EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN/PS 89-67 (CO)
15.12.1989

PROCESS EQUIPMENT DATA ORGANISATION IN CERN PS CONTROLS

L. CASALEGNO, J. CUPERUS and C.H. SICARD

Paper presented at the International Conference on
Accelerator and Large Experimental Physics
Control Systems
Vancouver, British Columbia, Canada
October 30 - November 3, 1989

Geneva, Switzerland

PROCESS EQUIPMENT DATA ORGANISATION IN CERN PS CONTROLS

L. Casalegno, J. Cuperus and C.H.Sicard

PS Division, CERN, CH 1211 Geneva 23, Switzerland

The CERN PS Control system has a widely distributed architecture, mainly for fast response in a real-time environment. The organisation of the data for equipment access must be compatible with this architecture and give efficient program access to the data. Moreover, it must also offer managerial features such as data integrity, easy backup and restore, adaptibility to changes in data structure, initialisation, data-entry facilities and automatic documentation. This paper shows how one can take advantage of a commercial database management system with its associated tools, adding to it some object-oriented programming concepts to meet the objectives of a manageable distributed data organisation having good run-time performance features and using a reasonable manpower investment.

1. Introduction

The controls for the PS accelerator complex have from the beginning accessed the accelerator equipment through a unique call interface ("Equipment Module") and data format (by classes of equipment, one data record per piece of equipment of the same class) [1]. The action to be executed is indicated by a "property" which causes a set of statements, the "property-code", to be executed.

The original system, installed in 1980, was designed with performance in

mind. It provided the programmer with a code template, which he filled with a data-record description and a property-code, which accessed the data directly (via record pointers). An additional facility was a small off-line database containing the list of equipment names and their hardware device mapping.

Experience with this system showed that, although programming was rather simple once the template was well known, several maintenance problems occured:

- the absence of data protection, together with the use of languages offering little static checking, caused operational data corruption problems which were very difficult to clean up.
- operational data, stored in computer memory/disk segments, had to be preserved, but its initialisation and restoring after cold-starts or segment corruption was not an integral part of the system, and was often overlooked by the original programmer. Specific programs had to be written for each module, to be called manually in case of problems.
- software module maintenance on templates "customised" by programmers was found difficult.
- documentation was hand-written, thus often provided late and incomplete to users.

2. Design objectives for an improved system

The prime objective is to minimise system maintenance, as changes in the equipment lists and functionality are typical of evolving accelerators such as the PS complex.

- Documentation is to be automated as much as possible, in particular for obtaining up-to-date lists of equipment, with their main characteristics.
- Operational data protection is to be improved against side effects

of property-codes or real-time tasks.

- The source-code template is replaced by a kernel, hidden from the programmer.

- A large amount of the data used in Equipment Modules, such as scaling factors, ranges, hardware adresses, was provided beforehand by the equipment specialists. These data do not need frequent updates but are needed in the documentation which should be taken from a unique source.

- For easy implementation of generalised functions such as data browsing or saving/restoring, the structural information (record description for each module) is to be loaded with the on-line data.

- All generalised functions must be guaranteed to work for all modules, with easy upgrades for new features.

- The more decentralised architecture of the control system and the increased local processing power results in the need for an extended call interface, providing integer and string types in addition to the normal floating-point type (the first for performance reasons, and the second mainly for customised messages).

3. Implementation Constraints

The Control Modules (generic name for Equipment Modules in this new generation) are loaded into embedded microprocessors, which offer no disk storage nor remote file access. In case of hardware failure, all operational data must be restored by down-loading the latest settings.

For an easier upgrade to another architecture, all access to Control Module functions is done via remote procedure calls (direct access to microprocessor memory, although much more efficient, is thus ruled out).

Equipment instances are defined statically (from definitions in the central database), and no auto-configuration is done at the front-end

levels.

The ORACLE relational database management system, running on a central IBM computer, was chosen for off line data storage. Data entry is exclusively by means of the ORACLE tool SQL*Forms.

The generation of the operational modules and their installation on the target processors must be as automatic as possible; data consistency and completeness must be checked.

4. Resulting implementation

Data and code are separated as much as possible. To attain this, a complete module consists of 3 parts:

Frame: a kernel code identical for all module classes. The frame is data driven.

Data: which give specificity to the frame. We distinguish class data, which define the module class (properties, column names etc...), and instance data which define values for individual members of the class (e.g. hardware addresses or maximum values for each piece of equipment.

Procos: (property-codes) subroutines which execute specific actions. These subroutines are normally short.

The method chosen for accessing data is to replace record pointers by a small set of data-access routines, which are the only interface for application programmer code. These routines must still allow fast real-time response typical of the PS (i.e, updating up to 40 set-points every 1.2 second, in a time window of about 10ms).

Performance aspects: Control Modules can be called for an array of equipment of the same class. For fast access, the Proco calls the data access routines also in array mode. This is satisfactory for simple actions

such as loading data into the equipment on a pulse-to-pulse basis. For more complex operations, the frame calls the Proco in a loop, once for each piece of equipment. We thus have two kinds of Procos: Array-Procos which handle an array of equipment and have a fixed calling sequence and Loop-Procos which handle one piece of equipment at a time and are called, by the Frame, with a user-defined record containing the variables used by this Proco. Array-Procos are faster but Loop-Procos are simpler to write and allow more complex operations. Relative performance of these two methods has been measured, showing the usefulness of the array method.

Data Initialization: the control system data is divided into Equipment Data (Read-Only) and Operational Data (Read-Write) : the first are entered through the Oracle Database and cannot be modified on-line. With a sufficiently user-friendly interface, the equipment specialist himself can enter the data for control system use, thus shortening transmission delays and reducing transcription errors. The operational data are not initialized in Oracle, and need backup and archiving tools.

Module development: to simplify the writing of new Modules, and to guarantee that all Modules implement the basic functionalities, an inheritance mechanism has been included in the implementation around Oracle [2]. In short, each module belongs to a Class, which itself can derive from a Superclass, thus forming a tree. The root of the tree is the Control Module Class, including all basic functionalities. Below, one can find a sub-class called PPM-Modules, which offer the features of cycle-dependent data (Figure 1).

Operational data backup is handled as follows: as soon as a console finishes working on a set of process variables, a full backup of operational data is done for all relevant local processors, onto a Front-end computer disk file (one per processor). This file can be used in

case of a cold-start of a processor. Structural changes (adding equipment or fields in records) are taken into account when restoring operational data.

Archiving: when a particular operation (e.g. antiproton transfer over a beam line) works well, we can store the set of all relevant equipment parameters in a named file. When a similar operation is required at a later date, we can load the archived data into the equipment. In the meantime, the cycle structure may have been altered and some pieces of equipment may have been added, removed or altered. To keep the archived data useful, the full context (structure of all cycles, working set contents) must be saved together with the data, and the operator must be warned of any changes when the archive is reloaded. In the future we hope to have ORACLE on-line in a secure way so that we can use this database instead of files.

Oracle implementation [3]: a class is defined with an SQL*form called GMNEW and permits the following definitions: general data such as module name, author, superclass, etc...; class variable names and types; instance variable names and types; property names and the Procos called; the data record definition for Loop-Procos. Once the class is defined, a second form, called GMFILL, permits values for the read-only instance variables to be entered (Figure 2). When all data are filled in, the program MODULGEN generates the data structure for the module in the form of C source code, which is then merged with the frame and procos. A second program, MODULDOC, provides the documentation.

5. Experience with the new data organisation

After initial correction of a few bugs in the kernel code and the associated services, no data corruption problem has been reported for any

module implemented in this system.

A survey of the Loop-Procos written show that they have a mean size of 34 lines of C code, including record description and comments. This is conducive to simpler software maintenance.

A number of on-line facilities have been provided in a generalised form valid for any type of module, existing or future, such as:

- Data browsing by record, both for current or backup data, with possible check for differences.
- Tracing the connection between any physical variable and the hardware devices actually controlling it.
- For fast changes in Equipment Data (or when Oracle is unavailable), a tool provides on-line updating, submitting in parallel an update request for Oracle (manually processed via file logging, for the moment).

6. Possible developments

The advent of SQL-NET allows efficient remote access to an ORACLE database. The extension to yet another development environment (IBM for DBMS) can be avoided by migrating data-entry forms and table generators to the control-system workstations.

References

[1] A.Daneels et al., Standard Software modules for Equipment and Composite Variable Control, CERN PS/CO Note 84-01.

[2] L.Casalegno et al.,Building software modules for driving hardware

controlling physical variables: an Object-oriented approach, EPS Conf. on control systems for experimental physics, Villars-sur-Ollon, Switzerland Sept.28 - Oct.2, 1987.

[3] J.Cuperus, The Database for accelerator control in the Cern PS Complex, IEEE Particle Accelerator Conference, Washington D.C., March 16-19, 1987.

Figure Captions

Figure 1 Class hierarchies

Figure 2 Example of data entry form

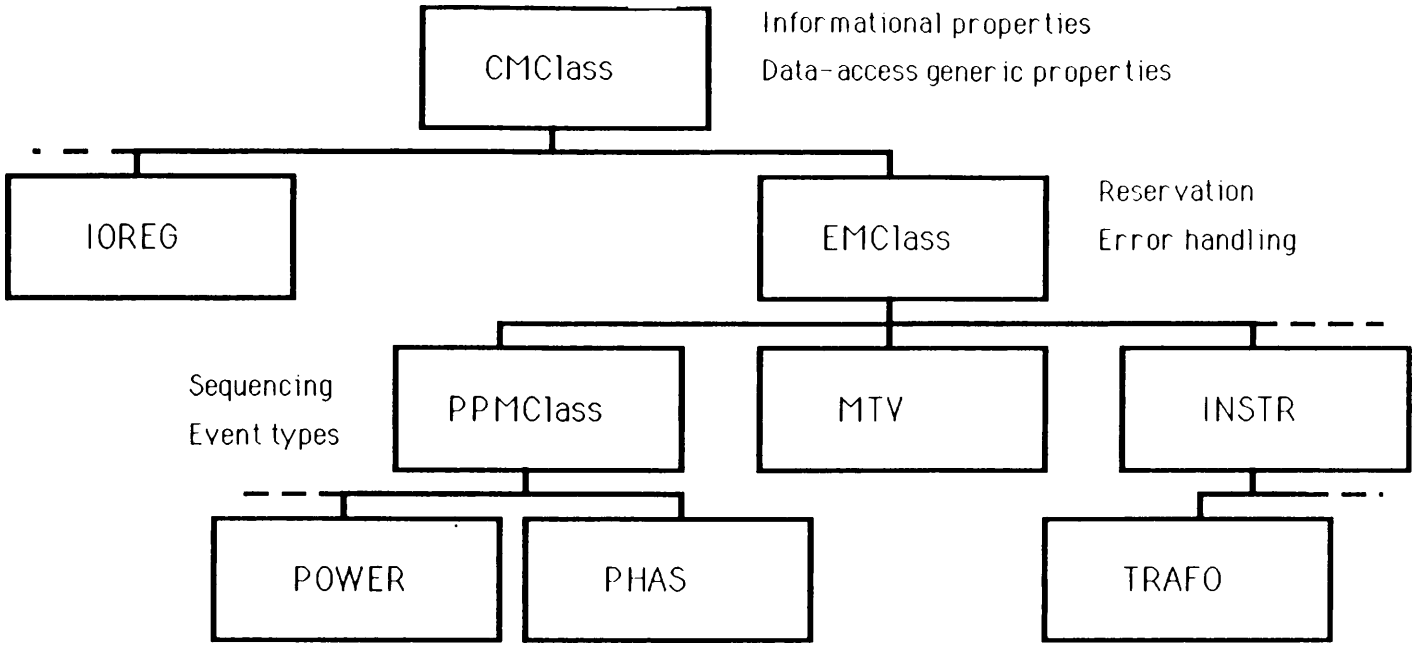
CMClass
Informational properties
Data-access generic properties
IOREG
EMClass
Reservation
Error handling
Sequencing
Event types
PPMClass
MTV
INSTR
POWER
PHAS
TRAFO

Fig. 4

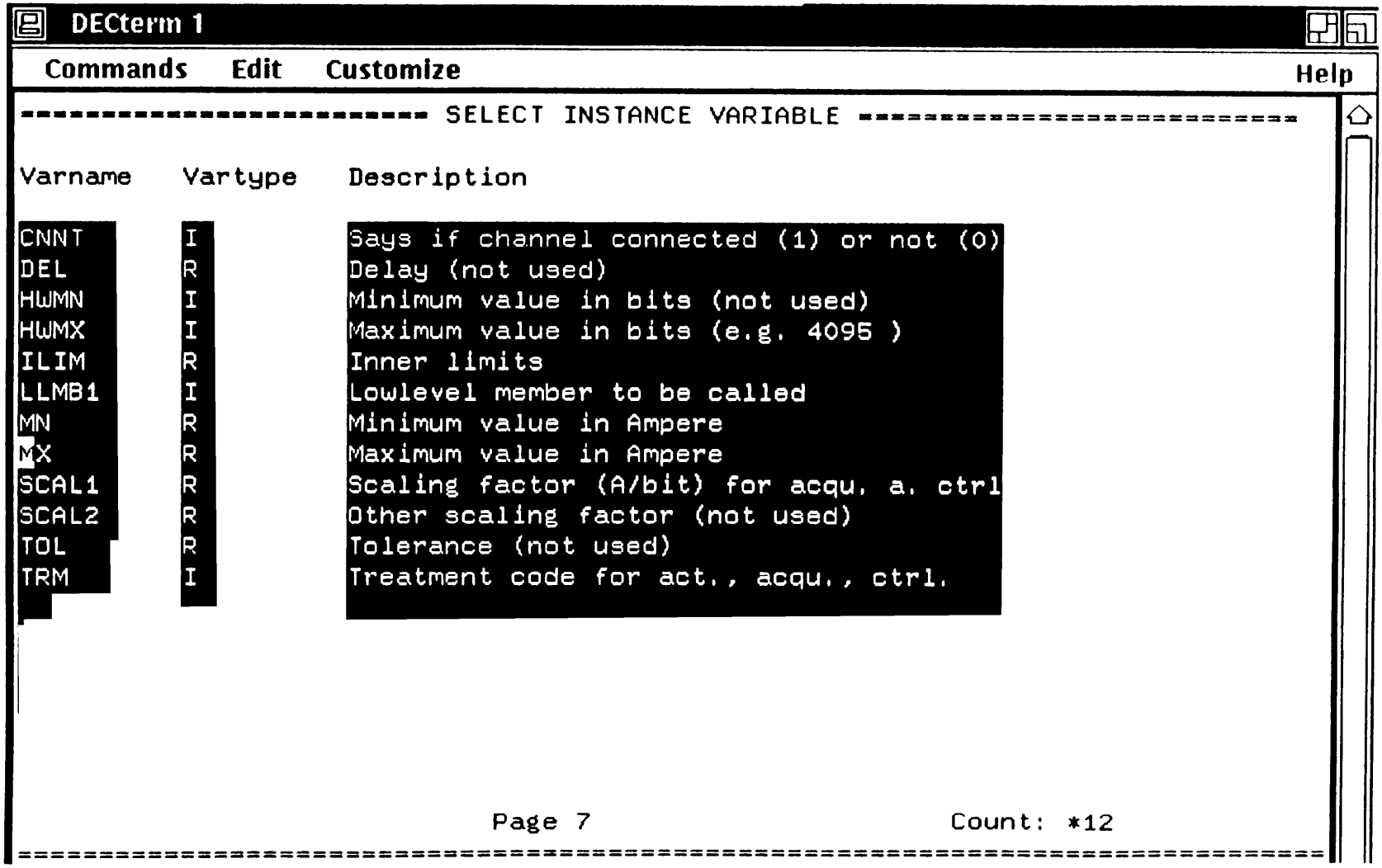
DECterm 1
Commands
Edit
Customize
Help
SELECT INSTANCE VARIABLE
Varname
Vartype
Description
CNNT I Says if channel connected (1) or not (0)
DEL R Delay (not used)
HWMN I Minimum value in bits (not used)
HWMX I Maximum value in bits (e.g. 4095)
ILIM R Inner limits
LLMB1 I Lowlevel member to be called
MN R Minimum value in Ampere
MX R Maximum value in Ampere
SCAL1 R Scaling factor (A/bit) for acqu. a. ctrl
SCAL2 R Other scaling factor (not used)
TOL R Tolerance (not used)
TRM I Treatment code for act., acqu., ctrl.
Page 7
Count: *12

Fig. 2