DD
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH CERN/ECP 93-22
30 November 1993

CM-P00060337

**BEAM-LINE OPERATION USING AN INDUSTRIAL CONTROL SYSTEM
AND DISTRIBUTED OBJECT-ORIENTED HARDWARE ACCESS**

H. Butler, D.R. Myers, W. Von Rüden and J. Yang

CERN, Geneva, Switzerland

Abstract

We describe the Control System of the CERN Neutrino Beam line, based on FactoryLink and equipment controllers using PCs running LynxOS. FactoryLink runs on a workstation and connects via communications tasks over Ethernet to both the PCs and the data acquisition systems of the experiments. The PCs access VME and CAMAC directly via the VICbus. Object-oriented control software, entirely data driven, deals with hardware modules and local survey operations. Remote stations have access via XWindow.

Beamline Operation using an Industrial Control System and Distributed Object-Oriented Hardware Access

H. Butler, D.R. Myers, W. von Rüden and J. Yang
CERN, ECP Division, Geneva, Switzerland

Abstract

We describe the Control System of the CERN Neutrino Beamline, based on FactoryLink and equipment controllers using PCs running LynxOS. FactoryLink runs on a workstation and connects via communications tasks over Ethernet to both the PCs and the data acquisition systems of the experiments. The PCs access VME and CAMAC directly via the VICbus. Object-oriented control software, entirely data driven, deals with hardware modules and local survey operations. Remote stations have access via X-Window.

INTRODUCTION

CERN's West Area Neutrino Facility (WANF) is being rejuvenated to prepare for three years of data taking by the CHORUS and NOMAD neutrino experiments. Although the basic beamline components and their hardware interfaces will have the same characteristics as before, the old control system based on Nord100 computers and serial CAMAC is being replaced.

The neutrino beam line extends over more than 1 km and has three major sections. The first part is the proton beam from the ejection point of the SPS until the target. Although this part is managed by the accelerator control system, the beam data need to be read for each ejection by the WANF control system. The second part is the Neutrino Beam Control (NBC) with the secondary beam equipment such as the horn, reflector, decay tunnel, etc. The last part is the Neutrino Flux Measurement (NFM) consisting mainly of silicon detectors and calibration equipment to measure the flux of secondary particles in the shielding. A detailed description of all beam components and the neutrino beam flux monitoring can be found elsewhere[1].

REQUIREMENTS

The control system has to deal with three major functions: initialisation and survey of the beamline equipment, data acquisition and transmission to the experiments, and calibration of the neutrino flux. There will be two ejections with an interval of 2.4 seconds per SPS cycle of 14.4 seconds. The amount of data to be collected and sent to the users is not more than about 1 kbyte per ejection.

The system has to be accessible simultaneously from several remote workstations and provide protection against accidental usage. It should offer a graphical user interface with animation, alarm handling, data logging and historical trending. These requirements suggested an industrial control system to avoid heavy manpower investment in software development. CAMAC had to be kept as the interface to the equipment, since most of the modules including special

purpose interfaces existed already. All software should be data driven to allow for easy reconfiguration and extensions of the system.

ARCHITECTURE

Figure 1 shows the architecture of the control system. A central supervisor contains all information required to initialise, operate and monitor the system. It supervises the equipment controllers of the sub-systems and links to the users both for data transfers and for supervisory functions. Sub-systems and/or users can be added or removed independently.

Sub-systems have independent controllers and look after all local actions and survey functions. They only receive commands and report changes to the supervisor. Since there is no critical equipment to be monitored, these systems do not need to be autonomous and restart automatically.

Fig. 1. Control System Architecture

Since the required data rates are very low and the distances between the systems rather large, standard networks services are sufficient, provided that one does not have to compete with public network traffic.

IMPLEMENTATION

We selected an industrial controls package, FactoryLink, which provides the required functionality and allows the interfacing to equipment controllers which deal with the application-specific equipment. The choice of the server, an HP730 running HP-UX, was made as a consequence of the choice of FactoryLink.

For the equipment controllers we followed the choices made by the accelerator controls group, i.e. rack-mountable PCs based on Intel 486s running LynxOS and having an

interface to CAMAC via the VICbus and the CES VCC2117 crate controller. Although we only require CAMAC, VME can be supported as well. LynxOS is a UNIX implementation with real-time extensions supporting POSIX threads, which are required for concurrent surveys and LAM handling. LynxOS supports all the usual software packages, including TCP/IP, NFS, RPC, X-Window and OSF/Motif. An ESONE CAMAC library provides both local and remote CAMAC access.

Ethernet and RPC over TCP/IP are used for all communications. Standard X-Window is used to access the supervisor remotely. Since both experiments use OS-9 and TCP/IP for their data acquisition systems, the same communication software can be used.

With these choices the system looks as shown in Fig.2.

Fig. 2. The WANF Beamline Control System

Industrial control principles are applied to all static equipment (power supplies, interlocks, parameter loading) with typical reaction times in the order of one second. Such operations are independent of the beam timing.

The second mode of operation, although not very demanding in speed, is related to data collection after each beam ejection. Information on the primary beam is combined with data of the secondary beam elements and sent to FactoryLink for logging and analysis.

The experiments require for each ejection a pre-defined fixed sized block of data. The equipment controllers prepare these blocks and send them to the data distribution task, which combines them and forwards them to experiments having subscribed to it.

THE SUPERVISOR

FactoryLink is constructed around a real-time database accessible by a set of tasks such as the graphical user interface, alarm handler, data logger and timer. Tasks are independent of each other and communicate only via the database. No user programming is required as the system can be configured via databases and pre-defined functions. Although a variety of industrial controllers are supported, two communication tasks,

named FLin and FLOut, had to be developed in our case using the Programmers Access Kit (PAK) to connect to HEP specific equipment; one to transmit transactions initiated by FactoryLink to the equipment controllers and one to send transactions initiated by the controllers back to the real-time database. Each basic element in FactoryLink is defined by a "tag", which can represent basic types such as integers, reals, messages, etc. We map tags to objects residing in the equipment controllers. Since FactoryLink has a flat structure, a tag naming convention has been devised containing the name of the equipment controller, the process and the object. This allows the routing of messages to an object in a data driven fashion.

EQUIPMENT CONTROL

As mentioned earlier the equipment controllers are based on industrial PCs interfaced to CAMAC via the VICbus. Figure 3 shows such a configuration.

Fig. 3. Equipment controller arrangement

It seemed to us desirable to reflect the modularity of the hardware in the application software. Therefore, one of the aims of the project is to evaluate how well the object-oriented programming model [2] supports this type of application. Class diagrams are used to visualise the hierarchical inheritance of classes and to document through them the design, at least partially.

The tool 'Diagram' available under NeXTstep was used for making designs of the class hierarchy, but no CASE tools were used as previous experience had shown that tools currently on the market were expensive and have problems of one sort or another. For example, we consider it essential that an OO CASE tool generates compilable code and manages the incorporation of hand-written algorithms which it inserts as appropriate. Without this feature the description of the application stored by the tool quickly ceases to correspond with the code and the tool becomes of no further use.

The GNU C++ [3] environment is used for all programming on the front-ends, although unfortunately the version 1.40 supported by LynxOS V2.1 is now very out-of-date and not bug free. A similar situation exists for the

debugger. Using C++ a set of classes has been written which describe CAMAC branches, crates and modules. The CAMAC module class has then been sub-classed in order to produce classes corresponding to particular hardware modules, as well as the items these contain, such as generic input registers, output registers, scalars, ADCs, timers, and so on. As CAMAC or VME modules do not usually contain single registers or bits, classes corresponding to generic sets of bits and registers were developed. Generic in this sense implies that, for example, the BitModule class provides a client interface independent of the hardware in which a particular instance is realised (see Fig. 4.).

Fig. 4. Partial Class Diagram for CAMAC Modules

Thus, a particular hardware module class, such as a KS3472, multiply inherits from CamacModule and BitModule. Through CamacModule it knows in which Branch, Crate and Station it is sitting, whilst the inheritance from the BitModule base class forces it to provide the member functions defining the standard interface for sets of bits, including readBit(number) and writeBit(number).

Instances of the classes modelling individual bits and registers contain references to the modules in which they sit and these references are defined in a Filemaker Pro database running on Macintosh. The database is used to ease the data entry by non-experts by using the user friendly Macintosh interface. From the database we generate a configuration file which is used to initialise the system as well as to produce cabling lists for the hardware technicians. Table 1 is an extract

of the text file seen by the tasks running in an equipment controller. This scheme makes the higher levels of the application independent not only of the particular channel or module which must be accessed in order to read or write a datum, but also the type of hardware bus used. Thus, a micro-switch, called doorOpen, could be moved from a CAMAC to a VME module and the only software change required would be to change the database entry for doorOpen to specify the new channel number and module name. The polymorphism inherent in C++ as an object-oriented language allows this to work for any named module so long as it inherits from BitModule.

In addition to a class corresponding to a simple ADC, a sub-class called LinearTransducer is used to model any physical device which produces a voltage proportional to some physical value, such as a thermometer, vacuum gauge, water level monitor, and so forth. A particular instance of LinearTransducer has a reference to the ADCModule in which it is situated, plus parameters specifying the slope and intercept of the generated function. By choosing appropriate values for these parameters such an object can provide its client(s) with values in sensible units, such as degrees or amperes, and so at the level of FactoryLink only physical quantities are visible.

Apart from classes used to model hardware elements, others are used to model more abstract features. As an example we can consider the Survey class. Some of the CAMAC modules may generate interrupts via LAMs but many others do not. To find out if their values have changed it is necessary to check them periodically, and for this purpose objects of the TimedSurvey class may be instantiated. A TimedSurvey object has a list of objects to survey and a sleep interval in seconds. When awoken it sends a message to each object on its list to ask them if they have changed and then goes back to sleep. As some values may change often, and others only occasionally, several TimedSurvey objects may be instantiated with different intervals. The "haveYouChanged" member functions compare the current values with the previous ones and send any changes to FactoryLink.

The front-end code all runs in a single process in order to avoid complications in a system with distributed objects. This process is configured as a daemon which can respond to remote procedure calls (RPCs) generated by the FactoryLink supervisor. However, the time taken for a TimedSurvey, or to run a data acquisition task is not well defined and in principle should not interfere with FactoryLink commands. To achieve this goal asynchronous tasks run in separate POSIX threads, and these are scheduled at different priorities.

Table 1: Extract of a WANF Front-End database file

Class	Name	Parameters				
CamacBranch	nbcBranchPwr	vcc_2				
CamacCrate	nbcCamacPwr	nbcBranchPwr				
Survey	surveyA	2				
KS3510	ADCModule-1	nbcCamacPwr	10	5		
KS3472	statusBits	nbcCamacPwr	12			
LinTransducer	hornVoltage	ADCModule-1	1	1.0	3.7	surveyA
PiraniGauge	driftVacuum	ADCModule-1	6	0.55	0.03	-0.5

Data acquisition runs at a higher level than the thread which handles RPC commands, and surveying runs at the bottom.

Threads, which are essentially light-weight processes all running in a single address space, were found to be very convenient for this purpose. Their use with C++ involved adding some function prototypes, which were missing in LynxOS, and the addition of an intermediate function call which receives the identifier of the object which should be run in the thread as a parameter passed by the call which starts the thread.

COMMUNICATIONS

As mentioned above, RPCs were used to send commands from FactoryLink to the daemon and, in fact also in the other direction. Two FactoryLink tasks, FLin and FOut had to be written for this purpose. When FactoryLink starts up it connects to the front-end machines and then tells them its machine name and the program number needed to communicate with FOut. Thus FactoryLink may be moved to another computer without any change to the front-end software. FactoryLink then instructs the daemon to read in a named configuration file, which creates all the necessary objects. At this point the daemon is ready for business and can accept commands to start and stop surveys or data acquisition and set command bits.

Data required by the experiments are transferred to another process in the BCS, called DistributeData. This transfer is also accomplished by RPC and the information necessary to make the call is provided in a similar way to that for communication with FOut. Experiments can subscribe to the data distribution system by sending requests to DistributeData, which forks a processes for each subscriber (threads are not available under HP-UX). After each beam pulse the NBC and NFM front-end machines send data blocks to DistributedData, which combines them and then pipes the result to each of its children to send on the experiments. The reason for this extra level of complexity is that if one of the remote computers dies this only affects the child process, and not DistributedData itself.

Data from the SPS equipment controllers is collected directly by our beam control sub-system using EQUIP calls [4] provided by the SL controls group. A special IP router had to be installed to avoid using CERN's public network to address the equipment on the accelerators private network.

CONCLUSION

We are satisfied with the functionality provided by FactoryLink. Our expectation to save manpower in the application development has been confirmed, even if the learning time was longer than anticipated.

The OO Model was found to deal very well with the task of programming hardware modules, registers, etc. However, for efficiency reasons, care must be taken when accessing multiple bits within the same register that only a single driver call is made.

We lost a lot of time in setting up the industrial PC with all its options and to work with CAMAC, although we were not supposed to develop anything (we used the recommended solution). The chief problem comes from the multitude of

hardware options, switch settings and incompatibilities between cards from different manufacturers. There is no support under LynxOS to set up cards, which come with their software on 5 inch MS-DOS floppies.

Despite such problems, we expect to have the system operational for the test run in October 1993.

ACKNOWLEDGEMENTS

The ECP-EDN group helped defining the requirements [5] and provided the hardware support for the equipment. R. Grabit developed the new electronics for the silicon detectors. We thank the Chorus collaboration for helping with the NFM calibration and D. Steele of Nomad for writing the data distribution task.

We would like to thank our colleagues from the SL controls group, who supported us strongly in our fight with the PCs, the support teams in CN division for help with LynxOS and for network and workstation support.

Last but not least, the AT-IC team convinced us to use FactoryLink and helped us to get going.

REFERENCES

- [1] Heijne, E.H.M., " Muon Flux Measurement with Silicon Detectors in the CERN neutrino beams", Yellow Report CERN 83-06.
- [2] As an introduction to OO programming see, for example, Bertrand Meyer, Object-Oriented Software Construction, Prentice Hall, 1988
- [3] Stroustrup B., The C++ Programming Language, second edition, Addison-Wesley, 1991.
- [4] CERN USAP Group, "Final Report on the uniform equipment access at CERN, CERN/PS 93-16 (CO), May 1993
- [5] Nathaniel, A. "Tentative specifications for the new NFM system", CERN internal note, February 1992
Ball, A.E., "Neutrino Beam Control", CERN internal note, March 1992
Maugain, J.M., "Technical Specifications of the WANF control system", Draft April 1992, private communication.