Nuclear Instruments and Methods in Physics Research A293 (1990) 368–372
North-Holland

SOFTWARE DEVELOPMENT AND MAINTENANCE: AN APPROACH FOR A LARGE ACCELERATOR CONTROL SYSTEM

L. CASALEGNO, L. ORSINI and C.H. SICARD

PS Division, CERN, CH-1211 Geneva 23, Switzerland

Maintenance costs presently form a large part of the total life-cycle cost of a software system. In case of large systems, while the costs of eliminating bugs, fixing analysis and design errors and introducing updates must be taken into account, the coherence of the system as a whole must be maintained while its parts are evolving independently. The need to devise and supply tools to aid programmers in housekeeping and updating has been strongly felt in the case of the LEP preinjector control system. A set of utilities has been implemented to create a safe interface between the programmers and the files containing the control software. Through this interface consistent naming schemes, common compiling and object-building procedures can be enforced, so that development and maintenance staff need not be concerned with the details of executable code generation. Procedures have been built to verify the consistency, generate maintenance diagnostics and automatically update object and executable files, taking into account multiple releases and versions. The tools and the techniques reported in this paper are of general use in the UNIX environment and have already been adopted for other projects.

1. Introduction

The CERN PS accelerator complex consists of several proton accelerators, two antiproton accumulators, two linacs, various transfer lines and the LEP preinjector. All these machines are controlled from a central room through a network of about 25 minicomputers and 100 microprocessors, interfaced to the process hardware through CAMAC. This network includes front-end process computers (FECs) which control subsystems of the accelerators [1]. The LEP preinjector is controlled by two FECs to which about 20 microcomputers are connected. The FECs are linked, through the network, to the computers driving the consoles of the main control room.

Each microcomputer (called SMACC) contains an MC68000 microprocessor, is located in a CAMAC crate and controls the hardware modules situated in the same crate. The software procedures written to execute the actions requested by the application programmes on the hardware devices have been called control modules. Control modules [2] are basically object-oriented pieces of code composed of (i) a frame that performs the selection mechanism, data access and parameter checking, (ii) procedures and (iii) data specific to the control module. The control module is a general superclass which includes, as subclasses, composite variable modules (CVMs) for beam-variable control, equipment modules (EMs) for remote operations on equipment, and interface modules (IMs) for remote access to hardware modules [3].

The frame of the control module is fixed whereas the procedures and data have to be supplied by the control-module developer. A number of different computers are involved in the process of software generation and execution for the LEP preinjector control, as shown in fig. 1, mainly due to compatibility requirements with the existing PS control system. The IBM main frame supports the Oracle DBMS, a VAX Cluster running ULTRIX is used for cross-development, NorskData minicomputers act as FECs and concentra-

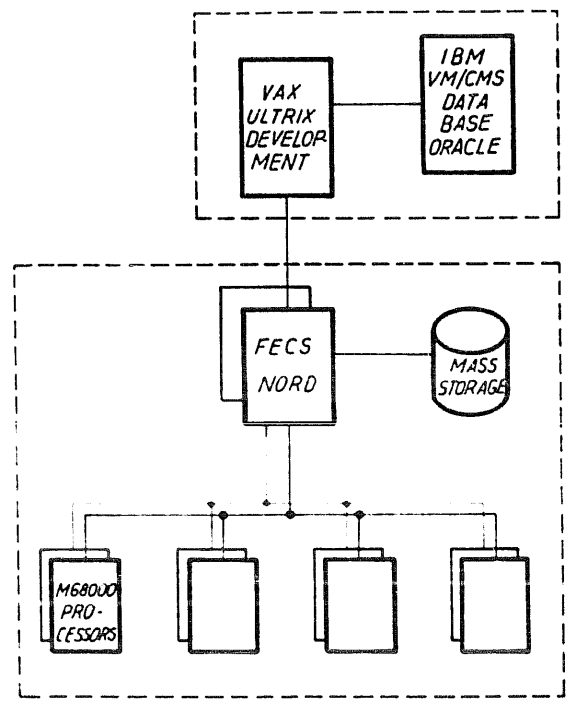

Fig. 1. Computer network configuration.

0168-9002/90/$03.50

tors and MC68000-based SMACCs supervise the equipment control. This configuration has required the creation of a development environment, separated from the execution environment. The development environment supplies compilation, debug, test and maintenance facilities, whereas the execution or target environment ensures compatibility with the existing system and supplies real-time facilities.

During the development of the LEP preinjector control system we were faced with four kinds of problems with regard to control-module development, upgrading and maintenance. It was necessary to:

- create a development environment that is homogeneous as seen by the developer,
- simultaneously maintain different versions of the same frame to allow regular maintenance and new developments in parallel,
- keep track of the files constituting each executable code in the microprocessors to enable easy rebuilding of the code, to cater for upgrading or bug fixing,
- maintain, at the same time, a test application frame, to be used on the development machine, and the frame to be used in all target SMACC processors.

This paper describes the solutions adopted to satisfy these requirements.

2. The development environment

The UNIX operating system has been adopted as the development environment because of the decision to implement the LPI control system using the language C. The application programmers should be mainly concerned with the problem at hand, rather than with the specific properties of the development operating system and its utilities. On the other hand, a complex development environment, such as the one used for the LEP preinjector, required a large number of UNIX facilities and some dedicated procedures built with standard UNIX facilities or "bricks" [4].

This involves some effort in learning the mnemonics used and is prone to error, particularly for the occasional user. In addition, no file naming scheme can be enforced by using simple UNIX commands. A consistent naming scheme is very important for automated maintenance.

Therefore it has been decided to hide the UNIX shell command level from the programmer and to build a menu-driven facility which included all the actions required to develop and test control modules and to build microcomputer images, i.e. the executable code to be loaded into the microprocessors. Fig. 2 shows a menu of the options available. For source code maintenance the Berkeley UNIX tool called SCCS [1,5] has been adopted. Source files can be edited directly from the UNIX source-code control-system repository.

Test case sequences, covering all the code specific to a control module, can be designed and maintained with the code itself, so that if later modifications are made, the test sequences can be run in order to verify the absence of side affects. An on-line help facility is available to give information about each of the menu options.

For all the options on the menu, the user only has to supply a file name, all the syntax of the shell commands being hidden. In addition, the menu acts as a filter and invalid answers are rejected. The occasional user does not need to remember complex commands, and a coherent naming scheme for object code files, make-files and

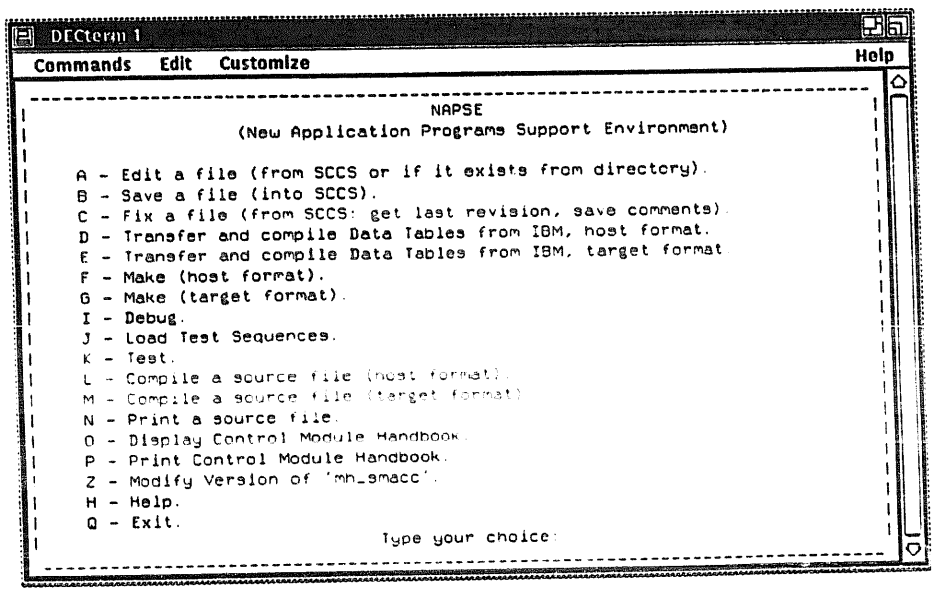

Fig. 2. Development-environment user interface.

debug files has been established. This naming scheme allows the maintenance of the whole control-system software through automated procedures which verify the existence of the required files in each directory and their generation dates.

3. Maintaining the frame

The LEP preinjector control system uses control modules written at different times by different people. Even if the goal is to obtain a stable situation in which all the Control Modules loaded into the microprocessors contain the same version of the frame code, sometimes it happens that different versions coexist, due to enhancements applied to new applications. Homogeneity is re-established, normally during shutdowns, by retrofitting the latest versions to old applications. Thus more than one version of the frame target code has to be available during normal operation.

In order to keep a single source of the code and to avoid including into the code itself conditional compilation statements, a maintenance environment has been set up. The maintenance environment is based on SCCS [1] and a special facility which can record the versions of each source file that constitutes the frame at a certain moment of time. The collection of all the source files and their version number at a certain time is called a base line. By means of the information contained in a base line, any frame version can be restored at any moment, by generating a make-file containing the required version number for each source file. Fig. 3 shows the actions that can be carried out on base lines. A base line can also be edited when a bug has been detected which is liable to affect the behaviour of all the frame versions present in the target microprocessors. The version of the file in which the bug has been found can then be updated.

The version number of a file, as generated by SCCS [1], is composed of a release number and a release version number. When a file is updated, the source-code management system increments the latter number automatically. When the number of modifications is large enough to require a distinct range to mark new versions, a change in the release number can be executed. Then the release number of all the source files is changed and the release-version number set to 1. The base-line numbering system roughly follows the same scheme.

It is also possible to build frame versions from the most recent version of all the files present in the source-code management-system repository, both for the target and the development environment. A help facility is provided to give details of each option in the menu. The scope of the maintenance environment is the directory in which it is loaded. Any directory can have this maintenance environment for maintaining different versions of the same software product, provided the SCCS [1] facility is used. As a matter of fact, the environment has already been used for maintenance of code other than the frame of the control modules. The SCCS system forbids access right to the files being edited on a user-name basis. Unfortunately, our access rights to software code are based on a "per project" user name, so all programmers working on the frame code use the same user name and no file access control could be set up.

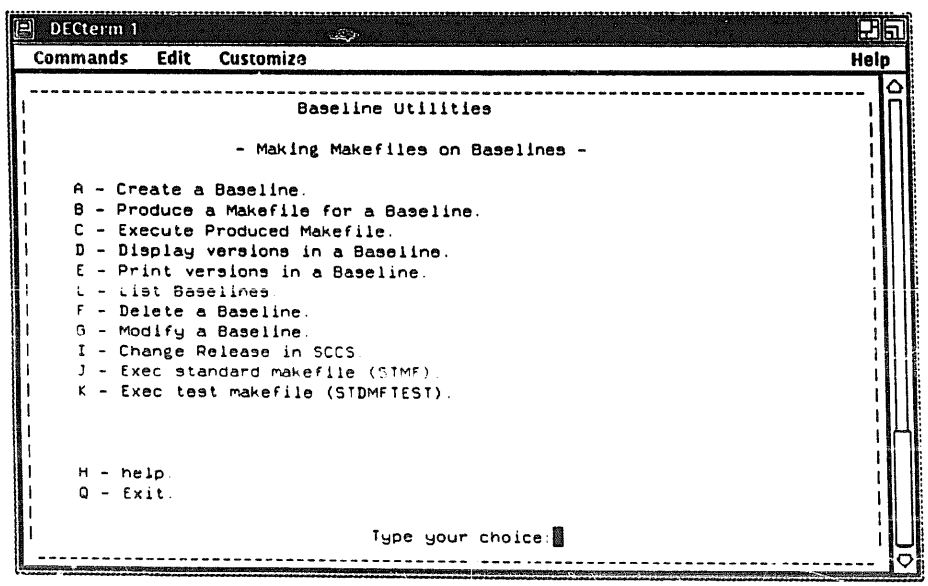

Fig. 3. Frame-maintenance environment user interface.

4. Building make-files for target microprocessors

The ORACLE data base contains all the information needed to create the data structures and the list of method names associated with the properties of each control module [2,6]. It has been decided to add some maintenance information to this operational data. Each method name declared in the data base has associated to it the full name of the file containing the corresponding code. Moreover, a special record in the data base is allocated to hold the names and locations of all files containing applications (normally real-time tasks or diagnostic tasks) to be included into a given microprocessor image.

Building a make-file [3] to generate an image for a microprocessor is not an easy task if one starts from scratch. Many files must be present, even if they are not directly referenced by the application at hand. The frame of the control module and its ancillary files must always be loaded. Moreover, maintaining these make-files can become very difficult when changing the release of general-purpose or frame-related files. A method has been developed to generate make-files from a common stub and information contained in the ORACLE data base.

The make-files are built by means of two stubs. The first stub is common to all make-files and contains the name of the files that must always be loaded into the microprocessors, with their correct release and version number, and the commands for the linkage and image generation phase. This stub provides the commands to send the generated images from the development machine into the FEC repository. A backup repository for SMACC images has been set up in each FEC [6]. The second stub is dynamically created, using the information in the data base. When a new image is built, a procedure first extracts from the data base the full names of all the files containing code specific to the given microprocessor. Then this stub is linked to the first one and the resulting make-file is produced and executed.

5. Maintaining source code for two different targets

The task of maintaining the source code for many different target systems is more general than the scope of the LEP preinjector project. However, for the sake of completeness, some comments about the method used to cope with this problem should be made. As far as possible, no conditional compilation was used, as having the same code in both environments guarantees better test conditions. When incompatibility cases arose, a subset of the C language was chosen so that the code would run on both machines. Instructions present in the language that might have given problems when porting the code, such as "field" [7], were banned. If one behaves carefully, a very portable code can be produced. No compromises in execution speed had to be made and no extraordinary manpower efforts were required. Plans for moving the present frame to a VME environment have been already set in place. No great problems are foreseen for this porting.

6. Future developments

The number of SMACCs in the LEP preinjector control system is rather large and very few SMACCs contain exactly the same core image. Thus many make-files have to be created and maintained. A maintenance make-file will allow selective execution of the make-files for which the target image is out of date. Updating of this maintenance make-file will be made through the information contained in the data base.

The possibility of accessing the ORACLE data base from a remote job, through SQL-NET, will provide for data-table generation into the make-files. The whole PS control system, including the LEP preinjector, is being upgraded by the addition of a network of workstations. The availability of these powerful machines will allow the building of a more sophisticated CASE support. An extended investigation will be made to evaluate the possibility of including the present development and maintenance scheme into the new CASE tools.

7. Conclusions

An effective maintenance environment for widely distributed executable modules produced from the same source code has been set up by hiding the plain UNIX commands under a layer of menu-driven facilities. The intrinsic structure of the UNIX operating system and C shell command language, together with the availability of a number of bricks suitable for building larger utilities, well matched the design requirements for such an environment. The menu-driven facility for control-module building can be considered as a basis for any application-development environment, because it contains all the actions necessary to obtain a software product:

- writing and maintaining the data part,
- creating and maintaining the code part,
- debugging and testing the product,
- creating the executable modules
- documenting the product, and
- helping the user in navigating through the different options by means of a context-sensitive help facility.

The acceptance of the development environment by the programmers was quite positive and they generally remarked on an increase in reliability and a reduced loss of time for occasional users. The tool SCCS [1] and

base-lining were found very useful to follow up the modifications and the upgrade of the frame code. Tests of new ideas could be performed safely without the need to keep multiple copies of the same file; wrong modifications were easily discarded. The control-module frame maintenance environment is now the only method used to update the control-module frame code. This experience encourages the development of this approach to the software workbench, which will integrate the new workstation tools.

Acknowledgement

The authors would like to thank Ana Paula Pereira for her helpful participation in the early phase of this project and for the production of the backbone of the development environment.

References

[1] L. Casalegno et al., Distributed Application Software Architecture Applied to the LEP Preinjector Controls, presented at the 7th IFAC workshop on Distributed Control Systems, Mayschoss/Bad Neuenahr, FRG, September 30–October 2, 1986.

[2] L. Casalegno et al., Building Software Modules for Driving Hardware Controlling Physical Variables: An Object-Oriented Approach, CERN/PS 87-81 (CO), presented at the First Int. Conf. on Accelerators and Large Experimental Physics Control Systems, Villars sur Ollon, Switzerland, 1987, CERN Yellow Report, to be published.

[3] L. Casalegno et al., Control Modules Handbook, PS/CO/Note 88-007 (1988).

[4] B.W. Kernighan and R. Pike, The UNIX Programming Environment, Prentice–Hall Software Series (1984).

[5] E. Allman, An Introduction to the Source Code Control System, Project Ingres, UNIX supplementary documents (University of California at Berkeley).

[6] L. Casalegno et al., these Proceedings (Int. Conf. on Accelerator and Large Experimental Physics Control Systems, Vancouver, BC, Canada, 1989) Nucl. Instr. and Meth. A293 (1990) 368.

[7] B.W. Kernighan and D.M. Ritchie, The C Programming Language, Prentice–Hall Software Series (1978).