DISTRIBUTED DATA ACCESS IN THE LAMPF CONTROL SYSTEM*

S. C. Schaller and E. A. Bjorklund, MP-1, MS H810
Los Alamos National Laboratory, Los Alamos, NM 87545

Abstract

We have extended the Los Alamos Meson Physics Facility (LAMPF) control system software to allow uniform access to data and controls throughout the control system network. Two aspects of this work are discussed here. Of primary interest is the use of standard interfaces and standard messages to allow uniform and easily expandable inter-node communication. A locally designed remote procedure call protocol will be described. Of further interest is the use of distributed databases to allow maximal hardware independence in the controls software. Application programs use local partial copies of the global device description database to resolve symbolic device names.

Introduction

The LAMPF computer control system upgrade project reached an important milestone on January 2, 1987, with the retirement of the original Systems Engineering Laboratory SEL-840 control computer. Having replaced the original central computer controls software with new software running on a VAX 780, we turned our attention to upgrading the remote computer network which provides dedicated data acquisition and control tasks for the central control computer. At the Second International Workshop on Accelerator Control Systems at Los Alamos in October, 1985, we discussed plans for this upgrade (see reference [1]).

Our goals for the network upgrade included extending the data acquisition hardware independence and flexibility achieved through the control VAX software, and providing a simple and straight forward way to distribute application program functionality across several network nodes.

In this paper we discuss the additions in hardware and software that have been made to the LAMPF control system network. The first part of the paper describes the upgraded controls network organization. The next part discusses control system extensions to provide uniform data access on and between remote network nodes. The final part describes the remote procedure call interface we implemented to allow simplified communication between application programs running on different network nodes.

LAMPF Control System Network Organization

The upgraded portions of the LAMPF control system network are shown in figure 1. The central control computers (labeled PROD and DEVEL) are Digital Equipment Corporation (DEC) VAX 780s connected to the accelerator operators' consoles in the central control room. Usually one VAX is used for production and one for software development. The two control VAXes are part of a cluster sharing the same set of cluster disks.

An Ethernet cable running the length of the accelerator and extending on into the experimental areas connects several DEC micro-VAXes used for dedicated data acquisition and control tasks. Each of these micro-VAXes is directly connected to one or more CAMAC crates giving access to real-world data. In several cases (notably IC and ISTS) local operator consoles are attached to the micro-VAXes to support independent hardware development and to allow local monitoring of the ion sources.

The central control VAXes run the VAX/VMS operating system and provide access to a multitude of application programs through standard accelerator operators' consoles. Since the IC and ISTS nodes needed standard consoles also, we decided to use the micro-VMS operating system on them to allow the use of existing LAMPF software. We had no trouble running the needed VMS software -- both device drivers and application programs -- on the micro-VMS systems. In several cases, environmental considerations and run-time response requirements ruled out the possibility of using disk-based operating systems such as VMS. For these systems we implemented our software in the VAXELN environment. VAXELN is a system for building memory-resident, real-time systems which can be down-loaded (and debugged) over Ethernet.

Application programs which supply the interface to the accelerator operators run primarily on the central control VAXes. Through references to symbolic device names they can acquire data either locally or from remote nodes -- micro-VMS or VAXELN -- across the network. Application programs running in the remote nodes use symbolic device names to access locally connected data.

We chose to use DECnet on our Ethernet LAN because of the large amount of software support available for it and because our time-critical tasks are concentrated in single nodes -- not distributed across the network. Since our network includes more than one operating system, it made sense to let DECnet handle the basic communications tasks of message delivery, routing, and error handling.

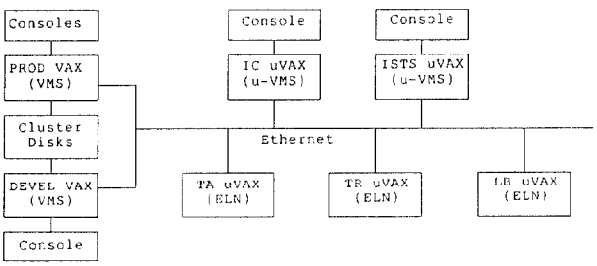

Figure 1 - LAMPF Control System Network

*Work supported by the US Department of Energy

CH2387-9/87/0000-0745 $1.00 © IEEE
PAC 1987

LAMPF Control System Data Access Software

The data access software on the central control computers was designed to provide a uniform interface for application programs. The interface provided a flexible and powerful mechanism for application programs to acquire data and issue commands in a hardware-independent manner. In particular, application programs were required to access data only through symbolic device names. The general properties of this software have been described elsewhere (see references [2], [3], and [4].

As part of the controls network upgrade, we wished to provide this same power and hardware independence to applications programs running on both the central control computers and on the remote network nodes.

Figure 2 shows the connections which exist between the various parts of the LAMPF data access software. Each node has a device description database which is used (implicitly) by application programs to resolve references to symbolic device names. Using information in the database, the data access software decides whether the device reference is to be handled locally or on a remote node. If the device is local, the data access software talks to the local CAMAC hardware. Otherwise, the data access software communicates over the network with a CAMAC server which, in turn, drives the CAMAC hardware. The communication may also be established with a remote data server which itself can make references to locally defined devices via symbolic names.

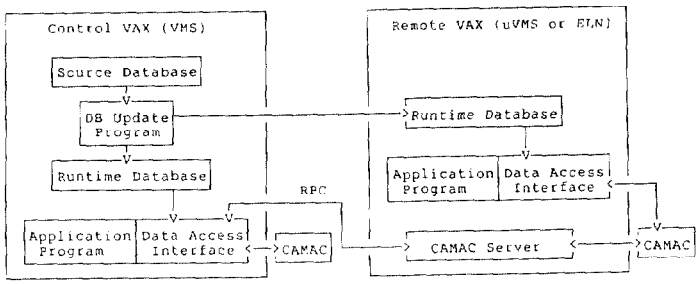

Figure 2 - Data Access Software

Distributed Database Maintenance

The LAMPF device description (Device Table) database actually exists in two forms, not unlike the existence of a Pascal procedure in both a source and an object form. The "source" form of the database is an ASCII database maintained with a commercial database management system (DRS, supplied by Advanced Data Management, Inc.). This source database exists and is maintained only on the cluster disks attached to the central control computers.

The "object" or run-time form of the device description database contains translated information derived from the source database. This technique allows rapid run-time access to the device information at the expense of having programs that must translate between the two forms of the database.

A version (full or partial) of the run-time device description database exists on each network node requiring access to data and controls. The control computers each have a run-time database that contains a description of every device in the control system. The run-time database in a remote node contains information only on devices immediately accessible to that node.

On VMS and micro-VMS nodes, the run-time database is implemented as a VMS global section, which allows multi-process access to the device information and uses the VMS paging mechanism to provide efficient input and output. On VAXELN nodes, the run-time database is implemented as an ELN "area" which allows multi-job and multi-process access to the device information. All ELN areas are memory resident, so there is no concern with input/output efficiency.

The source version of the device description database is updated using tools supplied by the commercial database management system. We have developed a set of update and report programs. The update programs use the full-screen update facilities of the database management system.

The run-time databases on the central control VAXes are updated automatically whenever the source database is modified. When a device which is defined on a remote micro-VMS node is updated, an incremental update of the remote run-time database is also automatically performed. Currently, run-time databases in VAXELN nodes can only be modified by reloading the entire VAXELN system across the network. We hope to be able to perform incremental updates of VAXELN run-time databases in the near future.

It is difficult to keep distributed run-time databases in agreement. Automatic update of the remote databases in parallel with the central control VAX database is an attempt to deal with this problem. However, if a remote computer is not available when a relvant update is made, we must have a mechanism for bringing the databases into agreement when that node next becomes available. Our current solution is administrative; we are seeking automated solutions.

Remote Procedure Call System

The remote procedure call (RPC) model for communication in distributed systems provides a mechanism whereby a process running on one node can "call", using standard procedure calling semantics, another routine that executes on a different machine. The advantages of RPCs include the simplicity of the calling and listening program interfaces. The details of link handling, message passing, error recovery, and operating system peculiarities are hidden from the user programs.

We implemented a remote procedure call interface as the standard inter-node communication mechanism in the upgraded LAMPF control system network.

The RPC system structure is indicated in figure 3 for the case of a synchronous call with no errors. (The LAMPF RPC interface also supports asynchronous calls and does extensive error handling. See reference [5] for more details.)

The RPC interface is divided logically into a "caller's interface" and a "server's interface." On the caller's node, the calling process is linked to a stub which has the same name and the same set of parameters as the remote procedure. The stub routine passes its parameters, along with some additional information, to the RPC interface remote call routine. If the call is synchronous, the calling process is blocked until the remote procedure completes. When the remote procedure completes, the reply message is unpacked by the RPC interface process-reply routine, which writes the values of output parameters into the caller's variables.

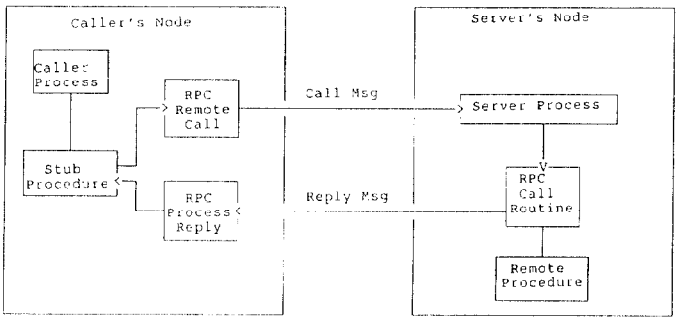

Figure 3 - Remote Procedure Call System Structure

On the server's node, remote procedures are contained in special processes called "server processes." The server process in figure 3 listens for call messages, determines which procedure should be invoked, and passes the address of that procedure, along with the call message, to the RPC interface procedure-call routine. This routine re-creates the argument list from information in the call message and then calls the specified procedure. When the procedure returns, the output parameters are packed into a reply message which is sent back to the caller process.

To bind a caller to a remote procedure, the RPC interface must know the name of the server process containing the remote procedure, the node on which the server is running, and the name of the procedure to call. The node and server names are used to create a DECnet "logical link" between the calling process and the server process. The procedure name is translated into a "procedure id" value and sent to the server process in the call message. By convention, procedure id zero is the id of a diagnostic echo routine provided by each server's RPC interface.

Whenever possible the binding information is provided by the stub procedure. In some cases, however, part of this information must be supplied by the caller. For example, if the same service is available on more than one node, or if the same procedure is available in more than one server, then the caller must supply the node and/or the server name.

The RPC interface is responsible for handling errors detected by the remote procedure, by the RPC interface itself, and by DECnet. If an error is detected, the RPC interface returns a status or raises an exception in the caller's process and, in some cases, shuts down the DECnet link.

RPC stub routines are not automatically generated in our system, but we have tried to make stub generation as easy as possible. In most cases, a stub routine needs only to supply a description of the parameters to be passed and call the appropriate RPC interface routines.

As indicated in figure 2, the LAMPF data access software uses remote procedure calls to make requests to CAMAC servers on remote nodes. The RPC interface is also used to communicate with a general "data access server" which handles requests for data from devices addressed by their symbolic device names.

Conclusions

We feel we have achieved our goals of extending the data acquisition hardware independence and flexibility to much of the LAMPF control system network. We have found that the update of distributed run-time databases is a manageable problem.

We are currently in the process of redesigning several large application programs which have portions running on several network nodes. The RPC paradigm is proving to be very useful in producing well-structured systems of programs that are distributed across several network nodes.

Timing tests reported in reference [5] indicate that the LAMPF RPC interface message exchanges add approximately 20% to the DECnet times. We find this to be an acceptable price to pay for the power and flexibility provided by remote procedure calls.

In the near future we hope to replace all of the aging PDP-11 computers in the control system network. We also plan to provide limited access to data between remote nodes using an RPC interface.

Acknowledgments

We would like to thank Gary Carr, Jim Harrison, and Pat Rose.

References

[1] S.K. Brown et al, Nucl. Instr. and Meth., A247, pp. 122-125 (1986).

[2] S.C. Schaller and P.A. Rose, IEEE Trans. on Nucl. Sci., NS-30, pp. 2308-2310, (1983).

[3] S.C. Schaller and J.K. Corley, Proceedings of the Digital Equipment Corporation Users Society, Fall 1983, pp. 471-475, Oct. 1983.

[4] S.C. Schaller, J.K. Corley, and P.A. Rose, IEEE Trans. on Nucl. Sci., NS-32, pp. 2080-2082, (1985).

[5] E. Bjorklund and S.C. Schaller, Proceedings of the Digital Equipment Corporation Users Society, Spring 1987, (to appear).