DIALOG: A SYSTEM FOR INSTRUMENTATION DIAGNOSIS

A. Burns, J. Menu
SPS Division, CERN
1211 Geneva 23
Switzerland.

The last upgrade of the SPS accelerator to become an injector for LEP led to an even more complex machine with the implementation of cycle to cycle modulation ('multicycling') and the increased use of microprocessors. This reinforced the need for an automated diagnosis system for instrumentation.

The Dialog system provides the required functionality, relying on programs residing in the equipment to be diagnosed and others distributed over the SPS network. The diagnosis is conducted by a supervisor running on a personal computer that interacts in real-time with the instrumentation equipment.

The system is able to diagnose multiple crates of the same type at once. In case of failures, corrective actions can be specified, including the location of spares which is fetched from a database.

1. Diagnosis of SPS instrumentation equipment faults

The beam instrumentation group at the SPS is responsible for the development and maintenance of a considerable variety of accelerator instrumentation. Much of this equipment is in daily operational use and an on-call repair service ('piquet') has to be provided 24 hours a day during periods when the accelerator is running. All members of the group participate in this 'piquet' and, therefore, it is most likely that the person called out is not the expert responsible for the equipment. He must try to repair the equipment using documentation provided by the specialist. This documentation includes instructions on how to identify possible faults and what corrective action should be taken, as well as the locations of the equipment crates and spares.

At present, equipment diagnostics can be obtained from:

- programs written by the expert for use mainly in the buildings housing the equipment (known as 'BA's), These programs often contain many facilities for the control and testing of the equipment, the details of which may be confusing to non-specialists;
- main control room console programs written by the operations group or sometimes the equipment specialist that may display some status information or report error conditions. These programs are not available in the BAs;
- visual signals on the equipment itself.

The programs are written in the interpreted language NODAL developed at CERN for accelerator control. They are easily modified from any terminal on the control network, but lack modularity and mix machine specification data and the test procedures themselves.

The introduction of multicycling at the SPS [1], in preparation for the use of the SPS as injector for LEP, has initiated an extensive program to upgrade the main instrumentation systems. Multi-board microprocessor systems using the VME or G64 backplane bus standards and connected to the MIL-1553B multidrop bus for communication are replacing the original, generally non-intelligent, systems. Already the function generators, the closed orbit acquisition system, and the electronics for the secondary emission monitors and the emergency beam loss detectors have been replaced; while the ring beam monitors, beam current transformers, television screens, wire scanners, synchrotron light detectors and dampers will all be converted in the next 1-2 years. All the new systems will be VME systems running 68000 family microprocessors.

This continuing conversion has made possible the acquisition of more diagnostic information by software and also permitted the imposition of a standard microprocessor core software within the instrumentation group (including multicycle handling, communication, treatment of resident test programs). This is turn has aided the development of a new automated diagnosis system for instrumentation equipment according to the following criteria:

- The system would be required to assist initial interventions on essential equipment by the instrumentation 'piquet' and, after a trial period, by the machine operators themselves.
- The system should be available in the instrumentation group laboratories and offices, in the BAs and in the main control room.

- The system could not reasonably be expected to correctly interpret all possible faults, and therefore the intervention of the relevant expert may still be needed in some cases, as at present.

2. System architecture

It was decided early in the project that diagnosis should be done in a hierarchical way. The system is composed of:

- a **supervisor** of the diagnosis, running on a personal computer;
- **test programs**, which are either resident in the equipment to be diagnosed, or NODAL programs stored in a library on the SPS control network.

The resident test programs can be written in any language by the designer of the given equipment. They communicate their results back to the supervisor by means of a **text page**, that is stored in a file in the supervisor's directory and may optionally be displayed to the user of the system.

In order for Dialog to be able to use the contents of the text page, it must be told how to do so. This is done by two kinds of information supplied by the designer of the equipment:

- a **page format** describing the structure of the page, in the form of grammar rules;
- a set of **diagnosis rules** that will tell Dialog which values to look for in the corresponding page, and what to do whenever they satisfy given boolean conditions.

Both page formats and diagnosis rules are specified in the dedicated language Dialog [2].

An example of a page format is shown below:

```
page_format( sem_status ) -->
  comment_lines(2),
    data_lines(
          computer = ascii(5),                blanks(1),
          n_value = decimal(3),               blanks(1),
          rt = decimal(2),                    blanks(2),
          tg3_status = ascii(3),              blanks(4),
          m1553_interface = ascii(3),         blanks(4),
          asynchr_acquisition = ascii(3),     blanks(3),
          acquisition_mode = ascii(6),        blanks(4),
          timing_interface = ascii(3),        blanks(5),
          supercycle_number = decimal(5),     blanks(4),
          first_acquisition = decimal(5),     blanks(4),
          second_acquisition = decimal(5)
          ).
```

A sample diagnosis rule on this text page is:

```
diagnosis_rule( sem_status ) -->
  if whatever_data_line timing_interface = 'BAD' then
     external_failure( 'No timing in all crates',
                ' in this sub-system', nl,
                '-- check Master Timing Generator')

  elsif exists_data_line timing_interface= 'BAD' then
     component_failure( timing_interface, 'No',
                ' timing being received', nl,
                '-- check distribution or change',
                ' interface card')
  endif.
```

For the purposes of diagnosis, each system is viewed as a set of so-called **objects**, that can be hardware as well as software components of the equipment. Test programs are supplied by the designer of the equipment for most of these objects together with the Dialog page formats and diagnosis rules for them.

Each test program tests only the object it has been designed for: there are no means in Dialog to test several objects at once. However a test program may well lead to another object being suspected, as in:

```
diagnosis_rule( sem_status ) -->
  if exists_data_line m1553_interface = 'BAD' then
     suspect m1553
  endif.
```

The system can be used via a direct RS232 connection to the equipment crate or via the SPS control network. In the latter case, a standard prologue is first launched to help select the subset of crates to be diagnosed.

3. The implementation

The key point for a specification language to meet its promises is **safety**. In the case of Dialog, we cannot afford to let people write down anything without full checking: in particular, page formats and diagnosis rules are useless if they cannot be guaranteed consistent.

The implementation is thus organized as follows:

- the **compiler** reads Dialog source specifications from files, performing extensive checks on it, and produces so-called compiled code onto other files;
- the **interpreter** loads that compiled code and executes it to conduct the diagnosis;
- since some objects can be components of more than one type of equipment, the files describing them are compiled **separately**, providing modularity. Such a file is recognized by the compiler as starting by a 'test_programs' specification;

- when a file describing an equipment is compiled, the compiled code for its component objects is automatically loaded and **linked** together with the compiled code for the given equipment. An equipment file is distinguished as starting by a 'components' specification, and the next element in it should be an 'investigate' element. Different load modules are generated for local or remote use (includes crate selection prologue if remote)

4. Running the diagnostic system

First, the personal computer, a Macintosh, is connected by the serial line directly to the crate (local mode) or logged in to NODAL on a NORD-100 on the control network (remote mode) Then, the appropriate local or remote executable file is loaded into Dialog and run.

In the case of remote mode, interaction with a special equipment data-base allows the selection of a particular sub-set of crates to be diagnosed. The initial status program specified in the Dialog file will then be run on the chosen crates, and then, depending on the status page received, test programs may be run in individual crates.

In local mode, a resident status program will run in the local crate, which in turn may trigger other test programs, as in the remote case.

By default, a minimum of information is displayed to the user and the diagnosis proceeds automatically, informing the user of its progress by messages about which test programs are launched and possible failures it finds.

If Dialog finds a failure in some equipment, it will display the corrective action to be taken, and in remote mode will interrogate a spares data-base to display information on the location of the equipment crate and the nearest spares.

A verbose mode may also be selected, in which all text pages are displayed as they are received, and in which it is possible to launch manually test programs known to the the system but not yet run. This mode is useful during development.

5. Current status and perspectives

A prototype Dialog system has been implemented on a Macintosh, which satisfies the requirement for a fast portable machine. The compiler and interpreter have been written in Prolog; the communication primitives have been written in Pascal and added to Prolog as predefined predicates. This version is entirely adequate for the development of diagnostic programs by the equipment designer, but, being a development environment, is not really suitable for use by the occasional user (i.e. 'piquet' or operation crew).

Current work is therefore going on to provide an autonomous Macintosh application that can be run without the Prolog development system.

Although the Macintosh can be used in the main control room while the present NORD-100 computers remain, the decision has been taken to port the Dialog system to the Apollo workstations, which will gradually replace the existing NORD-100 consoles. The proposal to provide a general purpose Apollo also in each BA will then allow running the same diagnostic system locally without transporting the Macintosh and using the RS-232 connection to the crate.

Currently, Dialog files exist only for the closed orbit and secondary emission monitor acquisition systems and they do not yet take into account all the possible diagnostics available from the equipment. Work to provide diagnostic programs for the other microprocessor-based instrumentation systems will start when the current level of pressure to provide new and improved instruments eases off.

It is with pleasure that the authors acknowledge the contributions of L. Burnod and E. D'Amico in the early stages of the project.

References

[1] G. Beetham, L. Burnod, R. Lauckner, J. Miles, G. Mugnai, C. Saltmarsh, D. Thomas, "Conversion of the SPS to a Multicycling Accelerator", presented at this conference.

[2] A. Burns, E. D'Amico, J. Menu, "DIALOG: A Language for Instrumentation Diagnosis", presented at the Journées sur les applications de l'IA en Physique des Particules, CPPM, Marseille, June 15-17, 1987.