EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN/PS 87-81 (CO)
6.10.1987

Project : NAPS
Domain : GENERAL MODULES
Category: CONF.PAPER
Status : FINAL

BUILDING SOFTWARE MODULES FOR DRIVING HARDWARE CONTROLLING PHYSICAL VARIABLES: AN OBJECT-ORIENTED APPROACH

L. Casalegno, J. Cupérus, A. Daneels, C.H. Sicard, P. Skarek

Abstract

Software modules to control hardware devices and physical beam variables have been written for several years in the CERN PS complex. They hide their intricacies and thus provide a user-friendly interface to the higher level application programmes. The concept leading to the design of these modules has strongly evolved whereas the interface towards the application programmes has been kept strictly the same for the sake of compatibility. All the basic software has been rewritten in C to improve portability and to be able to use the same techniques for driving hardware modules constrained by strong real-time requirements. Object-oriented concepts have been introduced in order to reduce the production time and to widen the applicability of the software to a larger domain. The paper shows how the concepts of dynamic creation of objects can be applied also in a domain such as accelerator control where the fixed layout of the hardware requires necessarily static images running on the target computers.

Paper presented at
the Europhysics Conference on Control Systems for Experimental Physics,
Villars-sur-Ollon, Switzerland, 28.9. - 2.10.-1987

Geneva, Switzerland

BUILDING SOFTWARE MODULES FOR DRIVING HARDWARE CONTROLLING PHYSICAL VARIABLES: AN OBJECT ORIENTED APPROACH

L. Casalegno, J. Cuperus, A. Daneels, C.H. Sicard, P. Skarek
European Organisation for Nuclear Research
1211 Geneva 23

Software modules to control hardware devices and physical beam variables have been written for several years in the CERN PS complex . They hide their intricacies and thus provide a user friendly interface to the higher level application programmes. The concept leading the design of these modules have strongly evolved whereas the interface towards the application programmes has been kept strictly the same for the sake of compatibility. All the basic software has been rewritten in C to improve portability and to be able to use the same techniques for driving hardware modules constrained by strong real time requirements. Object oriented concepts have been introduced in order to reduce the production time and to widen the applicability of the software to a larger domain. The paper shows how the concepts of dynamic creation of objects can be applied also in a domain such accelerator control where the fixed layout of the hardware requires necessarily static images running on the target computers.

Introduction

The CERN PS accelerator complex is constituted by several proton accelerators, two antiproton accumulators, two linacs, various transfer lines and the LEP preinjector. All these machines are controlled from a central control room through a network of around 20 minicomputers and 100 microprocessors interfaced to the process hardware through CAMAC. This network includes dedicated Front End process Computers (FECs) which control subsystems of the accelerators. The microcomputers are located in the CAMAC crates as Auxiliary Crate Controllers (ACCs or SMACCs for the newer projects).

SMACCs are double board M68000 based computers running the real time operating system RMS68k. Each SMACC is located in a CAMAC crate and controls the hardware modules contained in the same crate. The requests coming from the application programmes running in the FECs or from consoles are accepted by a routine resident in the FECs called DISPATCHER and redistributed to the relevant SMACCs. The software driving the equipments installed on the accelerators has been moved from the FECs to the SMACCs in order to delegate more and more the actual control to local processors. The DISPATCHER refers to routing dictionaries to send the commands to the relevant SMACC.

The software procedures written to translate and send the commands coming from the application programmes into the hardware devices will be called here Control Modules. Control Modules have been conceived to hide the intricacies of the underlaying structures, both hardware and software and to create a homogeneous interface to the higher level programmes.

Control Modules Structure

All Control Modules located in the SMACC are implemented with a unique frame which drives the specific code with the help of the data tables. The layout of a generic Control Module is shown in figure 1.

Data and code are separated in a Control Module: data are stored in data tables which are seen by the Control Module frame only which accesses them by means of data tables access routines.

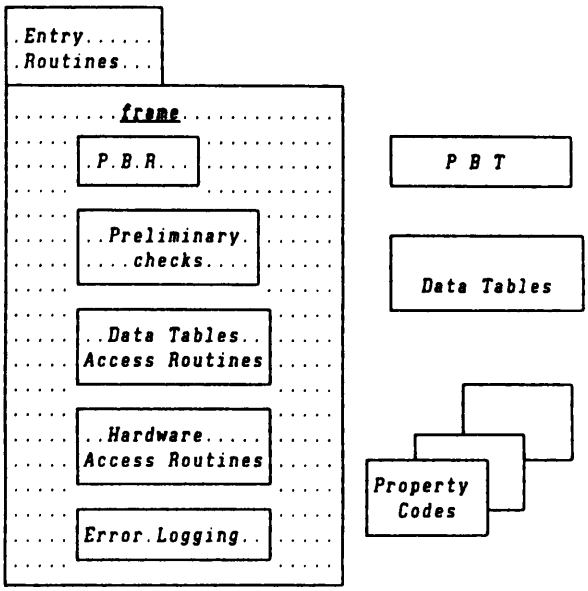

figure 1
Layout of a Control Module

The application programmes communicate with the modules by means of messages, implemented in our system with remote procedure calls. Part of the message is the property (the selector in object-oriented language) which indicates the required action.

In the Module the choice of the routine performing the required action is completely data driven: the Property Branch Routine (PBR) scans the Property Branch Table (PBT) which connects the property identifier with the address of the routine to be executed (Property Code).

The frame provides also the mechanism of parameters checking and sending elementary commands to the hardware (hardware access routines).

The flow of data in a Control Module is depicted in figure 2. First the Property Branch Table is scanned to find the address of the Property Code to be executed and the property description; then parameters are checked according to the property description; data reading from the data tables is performed before entering the Property Code. After execution of the specific code data are rewritten in the data tables and errors possibly generated are logged in an error table. In certain cases, the action can be so simple (e.g. direct hardware access) that it is executed directly by the frame with the help of the data tables, without accessing any specific code.

The first task of the Module writer is to fill in the tables which describe the Module and the objects in the Module. The programmer is helped in this by a user friendly data entry system, built with the powerful tools provided by the ORACLE data

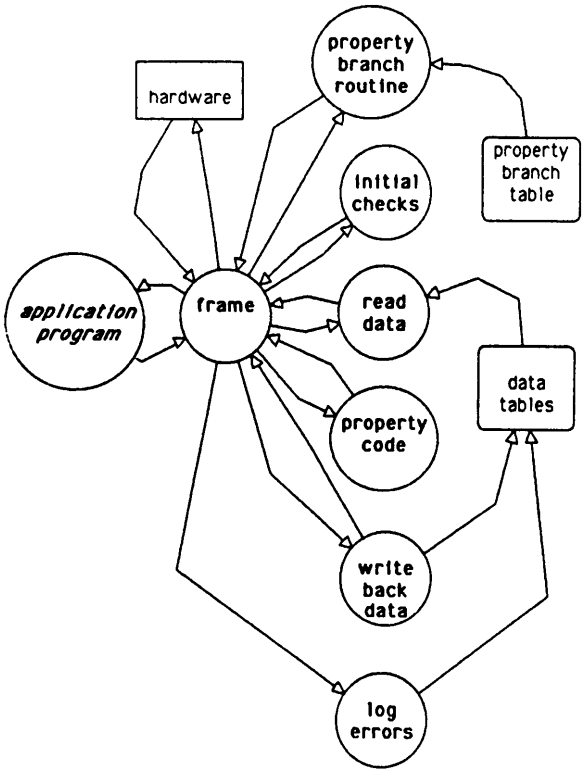

figure 2
Data flow in a Control Module

base management system. From the data base tables, the data are extracted by programmes which generate the data tables as source code of the C language and which produce printed documentation about the Module. The process is illustrated in figure 3.

The second task of the Module writer is to produce the various property codes which will execute the specialised actions required. Many of these codes are fairly standard and can be taken from other Modules with little or no adaptation. The property codes are included in a library of subroutines which will be linked to the frame and data table code.

The development of Control Modules is not realised directly on the target machine (SMACC); the DBMS seats on an IBM running VM/CMS whereas the routine libraries, the compilers and the linker are installed on a VAX running ULTRIX. The process of Module creation is illustrated in figure 3.

Control Modules and Classes

One Control Module usually serves many identical members differing only by the data contained in the data tables. One can think of a Control Module as a class of objects, e.g. a power supply driver, and an entry in the data tables as an instance of that class, e.g. a power supply.

The data base records all the classes present in the system and all the instances belonging to those classes.

The DBMS allows to create and delete Control Module instances : it takes care of the disk space not used any more, acting as a mass storage garbage collector.

Classes can have similarities; the data base interface programme allows to derive a class from an already existing one inheriting the list of the ancestor's class properties and the definition of the structure of the data tables.

A property can be of type class or instance: the response of a class property does not depend on the object being accessed but only on the Control Module (class), whereas the response of an instance property usually changes from object to object. Logically speaking a class property belongs to or is inherited by the class as a whole; an instance property belongs to or is inherited by an object.

It is possible to redefine the property code corresponding to a given function in a child class by overloading the property description of the ancestor's class. The Control Module sees its property description only and not the ancestor's one; access to the ancestor's code is possible by means of an explicit call inside the child's property code.

A programme extracts information from the data base and generates which are included in the module

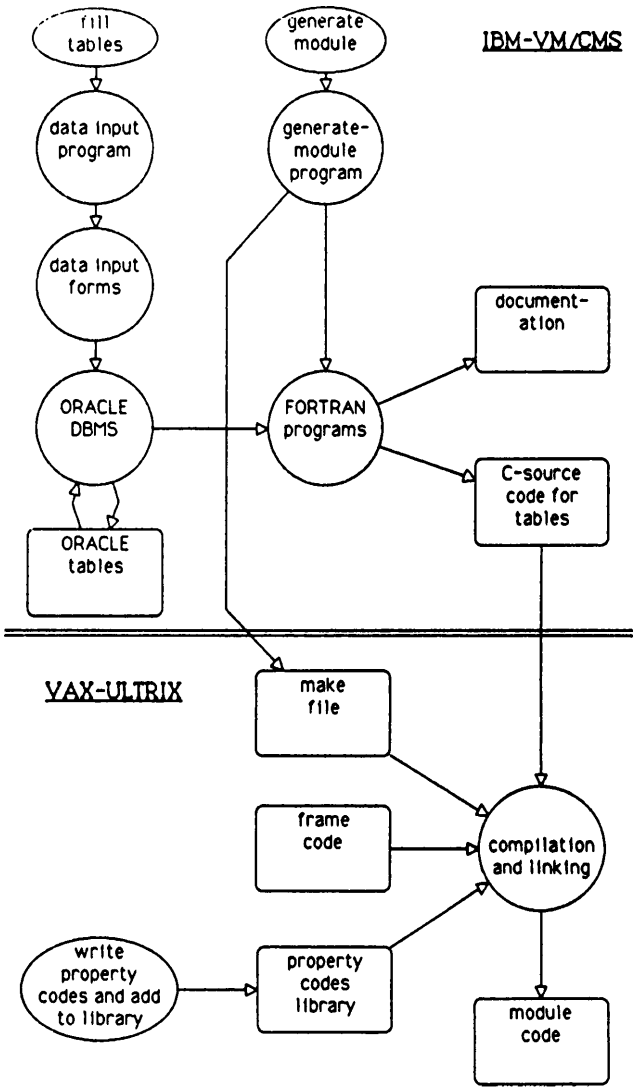

figure 3
Creating a new Control Module version

VAX and ULTRIX are trademarks of DEC

image to be loaded on the target computer. The objects are now frozen in the running code: they cannot be changed neither in size nor in number; only the data declared in the data tables as being read/write can be modified.

Thus some of the characteristics usually found in an object oriented language have been moved to the data base recording the classes present in the system. Whilst no dynamic creation or deletion of objects is allowed nevertheless the same operation can be accomplished by a two step mechanism (load information in the data base and then rebuild the target image) that in a widely distributed and heterogeneous system as the PS complex, guarantees safety, maintainability and coherence between documentation and actual layout.

On the other hand the selection mechanism realised by means of the Property Branch Routine and the Property Branch Table allows on line dynamic binding: the Control Module will accept different data types as arguments depending on the required property; the property code can be written to treat just one object at a time or an array of objects.

Finally, with the help of the frame, data are hidden from the user programmes and are accessed only by means of the properties that are an abstraction of the data themselves.

The development environment

The SMACCs are strongly process oriented and do not have a native development environment. As development environment a VAX running UNIX has been chosen, on which Cross Compilers were installed to produce object code for the M68000 . C was chosen as development language for the Control Module frame because it guarantees portability that is considered paramount in such a situation.

The code was debugged and tested with the native compiler with the help of the powerful symbolic debugger dbx[1]. Then it was optimised with the help of the profil[1] facility.: the execution time has been reduced by a factor of more than 2 .

By means of these optimisation techniques it has been possible to supply a high level language code time effective enough to be used also in critical real time tasks such as performing the modulation of the beam characteristics from one pulse to another . The code written in assembler was reduced to less than 100 statements.

Specific test programmes and techniques have been implemented to validate the code being written and to perform repetitive tests in case of modifications and enhancements. A set of test cases has been written and kept in a file; the file can be edited by means of a dedicated editor to append, insert or delete tests. A programme scans the file and performs the tests in the given order. A test report is produced and kept as a documentation together with the test cases list. These simple techniques have been found very powerful: they save production and maintenance time and enhance reliability .

Conclusions

Object Oriented programming principles have been applied to build Control Modules : data hiding, late binding, data abstraction at run time; inheritance, method overloading, instance creation and initialisation, garbage collection with the help of a data base keeping the description of the Modules. The system described in this paper has demonstrated that the same methodology are applicable to very time sensitive tasks with the help of code optimisation techniques that enable a strong reduction in the code bulk and in the execution time.

References

[1] L. Casalegno et al., "Distributed application software architecture applied to the LEP preinjector controls", presented at the 7th IFAC Workshop on Distributed Computer Control Systems, Mayschoss/Bad Neuenahr, Fed. Rep. of Germany, September 30 - October 2, 1986.

[2] A. Daneels, P. Skarek, A General Software Module for CAMAC, Equipment and Composite Variable Control, in Accelerator Control Systems, North Holland, 1986, pp. 141-145.

[3] B.J. Cox, Object Oriented Programming: An Evolutionary Approach, Reading: Addison-Wesley Publishing Company, 1986.

[4] K.J. Schmucker, Object-Oriented Programming for the Macintosh, Hasbrouck Heights: Hayden Book Company, 1986.

[5] B. Stroustrup, The C++ Programming Language, Reading: Addison-Wesley Publishing Company, 1986.

[6] Issue on Object Oriented Programming, Byte, vol 8, August 1986.

[7] Unix User's Manual