# The ESRF Control System; Status and Highlights

W.-D.Klotz
European Synchrotron Radiation Facility
BP 220
38043 Grenoble Cedex, France

## 1 Introduction

The European Synchrotron Radiation Facility[1][1] will operate a $6GeV$ $e^-/e^+$ storage ring of 850m circumference to deliver to date unprecedented high brilliance X-rays to the European research community. The ESRF is the first member of a *new generation* of Synchrotron Radiation Sources, in which the *brilliance* of the beam and the utilization of *insertion devices* are pushed to their present limits.

Commissioning of the facility's storage ring will start in spring 1992. A full energy injector, consisting of a 200MeV linear preinjector and a $6GeV$ fast cycling synchrotron ($10Hz$) of 350m circumference have been successfully commissioned during the last months.

The machine control system for this facility, which is under construction since 1988, is still under development, but its initial on-site operation this year has clearly made easier the commissioning of the preinjector plant.

A description of the current system is given and application software for start-up is briefly described.

## 2 Architecture

The ESRF control system is based on a multi-level architecture of distributed hard- and software processing units[2]. Logically the system is structured into four levels. From top to bottom we call them:

- Console Level (Presentation);
- Process Level (Applications);
- Group Level (Device Servers);
- Field Level (Equipments, Embedded Controlers).

On the lowest level, all equipments are interfaced; either by intelligent controlers, as they are delivered from the manufacturer, or by dumb interfaces. Equipments are logically grouped together on the group level. Grouping of equipments is done by similar functionality. The group level is responsible for hardware specific and real time I/O-operations. *Device servers* perform the task of hiding hardware specifics to the upper level. The process level represents that level of the control system where practically all higher level control tasks take place and where physics applications are processed. Powerful multitasking capabilities and fast processing is mandatory on this level. The presentation level presents the interface between the operators and the system. Within this level data entering from the lower level are presented graphically or are formatted to readable reports. Commands entered by means of interactive devices are decoded into events and finally passed as internal messages to the lower levels.

Physically the system is split into 2 levels. All nodes of the presentation and process level consist of UNIX based workstations and file/compute servers interconnected by Ethernet. The group level nodes are realised by VMEbus crates, equipped with 68030 CPU boards. These systems run the OS9 multitasking real time kernel/operating system. Every process level server connects to a private Ethernet segment onto which group level nodes it is in charge of are connected.

The physical boder line between group level and field level is fuzzy. In our system some dumb devices are directly interfaced to VME I/O-boards that are plugged into group level crates, but most dumb devices are interfaced by means of G64 crates. Groups of G64 crates, that interface classes of similar devices, are connected to multidrop highways that are mastered by group level crates. This multidrop highway[2] was developed at ESRF. However, intelligent devices with embedded controlers are, in the majority of cases, directly connected to VME (group level) crates[3].

## 3 Networks

Modern control systems are distributed[3, 4]. The larger the accelerator is, the more important is its network infrastructure. The ESRF control system is fully distributed and relies strongly on a high speed computer network.

Figure 1 gives an overview of the logical and physical implementation of the control system and its network.

Apart from the home-made multidrop highway all computer connections are based on the Ethernet(IEEE 802.3)

[1](ESRF)

[2]we named it FBUS

[3]in the majority by RS422 or RS232 asynchronous serial links

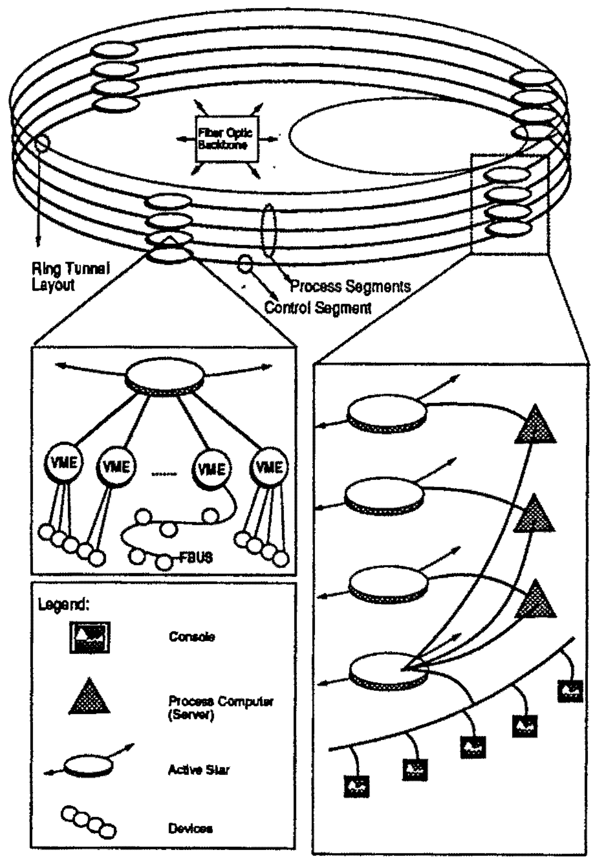

Figure 1: Overall Layout of Control System and Networks

LAN standard and the TCP/IP protocol suite. The network is constructed using 50/125$\mu m$ multimode graded index optic fibres[4] for cabling, that will allow a later migration to FDDI.

Four networking centers are located around the storage ring tunnel, a center comprising a "NODE" and a wiring "HUB". A NODE is the location that acts as the convergent point for IEEE 802.3 compliant active devices e.g. an *active star*. To the NODE are attached "LOBES" which are configured on a star wired topology. A LOBE is the remote system connected to a NODE, and in this case it is a remote "transceiver" attached by a standard AUI[5] cable to a VME crate, process computer, graphics workstation or fanout unit.

For the active stars at the NODES, the "Lannet Multinet II" system was chosen for its ability to operate in a single chassis and support up to four independent backbone bus's. In addition all backbone fibre optic links are run in a *synchronous* Ethernet mode. By using this mode it is possible to install more than the "four" repeaters that restricts standard asynchronous Ethernet systems, with the restraining factor being, the round trip delay that limits each Ethernet segment to approx. 4.5 kilometers.

The wiring HUB is the central point for passive network components, i.e. the *backbone optical fibres* that link all the HUBs together in a circular structure around the storage ring tunnel. It also acts as the termination point for all star wired fibres that are attached to the LOBES.

When installing the circular *backbone*, provision for 10 independent rings has been made, out of which four will be used initially. One ring functions as the main *control segment*, to which all upper layer processors are connected. There are three dedicated *process servers* that operate as network gateways to the other three rings. The latter are used as *process segments* to which all middle layer VME systems are connected. Devices in the field are either connected directly to them or accessed through the multidrop highway.

Response times on a heavily loaded Ethernet become quasi stochastic. Having this in mind, the whole system was designed in a way to provide maximum flexibility in distributing sinks and sources of network traffic. This is accomplished by fiber optic patch panels situated by the HUBs. By simply crosspatching between panels, any processor can immediately be connected to any of the independent rings comprising the backbone. Upgrading the backbone to FDDI would be another, although much more elaborate, remedy against network congestion.

## 4 Computers

### 4.1 Consoles

The standard console in the control room will be an HP Apollo 9000 Model 720 workstation with local disk to hold the bootable image of the operating system and to provide local swap space. File systems containing control system and physics applications software are remotely mounted[6] from the process servers. Currently we are running 5 consoles in the main control room.

In addition to that, RISC based X Window graphics terminals, like the HP 700/X familiy, that are served by the process servers, will be used as console devices. Their ideal usage will be that of "remote" consoles. They will be outside of the control room but plugging to the backbone's control segment. Their main purpose is to give scientists, working on an experiment, the possibility to control the movements of *their* "insertion device" by themselves.

### 4.2 Process Servers

As process servers the HP 9000 Series 800 Model 842 and 857 midrange super-mini computers were selected. There are currently three of these machines in the system. System features are:

- high reliability;

[4]Bandwidth specified at 850$nm$ > 500$MHz * km$ and at 1300$nm$ > 700$MHz * km$

[5]Attachment Unit Interface

[6]currently we are using SUN's NFS that we hope to replace by OSF's DCE

- 29 Mips; 6.9 DP[7] Mflops; CMOS RISC technology;
- internal disk storage: <2.68 Gbytes;
- I/O bandwidth 21 Mbytes/sec;
- integrated DAT[8] unit: 1.3 Gbytes capacity;

These machines are configured as network *gateways* between the control segment and one of the corresponding process segments. To accomplish this, each of them is equipped with two high speed LAN adapters.

The process servers and the console workstations run HP-UX; an AT&T System V Rel 3.0, and BSD 4.3 compliant implementation of the UNIX operating system. All machines run MIT's X Window System, which allows applications to run as X-*clients* on the process servers and to perform interactive I/O through X-*servers* running on the consoles.

### 4.3 VME Systems

All VME systems have an identical base configuration. This comprises a CPU with Motorola 68030 @ 20 MHz, 4 Mbyte RAM, on board Ethernet adapter, and additional 512Kbyte battery backed up RAM on a separate board.

On these systems we run Microware's OS9 realtime kernel/operating system. In addition the systems are equipped with a TCP/IP Internet Support Package providing Berkely sockets, and SUN's Network File System.

All systems are running in a *diskless* configuration for ease of maintenance and reliability. The battery backed 512Kbyte RAM holds the OS9 real time kernel and a minimal TCP/IP configuration. When the system boots, it loads additional OS9 modules, applications, and the NFS modules from a process server. Using NFS, it then copies configuration files from the process server into its RAM disc and initialises the applications.[9] Cold start-up still has to be done manually by toggling a switch on the board's front panel, but we are working on a remote facility.

## 5 Interfacing

Many of the VME systems drive fast multidrop highways. This FBUS is not a general purpose network but implements a low cost remote input/ouput facility. It relies on a master-slave relationship, where a controller (VME based module) drives a large number[10] of slave nodes. The nodes comply with the G64 standard, so that full advantage can be taken of existing interface boards from industry. The FBUS multidrop highway is based on a synchronous serial protocol. Physical implementation uses an extremly noise resistant Manchester encoding with transformer isolation, i.e. each node being galvanically isolated from the highway. The data rate is up to 2 Mbits per second on a 200m-300m long highway. FBUS can still be safely operated at a speed of 1 Mbits per second on distances of up to 1 km and 30 nodes without repeater. The transmission medium is a flexible shielded twisted-pair cable with a characteristic impedance of 78 Ohms.

G64 and FBUS interfacing has been used for control of main magnet power supplies, beam position monitors, magnet interlocks, corrector magnet power supplies, and injection/extraction elements. Other significant subsystems that include G64 crates are the system to distribute the slow timing pulses, the video cross point switch, and the video multiplexors for fluorescent screen monitors. The rest of devices is directly interfaced to the VME systems; either by asynchronous serial lines or digital I/Os.

As a result of an early taken policy, to stick to industry standards, *only a few* boards had to be designed by the ESRF Digital Electronics team:

- video multiplexor[11] (VME)
- delay unit[12] (VME)
- ADC with on-board memory (G64)
- FBUS master (VME and PC/ATbus)
- FBUS slave (G64)
- clock divider[13]

Unavoidably some other dedicated electronics had to be designed to adapt some exotic devices to the standards chosen.

Table 1 gives an overview of interfacing.

## 6 Software

### 6.1 Equipment Access

The low level system software for the distributed control system is based on a "Client/Server" architecture. The Client/Server technology is a simple mechanism to distribute software tasks across any number of processors. This approach is *open and object oriented*, can be implemented on existing systems (eg. OS9 and UNIX), and will be discussed in detail by a contribution to this conference from A.Götz.

Objects are sets of *hidden data* on which well *defined operations* may be performed by authorized users. Associated which each object is a "Server" process that manages the object and exports its functionality as a service. This server-based model is implemented using "Remote Procedure Calls"[14][5, 6]. When a user process wants to perform an operation on an object, it sends a request message to the Server in charge of it. The message contains access keys, a specification of the operation to be performed, and any parameters the operation requires. The user process,

[7] With floating-point processor, Double Precision
[8] Digital Audio Tape
[9] Microware has recently started to ship the BOOTP Port Packs for OS9. BOOTP is used to make a boot PROM with the possibility to boot OS9 directly over Ethernet, thus avoiding the battery backed up RAM
[10] up to 64 on one highway
[11] 15:1
[12] 6 channels, 32MHz resolution, 0-524 ms range
[13] for RF-synchronous triggers, 352MHz:32MHz
[14] RPC; in our implementation we use SUN's RPC and XDR

| VME crate | VME dig I/O | VME serial I/O | VME special I/O | G64 crate | G64 analog in | G64 analog out | Where |
|---|---|---|---|---|---|---|---|
| 1 | 20 | 8 | 6 | 15 | 84 | 84 | Transfer Line 1 |
| 1 | 48 | | 6 | 4 | 3 | 3 | Transfer Line 2 |
| 1 | | 6 | 2 | 3 | | | Injection/Extraction |
| 1 | | | 16 | 5 | 75 | | Synchr. Diagnostic |
| 2 | | 3 | 15 | 9 | | | Synchr. Magnet Power Supplies |
| 3 | | 56 | | | | | Synchr. Vacuum |
| 1 | | 34 | 36 | 8 | | | SY/SR slow timing |
| 1 | | | 37 | 33 | 224 | | Storage Ring Diagnostics |
| 1 | | 21 | 1 | 36 | 36 | 36 | Storage Ring Magnet Power Supplies |
| 16 | | 448 | 34 | 32 | 2050 | | Storage Ring Vacuum |
| 1 | | 1 | 5 | 51 | 576 | | Geodesy & Mag. Interlocks |
| 14 | | 48 | 14 | 9 | | 9 | Insertion Devices |
| | | | 68 | 17 | 816 | | Front Ends |
| | | 41 | 14 | | 82 | | X-ray BPMs |
| 2 | 34 | 53 | | | | | Radiation & Safety |
| 2 | | | | 2 | | | misc. Monitoring |
| 4 | | | | | | | RF & LINAC (subcontr.) |
| 51 | 102 | 719 | 254 | 224 | 3946 | 132 | TOTAL |

Table 1: Overview of Hardware Interfacing for the ESRF accelerator plant

known as the "Client", then blocks. After the Server has performed the operation, it sends back a reply message that unblocks the Client. The combination of sending a request message, blocking, and accepting a reply message forms a Remote Procedure Call, which can be encapsulated to make the entire remote operation look like a local procedure call.

The lowest level of objects in the accelerator plant are the actors and sensors (or physical devices). These objects are "terminator objects"; they are easy to identify, and their behaviour can be modelled and documented.

"Device Servers" are terminator objects that operate on physical devices. The more general term "Server" or "Virtual Device Server" covers objects on a higher level of abstraction. Objects on a higher level of abstraction can use terminator objects to offer a given service[15].

The Device Server is an intermediary between application programs and the physical resources of the accelerator system. It contains all device-specific code, and insulates applications from differences between hardware. It performs the following tasks:

- Allows access to the device by multiple clients. To implement security, the server, depending on its state, may deny access from certain clients.

- Interprets network messages from clients and acts on them. Messages are generated by clients through RPCs.

- Maintains complex data structures, including device state information. Server maintained information reduce the amount of data that has to be maintained by each client and the amount of data that has to be passed over the network.

At ESRF the Device Servers follow the OOP[16] paradigm and have to be implemented in a certain, fixed style. The OOP paradigm is based on the "widget" model from the X11 Intrinsics Toolkit[7] of MIT and is implemented in ANSI C.

The control system designers have decided to implement *all* important functions necessary to run the distributed system in Servers. This includes the processes to boot and manage the system, to access the database, to handle graphics objects, as well as Device Servers to access equipments.

According to our current state of knowledge about 53 Device Servers have to be written for the complete system. About 50% of them are currently released. The average size of a Device Server ranges typically between 2000–2500 lines of C code.

[15] The encapsulation of lower level services into higher level services can continue until a very high abstraction like physical machine parameters, i.e. energy, chromaticity, tune, emittance,..., is achieved.

[16] Object oriented programming

## 6.2 Graphics User Interface

This field of technology is in a state of tremendous innovations. Fortunately some standards exist now: **X11**, and **Motif.** The X11-window system, or X11, is a network-transparent window system. With X11, multiple applications can run simultaneously in windows, generating text and graphics. Network transparency means that application programs that are running on other machines scattered throughout the network, can be used as if they were running on a local machine.

The core components of the OSF/Motif technology include an extensible user interface[17], an applications programming interface[18], a user interface metalanguage[19], and a window manager. Motif is based on the X Intrinsics, a toolkit framework provided with X11. The Intrinsics use an object oriented model to create a class hierarchy of graphical objects known as "*widgets*".

Both X11 and Motif, are extremely helpful but their libraries are complex to learn and to use for programming. Coding of applications started initially with those libraries, and demanded substantial efforts in becoming familiar with this new technology.

User Interface Management Systems[20] (sometimes called interface builders) are the tools which help the application programmer to design the user interface part of the application interactively. A UIMS is generally composed of a graphic oriented editor and a code generator, and sometimes other complementary tools. There are several Motif compliant UIMS available. A few of them have been tested at ESRF. At present one of them has been selected for use[8].

This UIMS drastically eases now the design of Motif-based user interfaces. It generates stand-alone C code and/or a combination of Motif-compliant C and UIL code.

Synoptic drawings with selectable objects are scarcely supported by the above mentioned tools. We therefore work on an implementation of a Motif compliant widget that uses vectorial drawings generated by PHIGS[21]. Synoptics will be generated by CAE systems like AUTOCAD or EUCLID.

## 6.3 Database

The control system data are stored in relational databases which manage two logical parts:

- Resource data;
- Runtime data.

The resource database keeps permanent data. Examples are: start-up resources, calibrations, equipment definitions, installation- & maintenance data, etc...

[17] UI
[18] API
[19] UIL
[20] UIMS
[21] stands for Programmer's Hierarchical Graphics System and is an ISO standard

The implementation of the resource database uses ORACLE and its powerful set of development tools. Modifications on the resource database cause automatic update of runtime data sets, that are redundant copies of the *parent data set* in the resource database.

The runtime database is a central warehouse for all sorts of temporary or transient data. It is not a medium for permanent storage. It is simply a front for permanent storage. Only *memory resident* database systems can meet the demands for sufficiently short transaction times. A prototype of the runtime database is operational and uses a *Real-Time Database Base Management System*[22] available on HPs.

The runtime database can be used to alleviate congestion problems. Multiple processes can update data asynchronously in the database. Other processes can retrieve this information aynchronously without blocking the process doing the updating. Since the memory resident database profits from a much higher than normal I/O throughput, this mechanism is used to resolve information traffic jams that may occur.

The runtime database's prime source is a so-called *Update Daemon* that updates the current machine status if a particular client requests this. The database contains ring buffered tables to store brief histories of the results of requests issued to a device. Although physically the runtime database is distributed over all process servers, access to it is transparent to database clients. On-line data can be archived continously. Only a time window of some five minutes is kept in memory by RTDB, the rest of the data is dumped into the disk-based ORACLE database. An index to these data is constructed to allow *queries* in accelerator physics terms on archived data. Data can be stamped with time or accelerator status information.

The same runtime database can also be used by applications as a mean for interprocess communication. Applications dynamically allocate "tables" of formatted data, that can then be piped or multiplexed to other applications.

The volume of data that will be managed by the resource database is estimated to be some 10Mbytes. The whole control system comprises more than 3000 devices and more than 50000 static resources. The throughput of on-line data at its worst is estimated to be some 40kbytes/sec. If all data coming from the machine is stored at an interval of a second, it would mean 12Mbytes every 5 minutes.

## 6.4 Applications

Application program development at ESRF has been taken care of at an early stage of the project. Usually this class of software tends to be too little and too late. To give accelerator physicists the possibility to develop their applications in parallel with the control system software, an applications programmer interface[23] has been defined very early and kept stable until then.

[22] called RTDB
[23] API

| | NFS/RPC | | API |
|---|---|---|---|
| | UDP | TCP | UDP |
| open connection | 20-25ms | 35-45ms | 55-65ms |
| close connection | 0.1-0.2ms | 0.3-0.5ms | 10-20ms |
| RPC with 100bytes | 10-15ms | 15-20ms | 15-20ms |
| RPC with 8kbytes | 25-35ms | 55-65ms | 30-40ms |
| RPC with 40kbytes | | 220-250ms | |

Table 2: Performance figures for RPC and API

Access to the Device Servers is provided by a small set of C calls. These calls allow the users to develop their applications in peace without being affected by what goes on in the Device Server software. Initially very simple Device Servers have been written, that ran on the local host, and that only *simulated* devices. These API-calls hide the complexity of Device Servers and their implementation from users by offering them a set of high level commands as access method. *How* and *where* the Device Server executes the high level command is hidden. In the distributed environment this workload is spread over a number of machines.

The following functions form the basis of the Device Server API:

-
  ```
  int dev_import(name, access, ds_ptr, error)
    char *name;          /* Device Server Name */
    long int access;     /* Access Type */
    devserver *ds_ptr;   /* DevServer Handle */
    long *error;         /* Error Code */
  ```
  Is called by the application to establish a connection to a Device of the specified name.

-
  ```
  int dev_putget(ds, cmd, argin_ptr, typein
                 argout_ptr, typeout, error)
    devserver ds;           /* DevServer Handle */
    DevCmd cmd;             /* Device Command */
    DevArgPtr argin_ptr;    /* Call Parameter */
    DevType typein;         /* Parameter Type */
    DevArgPtr argout_ptr;/* Return Parameter */
    DevType typeout;        /* Parameter Type */
    long *error;
  ```
  Is called by the application to execute a command on the device. This is a "blocking" call which doesn't return until the command requested has been executed.

-
  ```
  int dev_put(ds, cmd, argin_ptr, typein,
                                  error)
  ```
  Is called by the application to execute a command on the device. This is an "asynchronous" call which will return as soon as the command has been delivered to the server or an error occurred. This call can only be used to start a command, no knowledge is returned about its execution and/or success. It is up to the application to interrogate the Device Server to determine its status.

-
  ```
  int dev_free(ds, error)
    DevServer ds;    /* DevServer Handle */
    long *error;
  ```
  Is called by an application to release a device properly.

Measurements of RPC and API performance that we achieve between a process server and a VME node are given in table 2.

Using strictly this device access interface and the X11 and Motif standards for interactive graphical I/O, an impressive number of physics applications have been developed in parallel with the basic control system software, and have considerably helped to commission the booster in due time. An enumeration of applications presently available follows:

**Transfer line 1 & 2:** These programs execute specific procedures for step by step alignment, emittance measurement, and modelling of beam envelopes.

**Closed orbit:** The program performs basic control of steerers and bumps, beam position readout, orbit plots, fourier analysis of orbit and steerers, and automatic orbit correction.

**Booster vacuum:** This program controls the vacuum system. It allows individual device control, display of periodically updated status, and display of pressure profile.

**Booster injection/extraction:** This program allows control of current- and timing settings of pulsed injection/extraction elements.

**Booster optics:** Different options in this application allow tune measurement/setting, chromaticity measurement/setting, measurement of $\beta$-functions at quadrupole locations, and measurement of dispersion ($\eta$-function).

**Storage ring injection:** Used for tuning of the injection kicker/septa to maintain an injection bump and control injected beam position and angle.

## 7 Conclusion

The ESRF control system is operational since August 1991. It played an important role during commissioning of the booster synchrotron. The system has been designed from bottom up, using object oriented programming techniques, and is based on proven industry standards. Its design has been guided by a clear preference for mature commercial systems over custom- or home-made ones, without formally excluding the latter.

The system is not finished yet, but it is easily extendable and adaptable to future needs. It is through the standards that have been selected for the control system, that ESRF will be able to migrate together with industry to new technologies while preserving considerable investments in hard- and software. The choices of UNIX, X11/Motif, VME/OS9, Ethernet, and TCP/IP have been fundamental in this sense.

## 8 Aknowledgements

The author is grateful to all contributing members of the ESRF controls group; in particular to C.Hervé, head of the Digital Design Group, and A.Götz, head of the Control Software Group, for their competent and efficient leadership of the project. He takes the opportunity to thank also A.Ropert, L.Farvacque, and all members of the machine division Theory Group, for their ongoing engagement in the production of control- and physics applications, and the smooth collaboration with them. Thanks also to J.Altaber, and F.Perriollat (both CERN), as well as collaborating groups from CERN, Trieste, and Jülich, who initially helped us to get the project going.

## 9 References

## References

[1] J.L. Laclare, "Overview of the European Synchrotron Light Source" in *IEEE Particle Accelerator Conference*, Washington, D.C., March 1987, pp. 417-421.

[2] W.D. Klotz and C.Hervé, "The Conceptual design of the ESRF Control System" in *European Particle Accelerator Conference*, Rome, June 1988, pp. 1196-1198.

[3] *Proceedings of Europhysics Conference on Control Systems for Experimental Physics*, Villars-sur-Ollon, Switzerland, Sept.28 - Oct.2, 1987.

[4] *Proceedings of the International Conference on Accelerator and Large Physics Control Systems*, Vancouver, BC, October 30 - November 3, 1989.

[5] Pal S. Andersen, V. Frammery, and R. Wilcke, "Tools for Remote Computing in Accelerator control" in *Proceedings of the International Conference on Accelerator and Large Physics Control Systems*, Vancouver, BC, October 30 - November 3, 1989, pp. 225-230.

[6] *Network Programming Guide*, SUN microsystems, Part Number: 800-3850-10, Revision A of 27 March, 1990.

[7] A. Nye and T. O'Reilly, *X Toolkit Intrinsics Programming Manual*, O'Reilly & Associates, Inc., 1990.

[8] *The Builder Xcessory User Guide*, Cambridge, MA: Integrated Computer Solutions, Inc. 1990

## Trademarks