GROUND TEST ACCELERATOR CONTROL SYSTEM SOFTWARE*

L. Burczyk, R. Dalesio, R. Dingler, J. Hill, J. A. Howell,†
D. Kerstiens, R. King, A. Kozubal, C. Little, V. Martz, R. Rothrock, J. Sutton

Los Alamos National Laboratory, Los Alamos, NM 87545

Abstract

The GTA control system provides an environment in which the automation of a state-of-the-art accelerator can be developed. It makes use of commercially available computers, workstations, computer networks, industrial I/O equipment, and software. This system has built-in supervisory control (like most accelerator control systems), tools to support continuous control (like the process control industry), and sequential control for automatic start-up and fault recovery (like few other accelerator control systems).

Several software tools support these levels of control: a real-time operating system (VxWorks) with a real-time kernel (VRTX), a configuration database, a sequencer, and a graphics editor. VxWorks supports multitasking, fast context-switching, and preemptive scheduling. VxWorks/VRTX is a network-based development environment specifically designed to work in partnership with the UNIX operating system. A database provides the interface to the accelerator components. It consists of a run time library and a database configuration and editing tool. A sequencer initiates and controls the operation of all sequence programs (expressed as state programs). A graphics editor gives the user the ability to create color graphic displays showing the state of the machine in either text or graphics form.

Introduction

The control system for the Los Alamos Ground Test Accelerator (GTA) is required to provide complete accelerator automation. That is, it must provide automatic start-up, optimization, steady state running, and shut-down. A more complete description of the automation issues and the levels of control required is found in [1]. In support of these requirements, the control system is being designed with these goals in mind:

(1) Provide early support for testing and commissioning GTA subsystems. This is to allow the physicist use of controls in order to check out components and to develop an early understanding of accelerator subsystems.

(2) Provide support for automation experiments. Currently, the main emphasis is on the ion source and output optics automation.

(3) Provide a system that is flexible and extensible so that the potentially difficult problems of building a completely automated and optimized accelerator can be solved.

The hardware and software components of the control system make use of commercially available components: workstations, networks, industrial I/O equipment, and software.

System Hardware Overview

A distributed network (Fig. 1.) has been selected to provide the computational power required to automate the GTA in an evolving research environment.

The various processors used in the network are as follows: the *Input-Output Controller* (IOC), the *Operator Interface* (OPI), the *Integrated Control Processor* (ICP), and the *Artificial Intelligence Processor* (AIP). The functions provided by these processors and their associated software is described in the following sections.

*Work supported and funded by the Department of Defense, US Army Strategic Defense Command, under the auspices of the Department of Energy.

† Presenter

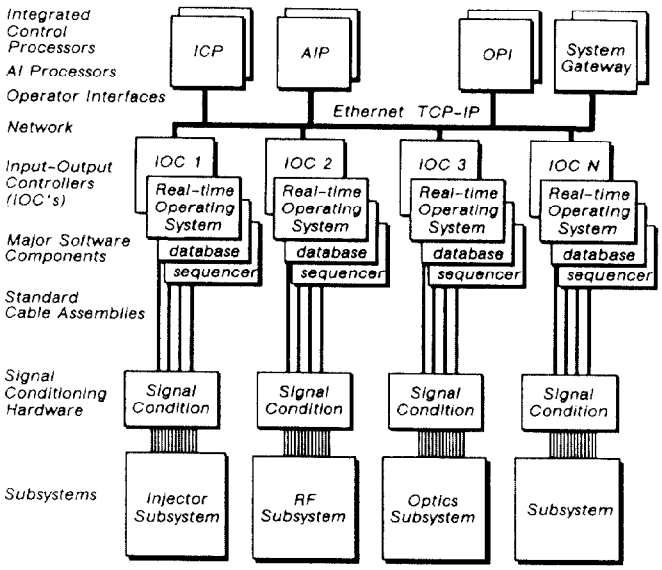

Fig. 1. GTA control system architecture.

System Software Overview

Network

Ethernet using the TCP/IP protocol has been chosen for the GTA control system network because it is commercially available and is well supported by established standards. The function of this network is to provide a generalized communication path between the computers, workstations, and processors that compose the control system. This includes the development system and the data analysis systems.

IOC Software

The IOC provides a direct interface to each accelerator subsystem. The standard IOC processor [2] is the Motorola 68020 32-bit processor running in a Versa-Module European (VME) bus. Software implemented on the 68020 will run on the successor (68030) at increased speeds. The IOC software architecture is shown in Fig. 2.

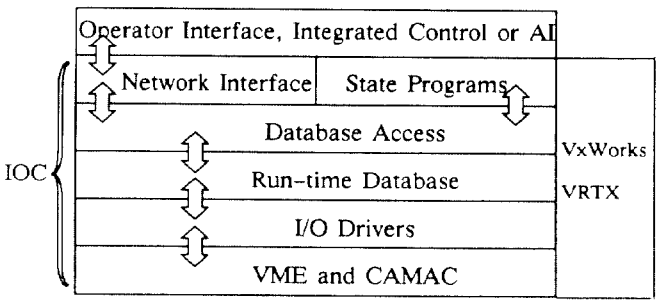

Fig. 2. Software Architecture.

VxWorks. The real-time operating system for the IOC is VxWorks [3]. It uses the VRTX real-time executive (from Ready Systems, Palo Alto, CA) as the kernel of the real-time operating system. The VxWorks (from Wind River, Inc., Emeryville, CA) development environment is currently networked with the SUN/Unix environment and will soon be networked with the VAX/VMS

environment. VxWorks and Unix work together to form a complete, hybrid environment. VxWorks is used for running, testing, and debugging real-time applications, while Unix is used as a high-level software development system. VxWorks provides extensive networking facilities, symbolic debugging, a Unix cross-development package, I/O drivers, a C shell, and memory management. The VRTX kernel provides the usual features found in real-time operating systems: preemptive scheduling, fast context switching, fast intertask communications and synchronization, and fast interrupt response.

Sequencer and State Programs. Application programs are typically implemented in the IOC in one of two ways: C language programs or finite state programs. Each state program is an independent entity within an IOC. State programs are expressed in a special state notation language and are compiled by a state notation compiler (SNC) [4]. Currently, the state programs are automatically compiled into C code. Although the state notation language provides building blocks for control, it is being enhanced to provide more sophisticated structure for effectively implementing complex control algorithms. Some of these enhancements are parameterization, hierarchical structure to control algorithms, and modularization of state programs.

All state programs are initiated and executed by a program known as the sequencer. The sequencer is capable of calling a C subroutine to provide local reduction of large groups of raw data. This is useful because it reduces the amount of data transfer required across the network. Like the other major software components of the IOC, the state notation compiler is written in a high-level language (C, YACC, and LEX) to be portable and therefore independent of a particular hardware implementation.

Sequence programs will be written for GTA to implement automatic start-up and fault recovery.

Database. The GTA database is distributed in nature, with each IOC database containing all the necessary data for a particular subsystem. The database contains the information describing the current state of the subsystem and the control parameters to maintain that state. This information is available throughout the control network via a database access library, and sections of it (fields) can be changed over the network.

A special task, the Database Configuration Task (DCT) builds both the database and the process variable directory, which resides on disk for modification and report generation. Each IOC has its database loaded into memory at load time. The DCT, described below, allows the database to be modified interactively.

The database is organized into records. Current record types are analog input, analog output, binary input, binary output, stepper motor, power supply, waveform, PID, and calculations. The smallest element in the database record is a field. Typical fields are process variable name, descriptor, conversion information, alarm limits, current value, desired output (a control parameter), archiving information, and display formatting.

The current state of the accelerator is changed by modifying the fields pertaining to alarm generation and the control parameters. These parameters are available to the sequencer task, the OPI, the AIP, and other control processors throughout the network via a library of access routines. For supervisory control, the operator can either change the outputs and monitor the effects or define the setpoint control parameters in the database and modify the setpoint to maintain a steady state. To achieve automation, the sequencer will need to assume the task of modifying the alarm and control parameters to change the state of the accelerator.

All database software is written in C.

Request/Response Servers. Two communications tasks provide external access to the IOC database. The *Request Server* accepts requests from the network on a predefined network port. The requests fall into three classes: (1) read from the database, (2) write to the database, or (3) monitor a value in the database. The read and write requests may cause the corresponding input/output channel to be accessed. The monitor request allows external nodes to process inputs as events occur, rather than by polling. The *Response Server* returns the value or values to the requesting task.

Device Drivers. The purpose of the GTA device driver modules is to provide higher level application tasks with a standard interface to the GTA system hardware.

The *analog input driver* will read the current value of a 12- or 16-bit analog input and store it at a location specified in the calling sequence. There are currently three types of analog input cards and a number of different uses for each type: VMIC-3100 (12 bit), VMIC-311616 (16 bit), and XYCOM-566 (12 bit gated).

The *analog output driver* will read/write values from/to 12- or 16-bit analog output cards. In both cases the driver will store the value read/written at the location specified in the call. There are currently two types of commercial analog output cards and an external bus I/O module that looks like an analog output to the GTA system software: VMIC-4100 (12 bit) and VMIC-4116 (16 bit).

The *binary input driver* will read the current value of a 32-channel binary input module and store that value at a location specified in the calling sequence. There are currently two types of binary input cards: Burr-Brown-910 and Xycom-210.

The *binary output driver* will read/write a specified value from/to a 32-channel binary output module. In either case, it will store the current value of the module ANDed with a specified mask at a location given in the calling sequence. There are currently two types of binary output cards: Burr-Brown-902 and Xycom-220.

The *GPIB driver* handles low-level interactions with the National Instruments GPIB-1014 interface. For the most part, this driver is supplied by National and is similar to a standard UNIX device driver. The device is opened, closed, read, written, and supports the notion of *ioctl*. The ioctl is a utility that is responsible for device specific operations such as setting an instrument on line, sending an interface clear, and conducting a serial poll. It is anticipated that we will develop some type of GPIB manager to free application processes from the need to arbitrate bus use among the instruments attached to a given GPIB.

The *serial interface device driver* allows for controlling communication with external devices through a Xycom 428/1 interface module. An example of such an external device is the Omega Engineering Programmable Temperature Controller used by the RF subsystem. It is not intended that this device be used as a terminal interface, nor is the driver designed to handle terminal I/O.

The *stepper motor interface device driver* allows for controlling a Compumotor 1830 motor indexer. This indexer provides sophisticated motion control of positioners, slug tuners, diagnostics, and many other accelerator devices.

The *waveform digitizer driver* will handle all low-level functions associated with a CAMAC LeCroy 8837 transient recorder, or any similar model.

OPI Software

The Operator Interface (OPI) is a network device utilizing a graphic windowing system. It may be located at any point on the facility network and maintain full functionality. The user has the ability to generate and alter control displays and have access to the database and sequencer. The present OPI hardware is a VaxStation II/GPX running the VMS operating system using the UIS window manager.

The next OPI efforts include adopting an open standard windowing system. The utilization of a standard will remove dependencies on specific vendor hardware and software. Two systems are currently being studied: X Windows from MIT [5] and the Network/extensible Window System (NeWS) [6] from Sun Microsystems. Also under consideration is a VME-based OPI.

There are three control system configuration tools under development: a graphics editor (to build control displays), a database

configuration task (DCT), and a state program editor. The development path for these tasks is to integrate them under the selected standard windowing system, with a common interface/response design and with facilities to allow moving from one to the other easily.

Graphics Editor. The graphics editor [7] allows the user to create color control displays in an interactive manner. Construction of displays is accomplished using a mouse in an OPI window. The operator can automatically connect to database channels by selecting one of many primitives (sliders, buttons, text updates, or graphs) and by giving the channel name.

Once a display has been created, it can be stored on disk as a metafile for later use or modification. This information includes display information such as screen location, channel information, and graphical information for items drawn (text, boxes, circles, etc.).

The metafile also includes *static graphic items*, such as labels, *process display items*, and *process control items*. The process display items include graphic and text representation of analog and binary data as well as plots, strip charts, waveform displays, and state program status. These are implemented as areas of the screen that are periodically updated. A special display execution module later reproduces the "picture" and sets up communication between the display and the database so that the functions (monitoring, control, etc.) selected by the operator are carried out.

Process control items currently include a variety of primitives to control binary and analog signals. *Buttons* control a binary output channel, read a corresponding input channel, and display the results. They also monitor state programs which are waiting for the operator's permission to proceed. *Sliders* control an analog output channel, read corresponding input channel, and display the results. Slider banks provide a grouping of sliders for control and monitoring of a group of analog channels as described above.

Database Configuration Task. DCT [8] is another OPI task that creates and maintains the process databases and the process variable directory. DCT is used to configure the process database. It is an interactive configuration tool, which currently runs on the SUN workstations. The configuration task supports database creation, modification, report generation, and deletion.

The process variable directory is created and maintained by the DCT. It contains a directory of all the process variables in the control system and their IOC ID, record type, and record number. It is loaded into the OPI and the IOC at start-up time. During operation, the directory is used by the display task in the OPI and the sequencer in the IOC to resolve database references to unique network addresses.

Sequence Editor. A configuration task to be implemented is one that allows the user to create state programs graphically. These graphics, containing bubbles, labels, and arrows, would then be translated into either state notation language, C programs, or tables.

AIP Software

For the GTA, artificial intelligence will be an integral part of the control system. Currently, the design of AI procedures is proceeding along a parallel but independent path of the IOC development effort. The initial interface to the AI resource will be a high-level network connection to a separate AI processor. As this software development effort matures, the AI resource may be ported to an IOC-resident AI hardware coprocessor. This processor would be from the same family as the standard IOC processing device.

Present AI work is being carried out on a Symbolics 3620 Lisp machine. The ABLE project prototype [9] was the first attempt to automate many of the tasks previously performed by accelerator physics experts. The ABLE prototype was built to find only dipole field errors in a beamline using beam trajectory data (beam position monitors). ABLE incorporates well-understood physical models to predict where errors in the beamline might occur. It is a coupled expert system [10] combining over 15 000 lines of FORTRAN simulation and optimization programs with a LISP knowledge base of rules. The prototype was tested on several beamlines at SLAC. It provided useful information about the problem in nearly every case that was tested.

This prototype has been incorporated into a package called the Generic Orbit and Lattice Debugger (GOLD) [11], which is a FORTRAN program that can be easily incorporated into many different control systems. This program is able to find beam kick errors by analyzing beam centroid data. Beam kick errors are manifested by dipole field errors, quadrupole field errors, beam energy errors, beam entrance errors, and monitor errors. This program has been used to analyze data and locate beam kick errors at CERN.

Expert systems are also being investigated for use in injector control.

ICP Software

The Integrated Control Processor (ICP) is a computer that coordinates the activities of more than one IOC. IOCs are grouped functionally and associated with an ICP. Software residing on the ICP will be designed to coordinate the interaction of software components that are distributed among more than one IOC. The sequencer will be an important part of this integration.

Conclusion

The GTA control system provides an environment in which state-of-the-art accelerator automation can be developed. It provides flexibility and allows for growth where needed.

References

[1] P. Clout, "Accelerator Control Systems and GTA," Los Alamos National Laboratory technical note AT-8:GTA:88-001, June 1988.

[2] L. Burczyk, J. Martinez, S. Mechels, "The Executives Guide for the I/O Cluster (IOC) - Preliminary Hardware Design," Los Alamos National Laboratory technical note AT-8:IOC:88-001, September 1986.

[3] J. Fiddler, L. Kirby, and D. Wilner, "Distributed Systems Using UNIX and VxWorks," presented at the BUSCON/88 West Conference, Anaheim, CA, February 1988.

[4] A. Kozubal, "Guide to the State Notation Language," Los Alamos National Laboratory technical note AT-8:SYS:88-001, April 1988.

[5] R. W. Scheifler and J. Gettys, "The X Window System," ACM Transactions on Graphics, Vol. 5, pp. 79-109, April 1986.

[6] Sun Microsystems, Inc., "NeWS Technical Overview," Part No. 800-1498-05, March 1987.

[7] V. Martz and R. Rothrock, "AT-Draw User's Guide," Los Alamos National Laboratory technical note AT-8:SYS:88-002 February 1988.

[8] R. Dalesio, "Database Configuration Task - User's Guide," Los Alamos National Laboratory technical note AT-8:SYS:88-003, March 1988.

[9] S. Clearwater and M. Lee, "Prototype Development of a Beam Line Expert System," in Proceedings of the 1987 Particle Accelerator Conference, Washington, DC, March 16-19, 1987, IEEE Catalog No.87CH2387-9, p. 532.

[10] L. Selig, "An Expert System Using Numerical Simulation and Optimization to Find Particle Beam Line Errors," Master's thesis, Stanford University, Knowledge Systems Laboratory Report No. KSL 87-36, June 1987.

[11] M. Lee and S. Clearwater, "GOLD: Integration of Model-based Control Systems with Artificial Intelligence and Workstations," presented at Workshop on Model-based Accelerator Controls, Brookhaven National Laboratory, Upton, NY, August 16-18, 1987.