γ

CONF-880364--21

UCRL-97871
PREPRINT

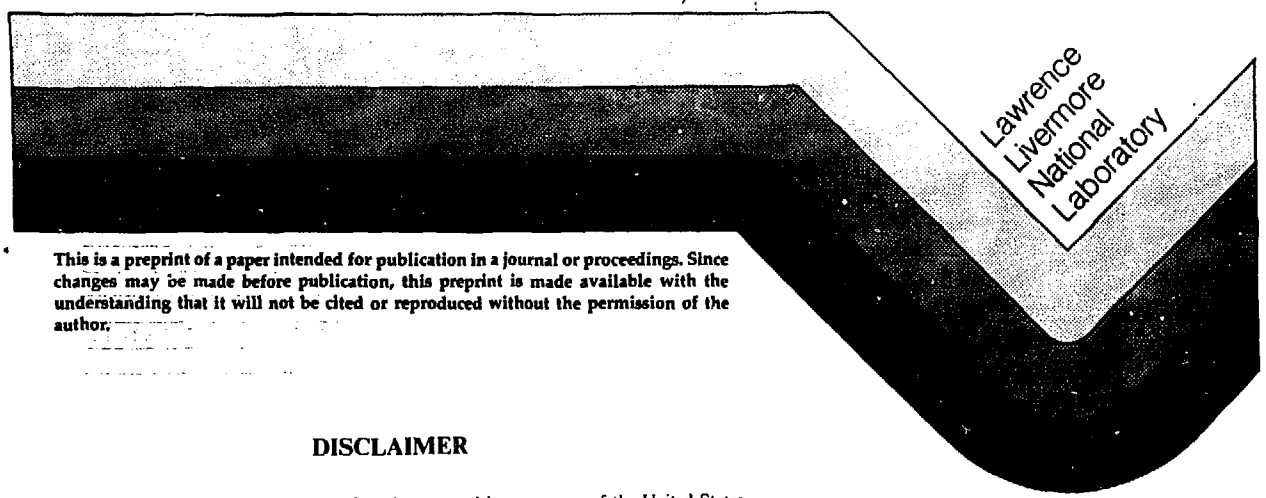
MAY 0 1988

An Integrated, Multi-vendor Distributed Data Acquisition System

David N. Butner, Marena Drlik, William H. Meyer,
Jeffrey M. Moller, and George G. Preckshot

This paper was prepared for submittal to the
Proceedings of the Seventh Topical Conference on High Temperature Plasma Diagnostics,
Napa, California, March 13–17, 1988

March 3, 1988



This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

An Integrated, Multi-vendor Distributed Data Acquisition System*

UCRL--97871

DE88 008900

David N. Butner, Marena Drlik, William H. Meyer,
Jeffrey M. Moller, and George G. Preckshot
Lawrence Livermore National Laboratory,
University of California,
Livermore, California 94550

Abstract

A distributed data acquisition system that uses various computer hardware and software is being developed to support magnetic fusion experiments at Lawrence Livermore National Laboratory (LLNL). The experimental sequence of operations is controlled by a supervisory program, which coordinates software running on Digital Equipment Corporation (DEC) VAX computers, Hewlett-Packard (HP) UNIX-based workstations, and HP BASIC desktop computers. An interprocess communication system (IPCS) allows programs to communicate with one another in a standard manner regardless of program location in the network or of operating system and hardware differences. We discuss the design and implementation of this data acquisition system with particular emphasis on the coordination model and the IPCS.

---

*Work performed under auspices of U.S. DOE by LLNL under Contract W-7405-Eng-48.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MTX Data Acquisition Scenario

We are developing a computer system to acquire data from the Microwave Tokamak Experiment (MTX) [1]. This magnetic fusion experiment uses an MIT-developed data acquisition software package that runs on DEC VAX computers and LLNL-developed data acquisition software packages that run on HP series 300 UNIX computers and HP series 200 BASIC desktop computers. Our problem is to develop an integrated data acquisition system that uses this variety of software and hardware, that supports MTX and future experiments, and that allows us to add higher performance computer equipment as it becomes available.

In magnetic fusion experiments, a plasma is generated in the tokamak for a period of a few seconds every few minutes; each plasma episode is called a shot. Our data acquisition system is oriented toward such a shot scenario, but many of its features can be used for or adapted to a continuous process or experiment.

The tokamak used in MTX was also used for the Alcator experiment at MIT. The tokamak and various support equipment, such as power supplies and diagnostic equipment, were moved to LLNL in 1987. The data acquisition system developed at MIT, called MDS [2], acquires data from CAMAC modules and stores the data in files. MDS allows a diagnostician to specify the types of modules used in a system, the pre-shot and post-shot commands for each module, and the files into which the data

from each acquired signal are stored. MDS also allows the diagnostician to specify that certain signals be automatically displayed after the shot. MDS is written in FORTRAN to run on VAX computers; it relies on VMS services so moving it to another type of computer would be impractical.

A previous magnetic fusion experiment at LLNL [3] used HP computers running UNIX and BASIC for diagnostic data acquisition. MTX diagnosticians want to use some of this equipment because we have a large investment in previously developed software for these computers. The HP computers also allow for more complex interactive control of diagnostics than does MDS.

In addition to acquiring experimental data and storing it in files, an integrated data acquisition system must keep track of where the data for each experiment and each diagnostic are stored so data can be retrieved in a reasonable manner. Our system includes a data location database for this purpose. The details of tracking and retrieving the data in a standard manner are discussed elsewhere [4].

This paper discusses two important features of the MTX data acquisition system: the shot model and the IPCS. The shot model is the key to coordinating a variety of data acquisition subsystems, and the IPCS provides a straightforward method so tasks on various computers can communicate with each other.

Coordinating Data Acquisition--the Shot Model

We created a method to coordinate data acquisition, the shot model, that relies on modular software subsystems and a state machine. In this shot model, we separated the diagnostic subsystem programs and the data location database programs into modules (Fig. 1). The modular approach provides flexibility in designing the data acquisition configuration for an experiment. Diagnostic subsystems can be modified, or new subsystems requiring different hardware/software interfaces can be added without affecting the basic shot model. The modular approach also divorces subsystem-dependent communication problems from the shot supervisory program and places this burden on the diagnostic subsystem programs.

The shot model is implemented by a shot supervisory program and diagnostic subsystem programs executing concurrently as a state machine. The rules that govern the sequence of events during a shot can be considered as a state diagram (Fig. 2). The shot supervisory program, SHOTSYNC, is the coordination interface between the shot leader and the diagnostic subsystem control programs. HPCONTROL and MDSCONTROL, the diagnostic subsystem control programs (Fig. 1), translate SHOTSYNC's state transitions into actions for their respective subsystems. The supervisory database program, DBSERVER, coordinates the collection of data location and shot configuration information.

SHOTSYNC accepts commands from a shot leader via a user-interface program. The shot supervisory program also compiles a list of diagnostic subsystems, supervisory databases, and various status-monitoring programs and coordinates their activities according to the state diagram. SHOTSYNC has no prior knowledge of what subsystems will be part of a shot. Instead, we use a "dynamic check-in" technique by which programs to be coordinated notify SHOTSYNC, via IPCS, that they need to receive shot coordination messages. SHOTSYNC maintains a list of checked-in diagnostic subsystems, and each subsystem stays on the list until it dies or checks out. For each shot, SHOTSYNC supplies the list of participants and other shot information as a package to the database server at the times required by the state diagram. Thus we automatically record the actual shot configuration. SHOTSYNC also sends shot cycle information to a logging device in the control room and supplies status information to user-interface programs. A diagnostic subsystem that wants to participate in a shot must send a check-in message to SHOTSYNC and must respond as appropriate to SHOTSYNC's shot coordination messages.

Currently we have two kinds of diagnostic subsystems: MDS acquiring from CAMAC modules, and HP BASIC computers and UNIX-based workstations acquiring from LLNL-developed diagnostics.

Multiple MDS subsystems can acquire data. Each subsystem is coordinated by an MDSCONTROL program, which communicates with SHOTSYNC using IPCS and with the MDS subsystem using MDS

events. Each MDS can reside on a different VAX computer, which allows us to increase system capability by adding more computers and running in parallel.

HPCONTROL runs on a UNIX-based workstation, talks via IPCS to SHOTSYNC, and uses the HP proprietary SRM protocol to communicate with the BASIC desktop computers. A major difference between HPCONTROL and MDSCONTROL is that HPCONTROL acts as an IPCS sponsor for many HP desktop diagnostics. All HP desktop IPCS communication is funneled through HPCONTROL.

The Role of IPCS

IPCS is an easy-to-use, portable software system for passing messages between tasks that reside anywhere in a network of computers [5]. By easy-to-use, we mean that programmers using IPCS need not know anything about the networks used by IPCS, and they have a few straightforward procedures to call to use IPCS. By portable, we mean that IPCS is written so it can be easily moved to another type of computer or operating system; portability is achieved by using a portable language and by judiciously using operating system services.

To send a message via IPCS, a programmer specifies the name of the receiving task, an optional set of class variables, and an optional variable-size data block (Fig. 3). A task that receives a message may use the message identification to return the message, optionally changing the

class variables. The receiving task may also use the message identification to send a reply message, optionally including class variables and a data block. The task that returns or replies does not specify the name of the task that is to receive the message; IPCS routes the message to the task that sent the original message.

To implement the logical structure of programs shown in Fig. 1, two features of IPCS are particularly useful: "addressing by task name" and the class variables. To send a message to another task, only the name of the task is specified, not the name of the computer on which the task runs. The class variables are eight 16-bit fields associated with every IPCS message; they are used to classify messages into a hierarchy of message types.

The method by which tasks communicate with SHOTSYNC is an example of the addressing-by-task-name feature of IPCS. When any program in the system wants to communicate with the shot synchronization task, it sends a message using only "SHOTSYNC" as the name of the message receiver. It does not matter where the SHOTSYNC task is running in the network of computers; IPCS finds the SHOTSYNC task and delivers the message. This feature allows us to run a given task on the most appropriate computer in the network. Without modification, a task image file may be moved among computers running the same operating system on the same hardware. If a task does not use a special feature of a computer or operating system, the source code is only recompiled and relinked to run on a different computer or

operating system. With this portability of application code, we can easily change the processor load of various computers, operate with one or more computers unavailable, or add more computers to the network.

The use of specific names for each task in the system is optional. Usually, only tasks that provide services, such as SHOTSYNC and DBSERVER, are named. Tasks that do not provide services are not named; instead IPCS creates a unique name for such tasks when they start. The reply and return message capabilities of IPCS ensure that response messages will be passed back to the appropriate task.

Class variables are used to implement a hierarchical structure of message types. The first class variable, called the major class, specifies a category of messages. In some cases, the major class designates a message to be handled by a certain task, such as DBSERVER. In other cases, the major class designates one of several types of messages handled by one task; for example, SHOTSYNC handles messages whose major classes are shot_registry and shot_command, among others. The second class variable, called the minor class, is used to specify functions involved specifically with the major class. For example, two minor classes for the shot_registry major class are "check in" and "check out"; two minor classes for shot_command are "start shot" and "hold shot". Other class variables can be used to indicate further subfunctions. With the message-type hierarchy, we can independently define message types for the various parts of the software system.

The programmer who writes the message-handling interface for the DBSERVER does not need to worry about these messages conflicting with those of the programmer who writes the message-handling interface for SHOTSYNC.

Portability of IPCS

The IPCS software is implemented in C. This language is available on most computers, and it provides the tools we need to write a utility that is similar to an operating system. Also, C permits us to transfer IPCS software to various workstations, most of which run UNIX.

The only indispensible operating system functions required to implement IPCS are common memory areas, locks (to provide exclusive access to data structures), and the ability to awaken another process. Most multi-tasking operating systems provide these functions in some form. The code that interfaces with the operating system has been isolated to a few software modules to aid in portability.

The VMS version of IPCS currently supports two network protocols: DECNET, the standard DEC network; and TCP/IP using Wollongong's software package. The UNIX version of IPCS supports the TCP/IP protocol. The IPCS has also been installed on Sun workstations.

References

[1] K.I. Thomassen, "Millimeter Wave Tokamak Heating and Current Drive with a High Power Free Electron Laser," presented at 14th Conf. Plasma Physics, St. Andrews, Scotland, July 1-3, 1987.

[2] T.W. Fredian, J.A. Stillerman, "MDS/MI High Speed Data Acquisition and Analysis Software System," Rev. Sci. Instrum. 57, 1907 (1986).

[3] T.A. Casper, H. Bell, M. Brown, M. Gorvad, S. Jenkins, W. Meyer, J. Moller, D. Perkins, "The TMX-U Computer System in Evolution," 6th Top. Conf. High Temperature Plasma Diagnostics, Hilton Head Island, SC, March 9-13, 1986.

[4] G.G. Preckshot, D.N. Butner, M.D. Brown, W.H. Meyer, "Transparent Data Access in a Multi-vendor, Distributed Data Acquisition and Data Processing System," 7th Top. Conf. High Temperature Plasma Diagnostics, Napa, CA, March 13-17, 1988.

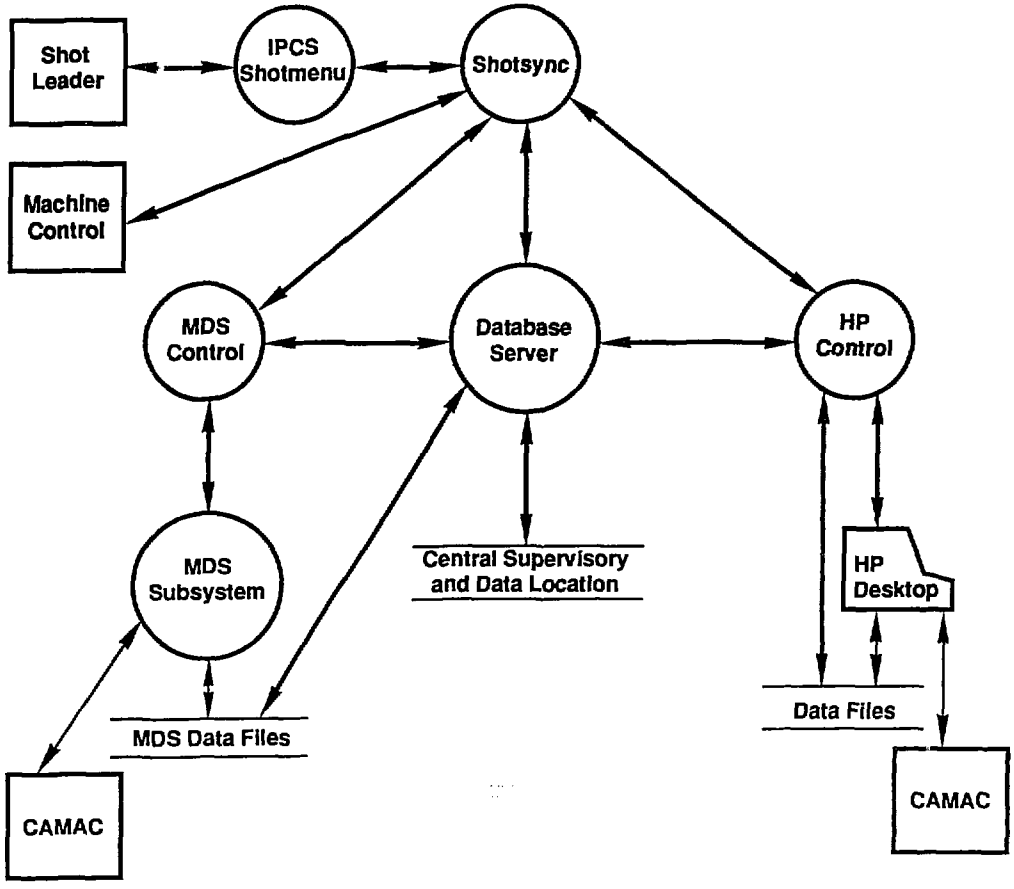
[5] G.G. Preckshot, D.N. Butner, "A Simple and Efficient Interprocess Communication System for Actually Using a Laboratory Computer Network," IEEE Trans. Nuclear Science NS-34, 858 (1987).

Figure Captions

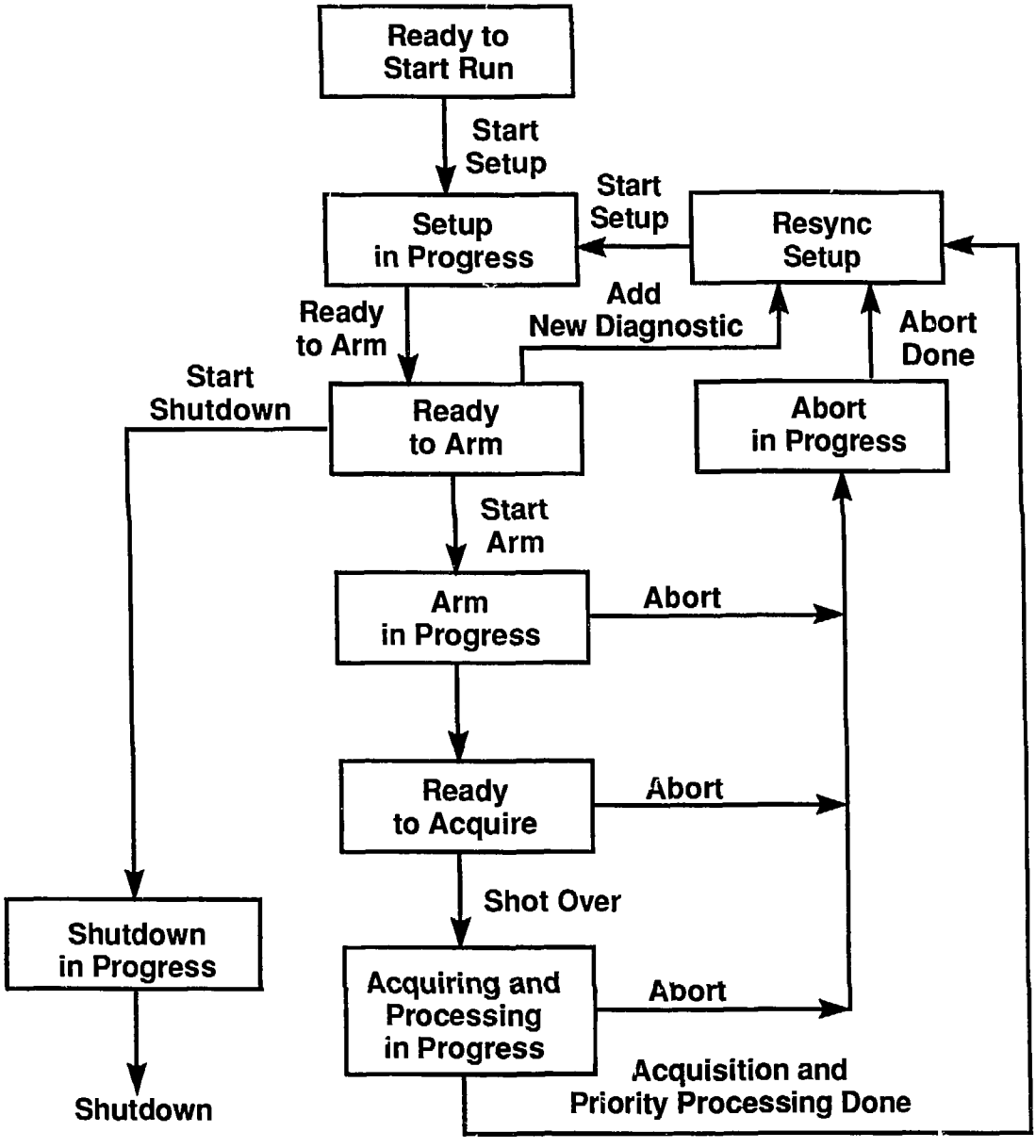
Figure 1. The shot supervisory program, SHOTSYNC, communicates with the diagnostic control programs MDSCONTROL and HPCONTROL. MDSCONTROL and HPCONTROL communicate with DBSERVER, a database program, and each coordinates its respective subsystem.

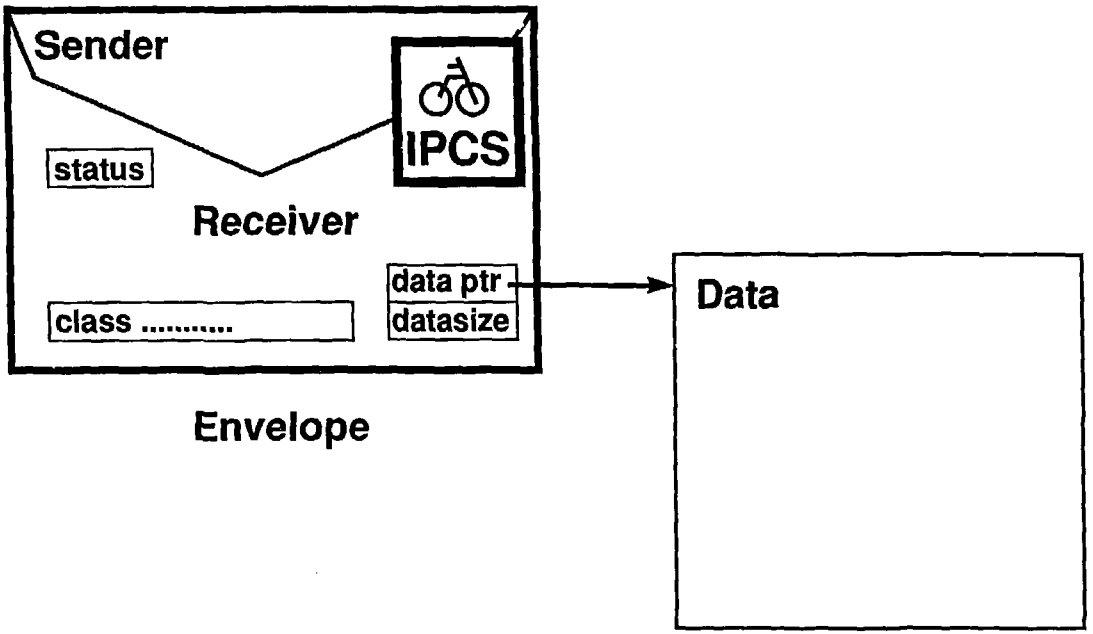
Figure 2. The shot model state diagram.

Figure 3. Conceptual model of an IPCS message.



FL-004-U-5868-01


Ready to Start Run
Start Setup
Setup in Progress
Start Setup
Resync Setup
Ready to Arm
Add New Diagnostic
Abort Done
Start Shutdown
Ready to Arm
Abort in Progress
Start Arm
Arm in Progress
Abort
Ready to Acquire
Abort
Shot Over
Shutdown in Progress
Acquiring and Processing in Progress
Abort
Shutdown
Acquisition and Priority Processing Done


Sender
IPCS
status
Receiver
data ptr
datasize
class .....
Envelope
Data