THE DATABASE FOR ACCELERATOR CONTROL IN THE CERN PS COMPLEX

J.H. Cupérus
CERN, 1211 Genève 23, Switzerland.

The use of a database started 7 years ago and is an effort to separate logic from data so that programs and routines can do a large number of operations on data structures without knowing a priori the contents of these structures. It is of great help in coping with the complexities of a system controlling many linked accelerators and storage rings.

Introduction

The CERN PS accelerator complex consists of 9 linked accelerators and storage rings. Together these machines accelerate protons, antiprotons, electrons, positrons and ions. The PS (Proton Synchrotron) can handle them all and redistributes them for local experiments or for further acceleration in the SPS (Super Proton Synchrotron) or LEP (the Large Electron Positron storage rings now under construction). Operation in each accelerator is organised in supercycles consisting of many cycles. A cycle can be as short as 1 s and is dedicated to one mode of operation (e.g. acceleration of positrons to a certain energy and ejection to the SPS) and the next cycle can be completely different. This means the switching of several thousands of parameters on a cycle to cycle basis. The operators can monitor and adjust each type of cycle independent of all others and can consider the accelerators as different virtual machines for each of them [1].

The accelerators were constructed and modernised over a period of more than 30 years and this results in a large diversity of equipment. Although the PS complex is not very large compared to some newer accelerators, it is nevertheless a very complex set of machines. The PS control system tries to make this complexity manageable and to give a uniform view of the equipment to the operators. For this we use several tools and one of them is the database. I will concentrate in this paper mainly on the real-time (RT) use of the data in this database and the emphasis will be on general concepts, which can be applied to many RT systems, rather than on the specific requirements of our application.

The PS Control System

The present control system [2] was developed 10 years ago to replace gradually many smaller systems which had grown independently. It consists of a network of 27 NORD computers for RT operation which we can divide in 3 categories: (a) Front-End Computers (FECs), one per accelerator, connected to the machine components and more than 100 microcomputers (Autonomous Crate Controllers or ACCs) through serial CAMAC; (b) console computers, each serving an identical non-dedicated general-purpose console and (c) service computers doing specific tasks such as message handling, sequencing, program development, and the TREES computer which provides RT database services.

The software is built in layers which, at each level, hide some of the complexities and diversity of the level below. On the lowest levels are the Interface Modules (IM) which drive specific CAMAC modules (which in turn control the equipment), the Equipment Modules (EM) which drive the various equipment elements through the IMs and the Composite Variable Modules (CVM) which can call a weighted combination of EM elements. The CVMs, EMs and IMs are instances of a general piece of data-driven software, the General Module (GM), which can recursively call itself [3].

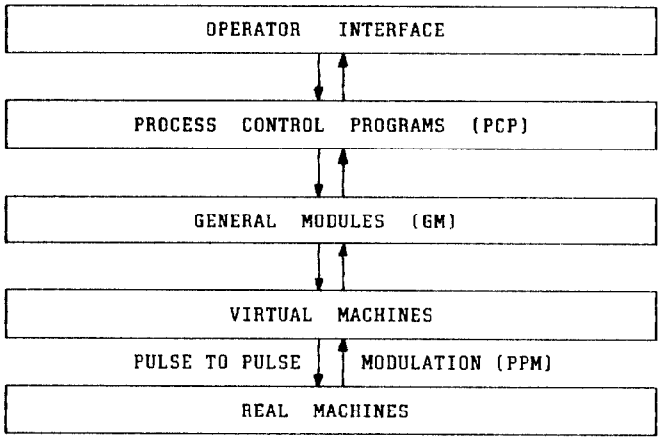

Fig. 1: schematic view of the control system

Fig. 1 gives a very simplified view of our control system. Most of the general services are omitted and emphasis is on two classes of software components: Process Control Programs (PCPs) and GMs. These classes consume the bulk of our programmer resources and productivity and reliability can be very much enhanced by making them more general (and as a consequence reusable) while giving them their specificity by making them operate on different data. Generalisation of the methods described here to other software is straightforward because these two classes stand for two general classes found in every control system: programs which initiate action and procedures which perform functions on request of the programs. In our system these software modules are distributed on more than a hundred computers and microcomputers and the subroutines are called with a remote procedure call mechanism.

The Database Management System

The control system project did not initially include the concept of a central database but each subsystem had its own data structures and data editors. In certain cases this may lead to a user interface which is exactly adapted to the requirements but, in practice, it means a lot of duplicate effort and, due to manpower limitations, often severely limited products. It also means that data entered for one application cannot be used for another and it is difficult to get an overall view of the data or documentation of the data on paper.

The need for more general data structures was first felt for the alarm system [4] which must survey the equipment and alert the operators in case of malfunctions. At that time (1980), only a hierarchical database (SIBAS) was available on the NORD computers. While such a system is well suited for large commercial applications where the data structure is stable over long periods, it was deemed to be badly adapted to our requirements of a large number of ever changing and growing data structures. We decided to build our own file system, INFO, with many features and data types

CH2387-9/87/0000-0704 $1.00 © IEEE

adapted specially to our needs. The data structures were relational but the system was not. The data base served mainly to generate RT data structures and this was done with application programs which could open many tables at the same time, each with an index. In that way we got relational capabilities but at the price of more complex application programs which had to do much of the work. INFO served well our needs and its use was gradually extended to other parts of our control system with the consequence that its limited capabilities got overstressed.

Recently we got network access to the relational database management system ORACLE on a central IBM 3090 computer and we are now loading all our data on this system. This will facilitate updating and manipulation of the data. It will make application programs simpler and will allow much more powerful database queries. It will not much change our RT data structures, however, which are derived from this database in much the same way as before.

Categories of Data

We can distinguish three categories of data in a RT system: static data, dynamic data and archive data.

Static data describe the structure of the system: parameters of the hardware (names, minimum and maximum values, conversion factors ...), configuration of the interface (CAMAC addresses, interconnections ...), relations between software modules and where they are located and so on. This structure is fairly stable and, when it is designed or modified, this is in general done by hardware specialists or control system specialists and not by the machine operators. The source of these data, directly or indirectly, is (or should be) the central database.

Dynamic data are the result of a setting by an operator (magnet current, switch position, description of the cycles in the supercycle ...) or the result of a measurement (beam profile, equipment status ...). These values are conserved, in computer memory or in hardware registers, until they are overwritten by new data. Backup of the current control settings is done from time to time on the disks of the various FECs. These data can be used to restore the FEC and associated ACCs to normal operation in case of corruption of the current data.

Archive data are permanent storage of past dynamic data. On request of the operator, the state of all control settings of a part of the machine is stored together with the identification of the parameter set, the kind of virtual machine and the date. Many different sets of archives can be stored. They are used later, in various combinations, to set up the machine quickly for a particular operation. Special kinds of archives are statistic data which store a compressed history of the beam measurement data and logs which give the history of certain actions and occurences.

Real-Time Data Structures

We use three classes of data structures which are derived from our central data base:

Dictionary Tables:

These are closely related to views existing on the central database. They are installed on the TREES computer and can be read by a remote procedure call to a routine which returns a complete record when the user produces the key. To make this service efficient, the records are ordered physically according to the key and the (short) index to the disk pages is permanently in memory. This has the advantage that any record can be retrieved with, at most, one disk access and often none if the disk page is already in memory, which often happens when closely related records are successively requested.

A heavy user of these tables is the alarm display program which gets a list of hardware element identifiers and alarm message numbers and then displays, with the help of the dictionary tables, a list of element names and alarm messages. The operator can then point to one of the elements and ask for details. The program will query the tables for the addresses of the relevant registers and their significance, make calls to the hardware and display the status of the registers, with the meaning of each bit described if so requested.

Another use of the dictionary tables is for setting up the hardware from a power-off state. An initialisation program looks up and executes the sequence of commands necessary to set up a class of equipment, followed by more specific commands for each individual hardware element. When all this is finished, the operational data from an appropriate archive are loaded. The list of hardware elements to operate on is given by the working set (see below).

It would be simpler to get the dictionary information directly from the central database but, for the moment at least, it is not possible for us to make a remote procedure call to ORACLE. Even if this were possible, there is a timing problem: ORACLE is quite efficient for responding to complicated queries for data sets but needs a minimum time even for returning a single record and a typical screen display may need a large number of them which can result in too long delays, especially in a multi-user database system where everybody has the same priority. The central system may also be down for maintenance at times unrelated to the operational needs of our control system.

Lists

A list is here a sequence of data of variable length. A list is identified by its type and element number. All lists are put together, one after the other, in one long file with an index pointing to the start of each list. A remote procedure call to the list handler will return the requested list. A list type can have any structure which is deemed useful. List types currently in use or planned for the near future are:

Working-Sets: an operator who wants to work with a set of equipment elements will go through a tree by means of a touch panel and will successively select the accelerator, the system, the sub-system and the virtual machine on which to work. At this level, the tree returns a working-set number. The operator may then call a general program, e.g. to initialise the equipment or to read or modify the status of the equipment. This program will acquire the working-set list and operate on the equipment elements in this list.

Scanlists: In most FECs and ACCs, there is an alarm scan program which surveys the hardware elements for malfunctions. This program is identical everywhere and, when it is started, it acquires its particular scanlist from the TREES computer. This is not only a list of equipment to operate on but it also tells the program how to do the checking for the element at hand. The scanlist is in fact a piece of simple pseudo code which is interpreted by the scan program.

Tables: Several programs or subroutines have data

in tabular form which is hardcoded or initialised by reading a corresponding data file. We should modify these modules so that they read the data as a list derived from the central database.

Indexes: Each archive (with control settings) is stored at present in its own file and there are difficulties to match old archive data to changed new configurations of equipment. It is planned to store them in the future in a single large file with an index pointing at the different archives. This index would be maintained with the help of the central database and would be available as a list. The structure of each archive would be kept in a dictionary file.

Compiled Data

In each FEC and ACC, sits an identical GM kernel which gets its specificity from data tables and a subroutine library. Most of the subroutines are fairly general and can be reused but some are linked to equipment that is so specific that they have to be written specially. The data tables have a complicated structure and variable dimensions. Acquiring them at initialisation would not be practical. Instead they are derived from the central database by an application program which produces source code (in Pascal or C) which is then compiled for the FEC or the ACC and linked to the GM kernel code and the subroutine library

The tedious job of rewriting a variant of an existing module is now reduced to filling in the data in the central database and writing a few well defined subroutines. Filling in the database tables is at present a bit abstract but this will improve when a shell has been built around these tables for guiding the module writer. A mechanism for inheritance will also be included so that the module writer can start from a previous module definition and fills in only the deviations from this definition as is done in object oriented languages.

After the database is updated, the writer can call a suite of programs will generate a complete GM code image ready for installation on a FEC or an ACC. A new version of the FEC dispatcher is also generated. This dispatcher distributes incoming remote calls for the GM to the relevant ACCs. Complete documentation is generated automatically on request.

Generation of the Real-Time Database Extension

New GM images are generated and loaded whenever necessary for an individual FEC segment or an ACC. New versions of the dictionary and list files are generated as a whole to guarantee consistency of the data. They are copied in a duplicate set of files on the TREES computer. Control is then transfered from the active to the duplicate files which now become the active files and vice versa. If something is wrong, we can do the inverse operation and come back to the old files. The alarm scan programs can sense that a new database version is installed and they will then automatically acquire a new version of their scanlist.

The delay for generating and installing new versions is about 30 minutes. This is usually no problem because most of the static data do not change much and about 2 updates per week are sufficient. For some special operations, however, or in case mistakes are detected, some faster action may be desired. In that case we will allow quick changes with a special editor which can override the normal read-only protection. This editor will leave an entry in a log file so that the central database can be updated accordingly later on.

Conclusions

Entering data in a database is a lot of effort. You get the most from your effort when this is done at the earliest possible moment. The data can then be used for planning, installation, running of the accelerator, general information, automatic documentation, help in troubleshooting and, in the near future, as an information source for expert systems. Some of our data were introduced into the database at a late stage and we did not profit fully from our efforts but even then it was worthwile. Any new design is now first looked at from a database viewpoint.

To conclude, some thoughts about possible future extensions. The 'compiled data' idea looks like a step in the right direction and we should try to generalise it and to apply it to other modules which may look more and more as objects communicating to each other with messages even if the real communication is still done by means of procedure calls. On the design side we should develop tools for manufacturing these objects by combining the best ideas from object oriented design with the use of a central database. This database can hold information about all modules in the system and how they interact with each other. It can be used to create virtual objects which can be checked for consistency with all other relevant objects. The data for the thousands of instantiations of these objects can be entered most conveniently in a tabular way directly into the database. When the designer is satisfied with his work, the virtual modules can then be compiled and installed in the RT system. Tools should be developed to make this installation easy and safe and with minimum disturbance to the existing system.

All this would greatly facilitate the building of sophisticated control systems but at the cost of complicated development and installation tools. This is not new however: programming in a higher level language is much easier than in machine language but at the cost of a complicated tool: the compiler. Nobody is frightened by this because you can buy reliable compilers cheaply while we have to build the tools for our control system ourselves which is difficult within the resources of a busy control team. Once built, however, such tools could be easily adapted to different computers and used for many different control systems.

Acknowledgements

I thank L. Casalegno, A. Daneels, P. Heymans, C.H. Sicard and P. Skarek for many good ideas on how to apply database access methods to our control system.

References

[1] P.P.Heymans, B.Kuiper, Concurrent Control of interacting Accelerators with Particle Beams of varying format and kind. Presented at EPS Europhysics Conference on Computers in Accelerator Design and Operation, Berlin 1983.

[2] G.P.Benincasa, J.Cupérus,A.Daneels,P.P.Heymans,J.P. Potier, Ch.Serre,P.Skarek, Design Goals and Application Software Structure for the CERN 28 GeV Accelerator Complex, presented at 2-nd IFAC/IFIP Symposium on Software for Computer Control, Prague, 1979.

[3] L.Casalegno, J.Cupérus, A.Daneels, Ch.Serre, Cl.H. Sicard, P.Skarek, Distributed Application Software Architecture applied to the LEP Preinjector Controls, presented at the 7th IFAC Workshop on Distributed Computer Control Systems, Julich, 1986.

[4] J. Cupérus, An Interactive Alarm System for the CERN PS Accelerator Complex, proc 1983 Part. Accel. Conf., Santa Fe.