ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S16MS02

Framework for Control System Development*

Carl Cork and Hiroshi Nishimura
Lawrence Berkeley Laboratory, 1 Cyclotron Road, Berkeley, CA 94720

Abstract

Control systems being developed for the present generation of accelerators will need to adapt to changing machine and operating state conditions. Such systems must also be capable of evolving over the life of the accelerator operation. In this paper we present a framework for the development of adaptive control systems.

I. INTRODUCTION

Several of the new generation of control systems hardware being developed today have the capability of fast, sophisticated control at all levels in the control hierarchy[1][2]. These systems are typically hierarchical and highly distributed with extremely high I/O throughput.

We have initiated the design of a framework for control system development which can accommodate the new architectures. This paper will present requirements, design decisions, and specifications that we have devised for this framework.

II. REQUIREMENTS

A. Adaptive

The control system must be adaptive. It must be capable of growth, evolution, and learning (supervised and self-taught).

The software for these systems is complex and generally in continuous development. The control system must be capable of growth during both commissioning and operational phases.

Many new control system algorithms such as model-based control, expert systems, neural networks, and fuzzy logic are emerging which look very promising in the accelerator control environment.[3][4][5][6]. A mechanism is required which is capable of evolution to accommodate these new control theories. The system must also be capable of arbitrarily complex combinations of these algorithms.

Most of these new control system algorithms are capable of either supervised or self-taught learning. This should prove to be extremely useful as an aide to finding 'golden *orbits*' in storage rings or as a means of reducing the complexity of data presented to the operator. The control system must facilitate this mechanism.

B. Hierarchical

The control system must support a hierarchical control structure. It must be capable not only of supporting the 'standard' supervisor-cell-local type of hierarchical control[7], but also each layer must be divisible into local subhierarchies. This latter requirement facilitates the incorporation of cascaded and adaptive control algorithms.

C. Distributed

The control system must support the underlying distributed hardware.

Many computer systems provide basic networking support. The control system must also incorporate mechanisms for the registration of computing services, the automated association of client and server, and the uniform representation of data transmitted between heterogeneous systems.

The control system must be designed to accommodate the known features of distributed control - such as error detection and recovery, virtual time synchronization, nondeterministic networks, concurrency, resource protection, and bandwidth-limited messaging.

D. Operational Continuity

The control system must support operational continuity. It must provide for dynamic, and transparent switching between compatible modules without interrupting operation.

Transparent switching is required to permit the exchange of control modules in the event where the system operation exceeds the bounds of the previous controller. This should be possible without bringing the system down and without leaving the machine uncontrolled. Sufficient machine state information should be transferrable to provide for 'bumpless' switching.

E. Dynamic Association

The control system must support the dynamic association of applications. Links between the control system and the application should be redirectable during normal operations. This is essential to provide for independent development of associated modules and also to provide support for the adaptive and operational continuity requirements listed above.

Dynamic association permits both application and control modules to be constructed without prior availability of the associated modules. Moreover, for client-server associations, the link process should not require specific knowledge of the server module (capability-based binding). It should be sufficient to specify the type of module and its interface, leaving the association mechanism to a third intermediate process.

F. Universal Graphical API

The control system must support a universal graphical application programming interface (API). Regardless of the operating system, windowing system, or window manager, the graphical application programming interface should be identi-

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

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S16MS02

cal. All that should be required is a recompilation for each graphical display workstation.

III. DESIGN

Based on the requirements listed in the previous section we have established the following design specifications.

A. Virtual Control Modules

The control system shall accommodate the adaptability, hierarchy, and continuity requirements by incorporating a recursive architecture. This may be expressed using the modified Backus-Naur Form (BNF) formalism which is often used to specify computer language syntax:

$$VirtualControlModule = ControlModule + [VirtualMachine]^{+} \quad (1)$$

$$VirtualMachine = Machine \mid VirtualControlModule \quad (2)$$

The *VirtualControlModule* is the control subsystem consisting of a controller (*ControlModule*) and one or more controlled objects (*VirtualMachine*). The *VirtualMachine* may consist either of the bare *Machine* or, recursively, of an additional *VirtualControlModule* subsystem.

The *Machine* represents the accelerator and its associated instrumentation. This system may be represented by a set of n measures (state variables) and its development over time may be expressed by trajectories of the state variables in state space. During the development and testing phases, the *Machine* might be replaced by a simulator which emulates all command and response characteristics of the real machine.

The *ControlModule* is required to counteract any motion of the machine system away form the stable operating point. The combined subsystem (*ControlModule* + *VirtualMachine*) should be asymptotically stable. In the adaptive control system the *ControlModule* is a mutable element. Its state parameters are dynamically adjustable, it might be layered, parallelized, or self-adaptive. It shall also be dynamically replaceable by an alternative *ControlModule* with synchronized exchange of control between the *ControlModules*.

This model can describe all of the standard control systems in use today. The following expressions represent a few such systems.

$$\begin{aligned} RemoteControlSystem &= ControlModule + Machine \\ SupervisedControl &= ControlModule \\ &\quad + \langle ControlModule + Machine \rangle \end{aligned} \quad (3)$$

A control system which accommodates *VirtualControlModules* will permit the control system to be modularly adjustable and to incorporate growth (and scalability) and evolution.

B. Distributed Task Synchronization

A synchronization mechanism shall exist to coordinate interaction with the machine elements and peer subsystems. The distributed machines shall incorporate partially ordered logical clocks to support virtual synchronization between coordinating processes across the network[8][9].

A multitasking environment shall be incorporated to provide synchronous and concurrent behavior on a single system. Task synchronization can be performed using any of the typical real-time mechanisms (e.g. semaphores, message queues, mailboxes).

C. Distributed Computing Services

A peer-to-peer message passing mechanism shall be implemented to satisfy the distributed communications requirements. This mechanism should have a programming interface which is independent of the network transport layer implementation. The design should be efficient enough to consider using it equally for local or remote task-to-task communications.

The control system should also support both message-based and remote procedure-based communication mechanisms. Message-based mechanisms will probably be best suited for event-driven processes which would normally be looping on an input message queue. Remote procedure-based communications will be best suited to transparent migration of library modules from local to remote configurations.

D. Object Communications Manager

An object communications manager shall be implemented to satisfy the dynamic task association requirement. The object communications manager will coordinate the interaction between applications and all other elements of the control system. The control system elements will be composed of software objects which interact to perform their assigned functions. Some of these objects will "advertise" their presence to external applications by registering with the object manager. External applications will query the object manager to select and associate with the advertised interfaces. The object communication manager permits the association to be dynamic and transparent. New control system objects can be substituted without requiring a restart of either the user applications or the control system modules. Moreover, the user application need not know whether or not the control system modules are operating locally or remotely - the interface is the same for both (the mechanism is similar to the X-windows byte-stream implementation). The operator interface applications are specific examples of applications which will use the object communication manager to interact with the control system.

All potential *ControlModule* and *VirtualMachine* modules must satisfy uniform interface requirements with respect to the object communications manager. This permits the modules to be dynamically replaced during operation and without requiring the reconfiguration of existing modules.

E. Network-Based GUI

A network-based graphical user interface (GUI) shall be incorporated to satisfy both the universal graphics application programming interface and the distributed control require-

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S16MS02

ments. The graphical display will be presentable on any candidate workstation on the network. The display application may reside either within the workstation or else on some remote computer. The interface must function in a heterogeneous environment and should function on a variety of platforms.

IV. FRAMEWORK

The following framework was established to implement the design specifications from the previous section. The framework is based on existing technology and/or standards. This was done not only to take advantage of commercial products and community efforts but also to guarantee a more timely implementation of the composite system. We attempted to select a minimal framework to avoid an overly restrictive development environment.

A. *Eiffel Object-Oriented Environment*

The *Eiffel* object-oriented programming language and application environment will be implemented to satisfy the virtual control module design specification[10].

Object-oriented environments support modular software development, data abstraction, polymorphism, and dynamic binding - all of which are required to satisfy the virtual control module specification. *Eiffel* in particular also supports automatic memory management, multiple inheritance, enhanced reusability, and a special reliability feature (assertions) which supports a 'software by contract' design methodology. The language specification is now in the public domain and has a strong international and educational backing which should assure its continual evolution.

B. *POSIX 1003.1 Operating System*

The standard multitasking operating system will be a real-time operating system which is compliant with the IEEE specifications for a portable operating system interface (POSIX 1003.1). It will also support the real-time extensions (IEEE 1003.4) which are presently awaiting finalization.

At the higher machine architecture level we will select either the LynxOS or else the Chorus real-time operating system[11][12]. Both of these are network-based, POSIX compliant, and support real-time computing features.

At the lower machine control level we will use the VxWorks operating system[13]. This is a network-based, embeddable real-time operating system with a wide support base in the VME environment. VxWorks will provide a POSIX compliant interface when the real-time extensions are finalized.

C. *Distributed Computing Environment*

The distributed computing environment (*DCE*) will be implemented using the *OSF/DCE* utilities from the Open Software Foundation (*OSF*)[14]. These utilities will provide basic services for remote procedure calls, network security, and distributed file systems. The *OSF/DCE* is layered upon any POSIX compliant interface and is composed of elements which are available commercially today.

These utilities will soon be available on all major variants of the UNIX operating system. Initially, it will be available from *OSF* on their POSIX compliant operating system, and later it will be available from the Unix Software Laboratory (USL) on their SVR4 UNIX base. The Open Network Computing (ONC) utility set which is the dominant remote procedure call facility in use today will probably adapt to incorporate *DCE* compatibility.

D. *Object Request Broker*

The object communication manager facility will be provided by the Object Request Broker (*OMG/ORB*) which is being specified by the Object Management Group in collaboration with several large computer companies[15]. Early versions of this facility will be available from Hewlett-Packard and from Sun Microsystems.

A working example of this facility, ToolTalk, is currently available from Sun Microsystems for use on their workstations[16]. Our first *ORB* compliant applications will probably be based on this toolkit.

E. *X-Windows, Motif, and IEEE 1201.1*

The network-based graphical user interface will be provided by the MIT X-Window system, the *Motif* graphical user interface, and the evolving IEEE 1201.1 universal application programming interface libraries.

The only universal, network-based, window environment available today is the X-Window system. The latest release (X11R5) is fast, supports scalable fonts, and runs on every major UNIX workstation. A large amount of public domain software is available for this windowing environment.

Unfortunately, there are several competing, incompatible, graphical user interfaces available for the X-Window system. The *OpenLook* GUI is being promoted by AT&T and Sun Microsystems, while the *Motif* GUI is being promoted by the Open Software Foundation (OSF) and most of the other workstation vendors. However, to our knowledge the *Motif* window manager and application programming interface is also the only environment which runs universally on all present Posix compliant systems. Moreover, a number of *Motif* compliant GUI tools are available for most of these platforms.

The IEEE 1201.1 committee is developing a specification for a standard GUI programming interface which can be used with any of the X-Window GUIs in use today. We will adopt this standard when it becomes available, but in the meantime we will use the Xm-based toolkit from *OSF* for their *Motif* GUI. Wherever possible, we will also be using the *Eiffel*-based graphics toolkit from Interactive Software Engineering[17].

V. PROJECT STATUS

We have developed several prototype components to test some elements of this framework.

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S16MS02

A class library for accelerator modeling and simulation has been constructed using *Eiffel*. Another *Eiffel* class library for the hardware database access is also being developed which interfaces with the LBL/ALS control system. Using these class libraries, one can create accelerator models dynamically with on-line and real-time access.

Several network-based, object-oriented device handlers have been written running under the VxWorks operating system on a VME target system. These handlers are being rewritten in *Eiffel*. An object management broker for VxWorks is also in progress.

VI. REFERENCES

[1] S.A. Lewis, A.K.Biocca, R.D.Dwinell, J.R.Guggemos, L.L.Shalz, W.L.Brown, G.S.Boyle, K.Fowler, and D.L.Meany, Progress on a New Control System for the Bevelac, IEEE Part. Accel. Conf., Vol. 89CH2669-0(1989)1645.

[2] S.Magyary, M.Chin, C.Cork, M.Fahmie, H.Lancaster, P.Molinary, A.Ritchie, A.Robb, and C.Timossi, Advanced Light Source Control System, IEEE Part. Accel. Conf., Vol. 89CH2669-0(1989)74.

[3] M.Lee, S.Clearwater, E.Thiel, and V.Paxson, Modern Approaches to Accelerator Simulation and On-Line Control, IEEE Part. Accel. Conf., Vol. 87CH2387-9(1987)611.

[4] S.H.Clearwater, and M.Lee, Prototype Development of a Beam Line Expert System, IEEE Part. Accel. Conf., Vol. 87CH2387-9(1987)532.

[5] J.E.Spencer, Accelerator Diagnosis and Control by Neural Nets, IEEE Part. Accel. Conf., Vol. 89CH2669-0 (1989) 1642.

[6] Bart Kosko, Neural Networks and Fuzzy Systems. A Dynamical Systems Approach to Machine Intelligence, Prentice Hall, Englewood Cliffs, NJ, 1992.

[7] B.Liptak, and K.Venxzel, Instrument Engineers' Handbook, Process Control, Chilton, Radnor, PA, (1985) 713.

[8] C.Fidge, Logical Time in Distributed Computing Systems, IEEE Computer, Vol.24,No.8(1991)28.

[9] Isis-A Distributed Programming Environment, Version 2.0 user's guide and reference manual, Comell University (April 1990).

[10] Bertrand Meyer, Object-Oriented Software Construction, Prentice Hall, Englewood Cliffs, NJ, 1988.

[11] LynxOS, Lynx Real-Time Systems, Inc., Los Gatos, CA, USA.

[12] CHORUS/MIX, Chorus Systems, Beaverton, OR, USA.

[13] VxWorks, Wind River Systems, Inc., Alameda, CA, USA.

[14] Distributed Computing Environment, Open Software Foundation, Cambridge, MA, USA.

[15] Object Request Broker, Object Management Group, San Francisco, CA, USA.

[16] ToolTalk (Beta) Programmer's Guide, Part No: 800-6093-05, Sun Microsystems, Inc., Mountain View, CA, USA.

[17] Eiffel Graphics Library, Interactive Software Engineering, Inc., Goleta, CA, USA.