs, put them to sleep, wake them up, stop them, delay their execution or run them periodically according to a schedule.

There are two ways to control the execution of other programs. One way is to use the HABITAT task manager routines such as TASKRUN in your code. These routines are useful, but they increase program complexity and maintenance requirements.

Another way to control programs is to use PROCMAN. PROCMAN simplifies programs by eliminating the need for task manager routines. In the PROCMAN environment, the functions of task manager routines are in a database, not in your program. To control your programs in the PROCMAN environment, you can use five PROCMAN utility functions and one subroutine which are described below.

PROCMAN_DEFINE (taskname)

This function informs the process manager that a specified task has started. The parameter 'taskname' is a character string that must be specified when the routine is called. It returns TRUE or FALSE depending on if the call is successful or not.

PROCMAN_NODE (path, node)

It informs the process manager that a task has completed processing at a specified node or has completed its startup sequence. The parameter "path" is a string of 10 characters and the "node" is a string of 8 characters. Upon completion, PROCMAN_NODE returns a value of TRUE or FALSE. A return of TRUE indicates that the calling program has work to do at the node defined by the "node" parameter. A return of FALSE implies that the calling program should exit.

PROCMAN_SETFLAG (flagname)

This function sets to TRUE a specified flag in the PROCMAN database. The parameter 'flagname' is a character string. It must be specified when the routine is called.

PROCMAN_RESETFLAG (flagname)

This function sets to FALSE a specified flag in the PROCMAN database. The parameter 'flagname' must be specified when the routine is called.

PROCMAN_TESTFLAG (flagname)

This function determines the state of a specified flag in the PROCMAN database.

PROCMAN_SETSKED (name_schedule, period, time)

This subroutine modifies a schedule in the PROCMAN database. The parameter schedule_name is a string of characters. It must be specified when the routine is called. The other two parameters "period" and "time" are optional depending on how you want to modify an existing schedule. The integer "period" specifies, in seconds, the interval at which a scheduled event is to occur. The integer "time" specifies the earliest time a scheduled event is to occur.

In a program, you execute a task defined in a PROCMAN database by using the PROCMAN_DEFINE and PROCMAN_NODE functions. For example, to execute a task called ES, first, you define ES as the task name of your program's process:

```
CALL PROCMAN_DEFINE ('ES')
```

or by issuing a command:

```
STATUS = PROCMAN_DEFINE ('ES')
```

If the task was not found in the database, the call failed. The value of the function return is assigned to STATUS which can be used for debugging purposes.

After defining the task, you call PROCMAN_NODE. It starts the task, identifies the node it is at and passes the node name back to your program.

The node name is then used in your program to determine if any processing has to be done at that node. If something needs to be done, it executes the appropriate code, then returns to the PROCMAN_NODE when processing is complete.

PROCMAN_NODE is usually called in a loop. When your program returns to PROCMAN_NODE, PROCMAN_NODE notifies the database that the processing at the node is complete and determines the next path to take. After identifying the proper path and selecting the next node, PROCMAN_NODE passes the node name to your program again.

The first example of using the PROCMAN routines is a FORTRAN program. It shows the standard order in which PROCMAN_DEFINE, PROCMAN_NODE and other relevant routines are called. It is assumed that a task named 'ES' and a node named 'SOLUTION' as well as the associated paths and flags have already been defined in the PROCMAN database. The paragraphs headed by dashed line are explanations.

PROGRAM ES_MAIN

CALL SYNC_ABORT('ES')

-- This routine is called at the very beginning of the main program to insure that only one process with the name 'ES' is running in the current context of the application family.

CALL PROCMAN_DEFINE('ES')

-- This routine must be called before calling PROCMAN_NODE. After calling PROCMAN_DEFINE, you can map to your database and declare your own exit handlers.

CALL ACCESS_DATABASE()

-- This routine opens all necessary databases for task 'ES'. It is recommended that this routine be called after calling PROCMAN_DEFINE and before calling PROCMAN_NODE.

DO WHILE (PROCMAN_NODE (PATH, NODE))

-- The function PROCMAN_NODE is usually called in a DO loop. Whenever the function is evaluated to be true, the program decides what routines to run based on the node it is currently at.

```
  IF ( NODE .EQ. 'MONITOR' ) THEN
    CALL VOLT_MONITOR (VIOLATION)
                -- to monitor voltage profile problems.
  ELSE IF ( NODE .EQ. 'SOLUTION' ) THEN
    CALL VC_SOLUTION()
                -- to run the voltage control routine.
  ELSE IF ( NODE .EQ. 'CLEANUP' ) THEN
    CALL ...        -- to run cleanup routine.
  END IF
END DO
```

The second example is an OPS83 main program in which the additional interface routines are called to access the PROCMAN routines. This OPS83 code implements almost the same control logic as presented in Example 1. In addition, the calls to set or reset the control flags are given.

As indicated in Appendix B, the FORTRAN subroutines to be directly called in the OPS83 program must be predefined as external procedures. The data types of the parameters between the OPS83 calling procedure and the called FORTRAN routine must agree. Translators and additional interface subroutines are needed to map the data correctly.

First, a list of the additional interface subroutines and OPS83 procedures is presented. Then a main OPS83 module is given.

ABORT_ES();

-- This routine is designed to be called in an OPS83 module. It, in turn, calls SYNC_ABORT('ES'). Since no parameter is involved in the routine ABORT_ES, no data translation is needed in the OPS83 module.

DEFINE_ES();

-- This routine is to call PROCMAN_DEFINE ('ES').

ACCESS_DATABASE();

-- All necessary databases for the task are opened when this routine is called.

PROCMAN_ND(&path, &node, &test)

-- The variable &path is an array of 10 characters, &node is an array of 8 characters, and &test is an integer variable. This routine calls the function PROCMAN_NODE(PATH, NODE). The function value of PROCMAN_NODE, is transferred to &test.

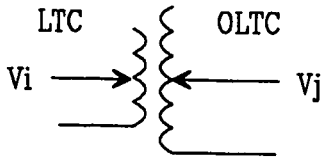
```
function procmannode(&sp:out symbol, &sn:out symbol):logical
        -- PROCMAN_ND is called in this function which performs the same function
        as PROCMAN_NODE. Note that, &sp and &sn are Symbols. Thus, data
        translators are necessary in this OPS83 function. Refer to Appendix B for the
        data mapping technique.

SETFLAG_SOLUTION();
        -- No argument is involved in this routine. This routine calls
        PROCMAN_SETFLAG('ES_SOLVING') to set the condition flag
        'ES_SOLVING' to TRUE.

RESETFLAG_SOLUTION();
        -- This routine calls PROCMAN_RESETFLAG('ES_SOLVING') to set the
        condition flag 'ES_SOLVING' to FALSE.

module es_main(start)
{
        -- This is an example of an OPS83 main program.
        -- First, it declares the data structure, the modules and the external procedures
        it uses.

use data_definitions in |datadef_module|;
use inference in |inference_module|;
use vces in |vces_rules_module|;

external procedure DEFINE_ES();
external procedure PROCMAN_ND(&path:out c10_array, &node:out c8_array,
        &test:out integer);
external procedure ABORT_ES();
external procedure ACCESS_DATABASE();
external procedure VOLT_MONITOR( &violate: out integer );
external procedure SETFLAG_SOLUTION();
external procedure RESETFLAG_SOLUTION();
external procedure COMPUTE_SENSITIVITY_MATRIX();
```

```
procedure start()
{local .....        -- local data type definition
call ABORT_ES();
call DEFINE_ES();     -- calls PROCMAN_DEFINE('ES')
call ACCESS_DATABASE();
                      -- open all databases needed for task ES.
while ( procmannode(&spath, &snode) )
{                     -- This is equivalent to a do loop.
if ( &snode = |SOLUTION| )
  {
  call SETFLAG_SOLUTION();
        -- The flag 'ES_SOLVING' is set to TRUE which prevents the control path to
        the node being executed during the ES solution process.

  call VOLT_MONITOR( &violation );
        -- It accesses the SCADA database to monitor the voltage problems. The
        parameter &violation is the total number of violated buses.

  if ( &violation > 0 )
   { call COMPUTE_SENSITIVITY_MATRIX();
    make ( task  id = begin;);
                      -- The rule base execution starts with
                      -- making this task element.
    call run();
                      -- run VCES rule base system.
   }                  -- end if (&violation > 0 )
        else          -- There is no voltage violation problem.
  call RESETFLAG_SOLUTION();
                      -- reset the 'SOLUTION' flag to FALSE
  };                  -- end if ( &snode = |SOLUTION| )
};                    -- end while
};                    -- end of procedure start()
};                    -- end of module es_main(start)
```

Appendix E

Transformer model and sensitivity computation

Two-Winding Transformer and Its Model:



lossless on-load tap (LTC)

Vi = Ti * Vk

where Ti is complex

lossless off-load tap (OLTC)

Vj = Tj * Vl

where Tj is scalar

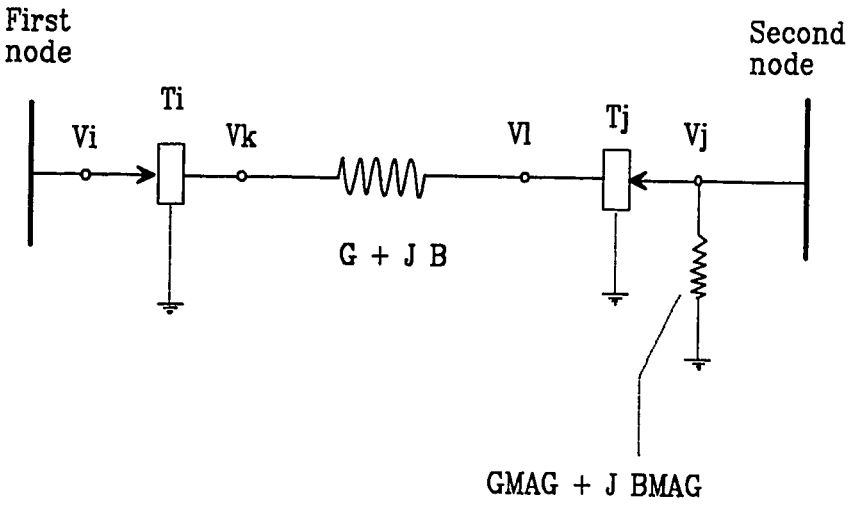


Figure E-1. Two-Winding Transformer Model

A list of the terms and parameters related to the tap transformer model [37], and to the computation is given below:

XF = transformer

TAP_XF is the normal tap number on the LTC.

ZTAP_XF is the normal tap number on the OLTC.

P_XF is the real power (MW) flow into the XF.

Q_XF is the reactive power (MVAR) flow into the XF.

ST_XF is the sensitivity of the tap voltage to the transformer tap ratio, i.e. (p.u. volt / p.u. tap ratio).

R_XF is the resistance of the XF.

X_XF is the reactance of the XF.

G_XF is the conductance.

B_XF is the susceptance.

KVNOM_XF is the nominal KV of the XF on the LTC side. Note that, KVNOM_XF is not necessarily equal to KVNOM_VL, the nominal voltage of the system on the LTC side.

ZKVNOM_XF is the nominal KV of the opposite side (the OTLC side).

GMAG_XF is the per unit shunt magnetizing conductance.

BMAG_XF is the per unit shunt magnetizing susceptance.

STEP_TAPTY is the step size of the tap in per unit.

NON_TAPTY is the nominal tap number.

MN_TAPTY is the lowest tap number, e.g. -16.

MX_TAPTY is the highest tap number, e.g. 16.

T_XF is the per unit tap ratio of the on-load tap.

T_XF = 1/Ti

TZ_XF is the per unit tap ratio of the off-load tap.

TZ_XF = 1/Tj

Figure E-1 shows the two-winding tap transformer model. It consists of an on-load tap, an admittance, an off-load tap and a magnetizing admittance. The steady state power flow equations for the transformer are:

$$Pi(Vl,Vk) = |Vk|^2 * G - |Vk| * |Vl| * [G * \cos (Ak - Al) + B * \sin (Ak - Al)] \quad \text{(E-1)}$$

$$Qi (Vl,Vk) = |Vk|^2 * B - |Vk| * |Vl| * [G * \sin (Ak - Al) - B * \cos (Ak - Al)] \quad \text{(E-2)}$$

where Pi, Qi are the real and reactive power flows out of node i to node k . The quantities G and B correspond to the conductance G_XF and susceptance B_XF of the transformer. The quantity |Vi| indicates the voltage magnitude at node i and the Ai indicates the voltage angle at node i.

The flow out of node j to node i is given by:

$$Pj(Vl,Vk) = |Vl|^2 * G - |Vl| * |Vk| * [G * \cos (Al - Ak) + B * \sin (Al - Ak)] \quad \text{(E-3)}$$

$$Qj(Vl,Vk) = |Vl|^2 * B - |Vl| * |Vk| * [G * \sin (Al - Ak) - B * \cos (Al - Ak)] \quad \text{(E-4)}$$

At the on-load tap, the voltages are transformed according to:

$$|Vi| = |Vk| * |Ti| \quad \text{(E-5)}$$

where:

$$|Ti| = [1.0 + STEP\_TAPTY * (TAP\_XF - NOM\_TAPTY)] * KVNOM\_XF / NOMKV\_VL \quad \text{(E-6)}$$

At the off-load tap, the voltages are transformed according to:

$$|Vj| = |Vl| * |Tj| \qquad \text{(E-7)}$$

where:

$$|Tj| = [\ 1.0 + STEP\_TAPTY * (\ ZTAP\_XF - NOM\_TAPTY\ )\ ] * ZKVNOM\_XF / NOMKV\_VL \qquad \text{(E-8)}$$

In order to obtain the sensitivities of the other buses to a tap change, it is needed to find the variations of reactive power at node i and node j, dQi and dQj, due to a step change of the tap at the LTC side (i.e. dTi<>0, dTj=0). Applying the derivative chain rule to Eqs. (E-2), (E-4), (E-5), (E-6), (E-7), (E-8), the dQi and dQj can be computed as:

$$dQi = (dQi/ d|Vl|) * (d|Vl| / d|Ti|) * (d|Ti|/ d(STEP\_TAPTY)) + (dQi/ d|Vk|) * (d|Vk| / d|Ti|) * (d|Ti|/ d(STEP\_TAPTY)) \qquad \text{(E-9)}$$

$$dQj = (dQj/ d|Vl|) * (d|Vl| / d|Ti|) * (d|Ti|/ d(STEP\_TAPTY)) + (dQj / d|Vk|) * (d|Vk| / d|Ti|) * (d|Ti|/ d(STEP\_TAPTY)) \qquad \text{(E-10)}$$

The values of dQi and dQj are then put into the right-hand side of Eq.(4.3-3), [J] $d\underline{X} = d\underline{Y}$, with all other control variables set to 0. Hence, Eq.(4.3-3) can be solved for $d\underline{X}$ which contains $d\underline{V}$, a vector of the variations of the load bus voltages due to the step change of the tap.

For a load bus m, the voltage deviation scalar dVm can be identified from the vector $d\underline{V}$. Finally, the sensitivity defined in (4.3-7) is obtained by multiplying dVm with a normalization factor (1.0 / Vm).

Vita

Dr. Ming Chen, was born on March 10th, 1946 in Fuzhou, China. His middle school and high school was the Affiliated Middle School of the Fujian Normal College, Fuzhou, China. He received the B.S. degree from the Radio-electronics Department, China University of Science and Technology in 1968. After graduation, he worked for 3 years as an engineer in carrier broadcasting. Then, he engaged in the development of silicon monocrystal, silicon controlled rectifier and its applications in industry. He also lectured the power electronics course in the Fujian College of Water Conservancy and Electric Power, studied in the graduate school of the Beijing Institute of Post and Telecommunication, worked for the Fujian Electric Power Test and Research Institute.

After 13 years work experiences in China, he came to the U.S. and started a graduate program in the Electrical Engineering Department, University of Washington. He received the M.S.E.E degree in 1984, and the Ph.D degree in 1989. His recent research interest includes the expert systems applications to power systems, the database management system, and the software integration.

His major publications are:

Ming Chen, "The Integration of Expert Systems into Energy Management System Centers Using a Dispatcher Training Simulator," Ph.D. Dissertation. University of Washington. Seattle, Nov. 1989.

Ming Chen and M. J. Damborg, "Full Integration of an Expert System into an Energy Management System Using a Dispatcher Training Simulator." Proceedings of the Second Symposium of Expert Systems Application to Power Systems. Seattle, Jul. 1989.

M. J. Damborg and Ming Chen, "An example of integrating an expert system into a control center using a dispatcher training simulator." Proceedings of the First Symposium of Expert Systems Application in Power Systems. Stockholm, Sweden. Aug. 1988.

Ming Chen, "On the Light Weight Electrical Defibrillator." M.S.E.E. Thesis, University of Washington. Dec. 1983.