PROGRESS ON A NEW CONTROL SYSTEM FOR THE BEVALAC*

S. A. Lewis, A. K. Biocca, R.D. Dwinell, J. R. Guggemos, L. L. Shalz,
W. L. Brown, G. S. Boyle, K. Fowler, D. L. Meaney
Lawrence Berkeley Laboratory, 1 Cyclotron Road, Berkeley, CA 94720

Abstract

Progress is reported on replacing a heterogeneous triplex of outdated control systems with a single system. Prominent use of hardware and software standards is made to allow for an incremental upgrade, for best use of rapidly changing technology, for off-the-shelf computer and I/O hardware, and for several commercial software packages. A non-hierarchical architecture with a distributed run-time database is employed.

Introduction

As the Bevalac enters its 35th year of operation, we are designing and implementing a third generation of the computer control system. In this paper, we discuss the goals and requirements of the new system, outline the implementation techniques being used, and then summarize progress to date.

Bevalac control systems have remained operational for about 12-15 years after their initial design—a span of several computer "generations." Early in their life-cycles, effort was concentrated on the basics: systems, networks, and graphics. The middle phases saw the peak of productivity from the perspective of high-level applications: closing loops, modelling, and data management. And the end portion was beset with increasing hardware reliability problems and low productivity for new applications as the systems reached saturation. Each new system demanded new hardware, engineering, software, and retraining for the operators. Although hardware costs dominated for the first half of the life-cycle, in the end software costs were larger. Studies for this[1] and other[2] future controls systems bear this out to an even greater extent.

Operations at present are directed from four different control rooms, representing about 8 operator stations, with about 1/3 of the settable/readable parameters (of a total of about 3000 to 5000) not interfaced to any computer system. The remaining 2/3 are roughly equally divided among three unconnected systems. Two of these[3] are 13 years old, based on 16-bit minicomputers; the third[4] uses 8- and 16-bit microprocessors and is 9 years old. All three use different interface technology and two varieties of proprietary networking. In combination they process about 100,000 asynchronous events unevenly distributed over a 4-6 second major machine cycle.

Goals and Requirements

Because the future growth rate of the Bevalac is not accurately known, the new control system must show a much more seamless behavior as new hardware technology is phased in almost continuously, but in unpredictable chunks. At all times, the bulk of the effort should be directed towards the high-level applications, for which the demand will actually increase with time. The programming staff size and costs must remain constrained. It is also becoming clear that new productivity tools (for program design—"CASE", graphics presentation, networking, and so forth) typically have voracious appetites for computer power—another incentive to plan for hardware turnover, and to use extra hardware to minimize software costs *over the life-cycle*.

Although the existing control rooms and their equipment must still be capable of independent operation, now any one control room must be able to operate the entire facility. Start-up, shut-down, and switch-over must be very rapid, requiring more "smarts" from the control system (including sequencing, fault analysis, and auto-tuning). Where manual operations are desired, presentations must be of the highest quality. Operators will have to accept new assignments readily, therefore the control consoles must present a uniform, intuitive "look and feel" (see [5] for a thorough discussion). Diagnostics must be available *during normal operations*, for the components of the system itself, as well as for the accelerator devices.

The old systems must be *incrementally* replaced as funding allows, and all remaining parameters brought under computer control. The many proprietary and disparate development environments need to be discarded, and a single, new one chosen that is likely to allow easy interchange of engineers and programmers among other LBL projects and that is not locked to specific vendors or instructions sets. Maximal use should be made of formal and industry standards and market-driven products (see [6] for an overview). Because the cost of hardware continues to decrease, it is better to defer as much as possible to future purchases—a strategy that is effective only if software and hardware standards are chosen carefully for their longevity and widespread acceptance.

Architecture

The layered architecture is shown in Figure 1. The "front-end" is externally synchronized to machine interrupts down to the finest level required, and performs all interfacing, scaling, smoothing and time-critical operations. The "high-level" deals with display formatting, archiving of data sets, sequencing, and global fault analysis on a much slower time scale. These two layers are connected through the "datapool."

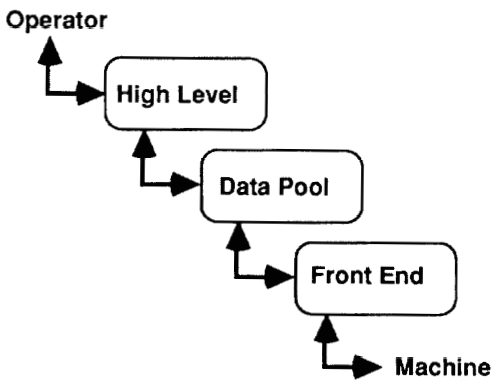

Figure 1: Layered Architecture. Results flow upward from the machine to the operator and commands flow downward.

In this "flat" architecture the two layers are placed in different subsystems. These are highly replicated and connected by a single logical network, as shown in Figure 2. The datapool is distributed within the front-end. Each "area" behaves like an independent

*This work was supported by the US DOE under Contract No. DE-AC03-76SF00098.

CH2669-0/89/0000-1645$01.00©1989 IEEE

subsystem, yet there is full, direct connectivity between areas. Analysis[1][2] shows that most data transmission will occur within areas, which are naturally occurring centers of activity, such as a control room and its associated accelerator.

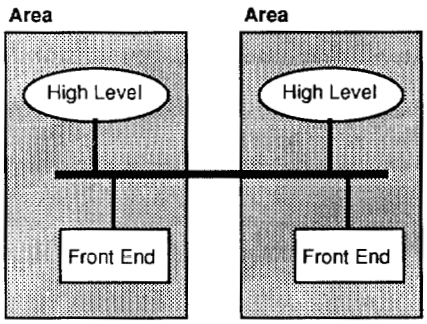

Figure 2: Logical Network. High-level and front-end components are directly connected. Most traffic stays within an area. The data resides in the front-end.

The flow of data is presented in Figure 3. Each high-level task is driven by a "device list." It queries a "locator" task to determine the distribution of the needed data, then exchanges that data directly with one or more responding tasks in the front-end. Similarly, front-end tasks are configured from a "control list" and so inform the locator of the devices they control.

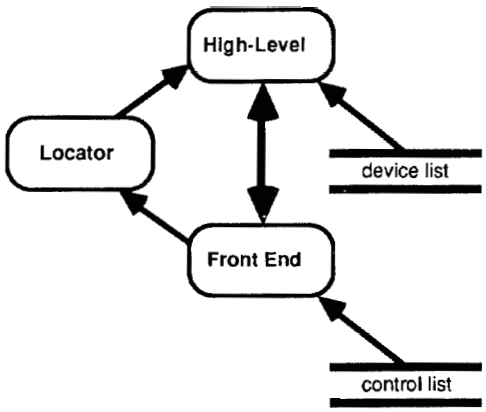

Figure 3: Dataflow among Tasks. High-level and Front-end tasks are list driven. They use the Locator task to learn each other's "address," then exchange data directly.

Data exchange can consist of a single transaction, much like a function returning a result. Or it can be a two step process: first, a request for the data is made; then, based on a time period or a machine "cycle," the response is returned in a "call-back." The second step is repeated continuously on behalf of long-lived tasks such as operator displays. The call-back may be further restricted to only *changed* data. Together, these techniques have the properties of minimal data movement, of ensuring that data is always fresh, and of guaranteeing that "chunks" of data come from the same cycle.

The device list can be built in to the task, it can be read from a disk file, or it may be synthesized by a generic screen manager as an operator configures his display station (discussed below). The underlying data moving mechanism must support the need for requesting and responding tasks to dynamically alter their connections for any of several causes: change of device list by requester, change of control list by responder, inactive requester, or inactive responder.

Implementation

The network is built from Ethernet coaxial and fiber-optic segments connected with repeaters. Bridges will be added if area traffic isolation is needed to increase the effective bandwidth. Loose coupling via gateways will be used to reach experimental and administrative areas. Major increases in performance can be straight-forwardly accommodated by using a high-speed backbone such as the emerging FDDI standard.

The high-level console computers are general purpose, high-resolution, color engineering workstations. Each is self-contained, having enough cpu power, ram, graphics hardware, and disk storage to present all needed programs, screen displays and so forth. They rely on the network only for data exchange, for accessing a central archive, and for occasional software updates. They utilize the Unix™ operating system, the TCP/IP network protocol, and (soon) the X-Windows graphics protocol. As the need for more consoles grows, more workstations will be purchased. Performance increases will be accommodated with upgrades and replacements—compatibility is required only at the level of the standards just mentioned.

Other high-level functions such as automatic sequencing and fault analysis can be placed on computers more specialized for compute-bound applications. On the other hand, a model, which is essentially a mathematical entity with inputs and outputs, logically belongs at the front-end. It might be implemented, however, on similar hardware (again, see[5]).

The front-end is built from VME crates. Within each crate, one "primary" computer is dedicated to managing the datapool. This is typically a single board that also contains the Ethernet interface, so it does not load the backplane. The actual real-time activities take place on "secondary" computers that are closely coupled with off-the-shelf I/O boards or entire commercial I/O subsystems, or sometimes configured with on-board I/O circuitry. At present, we are designing only one type of custom I/O board, to interface with the older systems that have no Ethernet capability. (This approach "hides" the older systems in the new front-end.) Interfaces also exist from VME to older systems such as CAMAC.

All VME computer boards contain sufficient local memory to hold their programs and local data. They will use a simple kernel, and most will support a real-time operating system with TCP/IP (required on the primary). The backplane can be used as either a fast LAN or for direct memory coupling. Groups of boards that support the VSB standard can form tightly-coupled subsystems with little VME load. Expansion will be accommodated with faster boards, by using more boards and/or crates, by direct coupling of VME crates, and eventually by moving to FutureBus (which offers a planned compatibility path).

Both high-level applications code and real-time code are developed identically in Unix using the "C" language (with migration to C++ anticipated). VME tasks are downloaded and debugged using VxWorks™, a commercial product that supports TCP/IP networking for both development *and* run-time activities.

Applications programs use the Remote Procedure Call (RPC) for task to task communications. We have taken the lowest level of this public-domain[7] protocol (available on most systems supporting TCP/IP) and added some functionality that reduces the

programming effort for typical accelerator applications. We use the UDP transport mechanism since this simple datagram facility closely models the type of transaction we want. RPC directly supports the dynamic mechanisms required for all configuration situations described in the Requirements section. Because RPC contains an efficient[8] data-translation facility called the External Data Representation (XDR)[9], different cpu instruction-set architectures can be used either for consoles or VME boards. This is particularly attractive in light of the high-performance RISC computers now beginning to become available.

We use a graphics-based, "windows, icons, menus, pointer" ("WIMP") human interface. Everything from mimic diagrams, to trend logs, to simulated meters can be flexibly accommodated. Input from the operator takes place using simulated buttons, sliders, "pick" lists, check-boxes, and other now familiar user-interface techniques from the world of personal computers and engineering workstations. (Some conventional knobs will remain in use.) However, all of the "gimmicks" of managing multiple windows, decoding the screens, and so forth are handled by a single, generic task, "Picture," that is paired with every "real" application, greatly reducing the complexity of the latter programs since they have no graphic programming content.

In Figure 4 , the Picture task loads and presents screens. In response to operator activity, it decodes the screen objects and prepares a dynamic device list which it sends to the Application. The latter obtains "raw" response values for the devices (from the front-end), synthesizes the requested attributes as required, and passes them back to Picture. Picture updates the screen objects as needed. Command values are handled similarly in the opposite direction.

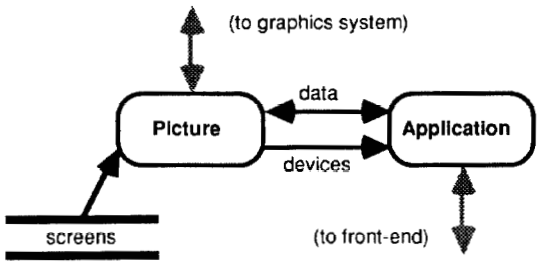

Figure 4: Separation of Graphics Component. Picture manages screens and their input and output objects. The Application task collects and disburses data from and to the Front-end. Only required data is exchanged between tasks.

We use a commercial package called DataViews™ to both prepare screens and animate them. The preparation phase proceeds much like any desktop presentation would—operators and other accelerator personnel build and modify the "control panel" screens over the life of the facility. These screens specify the device name or names to be associated with every input and output object and become the "lists" that drive the console applications—which are reduced to a few generic programs.

The DataViews package provides a *dynamic* referencing feature, so that common objects with many instances are updated by changing only the master template. This aids uniformity of detail across applications and eliminates redundant effort. The package provides very high-level runtime functions for screen and object management, as well as the low-level routines found in all traditional graphics packages—and like them, DataViews insulates us from all details of the underlying graphics of the workstation. Thus, although the present complement of consoles consists of Sun 3/60's with SunView™ and no hardware graphics, we expect to make no changes when we upgrade to X-windows and when newer workstations are purchased with different graphics hardware.

Status of Project

We have in place 7 console workstations, and 3 VME crates. Two of the three older control systems are connected to the new network with high-speed links from VME crates. The third crate is being outfitted to control a few hundred new parameters.

The new console software is in use for large survey displays, trend spotting, and bulk operations such as major start-up and shut-down, and about 10% of the display workload has been moved. Prototyping is underway for more input-intensive disciplines such as the tuning of small sections of beamline. We expect to complete prototyping in mid-1989. A small and growing operations crew is maintaining the screen repertoire.

The first generation of production software is in place to support the basic features of the distributed database. It is expected that refinements will be complete by late-1989.

Plans call for 15 consoles on-line by mid-1990, with all high-level software moved from the two oldest systems, and six VME crates in service: three linked to the three older systems, and three with new devices. Funding levels will then determine how rapidly the remaining moves to the new system can take place, in this order: high-level functions from the third of the older systems; front-end applications from the first two older systems; front-end applications from the third older system. About half of the present effort goes into maintenance of the older systems.

References

[1] *Bevalac Upgrade Conceptual Design Report*, LBL-5183Rev, 128 (1987).

[2] W.K. Dawson, *et al.*, *Conceptual Design for the TRIUMF Kaon Factory Control System*, TRI-87-1 (1987).

[3] R.A. Belshe, V. P. Elischer, V. Jacobsen, "The Feasibility and Advantages of Commercial Process I/O Systems for Accelerator Control," *IEEE Trans. Nucl. Sci.*, **NS-22**, 1036 (1975).

[4] S. Magyary, *et al.*, "A High Performance/Low Cost Accelerator Control System," *IEEE Trans. Nucl. Sci.*, **NS-28**, 1461 (1981).

[5] V. Paxson, V. Jacobsen, E, Theil, "A Scientific Workstation Operator-Interface for Accelerator Control," *Proc. 1987 IEEE Accel . Conf., Wash, DC*, 556 (1987).

[6] E. Theil, V. Jacobsen, V. Paxson, "The Impact of New Computer Technology on Accelerator Control," *Proc. 1987 IEEE Accel . Conf., Wash, DC*, 529 (1987).

[7] *Remote Procedure Call Programming Guide*, Sun Microsystems, Inc (1986).

[8] M. Rosenblum, "The Performance of Sun's Remote Procedure Call," private communication.

[9] *External Data Representation Protocol Specification*, Sun Microsystems, Inc (1986).