Implementation of PCs in the HERA Control System

P. Duval (DESY-MKI, 22607 Hamburg FRG; duval@pktr.desy.de)

Abstract
Progress in controlling the HERA machine at DESY with PCs is described, with emphasis on the use of producer-consumer communication links. To date the quench protection system for the superconducting proton ring, the proton collimators, and major portions of the beam diagnostic control and RF controls are incorporated in a control system using PCs running MS-DOS and WINDOWS-based software in a NOVELL network environment. Although dominated by PCs communicating with IPX protocols, the system also supports IP-based communications for links to UNIX machines and VME CPUs. HERA Magnet control is still performed by NORD minicomputers, but the coming upgrade is also discussed in the context of the PC magnet control system planned for the HERA pre-accelerators and peripheral machines, which is now being tested in the PETRA and DORIS rings.

Introduction

The HERA (Hadron-Elektron Ring Anlage) accelerator at DESY consists of a 6.3 km electron storage ring (30 GeV) straddling a 6.3 km superconducting proton storage ring (820 GeV). Prior to injection into HERA, electrons and protons undergo similar sequences of acceleration starting with LINACs, passing on to small synchrotrons, and the PETRA ring. The beams are then accelerated in the HERA rings, stored at full energy, and brought into collision. The design luminosity of 1.6E31 requires 210 bunches in each ring. Collisions are observed by the H1 and ZEUS experiments, each consisting of a large superconducting solenoid surrounding the beam pipe at one of the interaction regions, and associated detectors. The electron beam can also be polarized and used to study nucleon spin structure, while the protons diffusing out of the beam are used to produce (among other things) B mesons to study CP violation.

Since the smaller accelerators and the experiments have their own control systems, we can speak of the HERA control system as referring only to the HERA machine. However, we should keep in mind that each system needs vital information from the others, so there has to be a mechanism for fast data exchange. Of necessity, each sub-system originally followed its own frenzied development during commissioning, and each has its own view of the “Standard Model,” so establishing such a data-exchange mechanism is in some cases no small task.

In this report we shall focus primarily on the HERA machine, but nonetheless present a few details concerning data-exchange among the sub-systems.

The HERA Machine

Before we begin discussing the relative merits of PCs versus something else, let us first take a quick look at what we have to control. Beam storage and steering is achieved using magnets controlled by approximately 1200 independently powered circuits in the two rings. Setting and reading these magnet currents is the primary task of the control system. Acceleration is achieved via coordinated ramping of the RF system frequency and voltage (consisting of 6 cavities and transmitters for the proton ring and over 80 for the electron ring) and the magnet currents. Backgrounds are controlled by scraping off the beam halos with 3 proton and 12 electron collimators. There are independent proton and electron vacuum controls. As the proton magnets are superconducting, there is a separate cryogenics control for these elements, and an associated quench protection system. Monitoring the beam is of prime importance and to this end there are ~300 proton and ~300 electron Beam Position Monitors (BPMs), and approximately the same number of proton and electron Beam Loss Monitors (BLMs) and associated Alarm modules. As the proton BPMs have a external trigger, there is also a separate HERA Integrated Timing system with corresponding trigger and delay modules for each BPM. There are several integrating beam current monitors for both electrons and protons, as well as fast single and multi-bunch current monitors. Likewise, there are several beam profile monitors, including wire scanners and residual gas monitors. The proton and electron tunes are also monitored, as are various state parameters of the machine in general, such as tunnel temperature.

From the preceding paragraph, a dichotomy of controllable objects is clearly seen, namely those objects related to diagnostics and those which actually control the beam. In the case of diagnostics, under most circumstances devices of a given type, say BPMs, can be read, controlled and coordinated from a single CPU for the entire ring. Furthermore, if such a CPU needs to be rebooted (for whatever reason), it does not affect the machine to any great extent. This is not necessarily true for a machine control object such as a magnet. It might not be possible to control all such objects from a single CPU, necessitating distributed control in some form or another, and if one of the controlling CPUs needs to be rebooted it could easily lead to loss of control and in the worst case a beam abort.

PCs in HERA

1. What is a PC?

A relevant question these days is: When I say PC what do I mean? Do I insist on a hardware definition centered around the INTEL CPU and inexpensively priced end-units, or do I stick with an operating system definition centered around Microsoft, or both? Although, when people run LINUX on a PENTIUM they still refer to the machine as a PC, and when the same people run NT on an ALPHA they don't, in this report we'll generally abide by the third (i.e. both) definition, and go a step further and state that a HERA PC is an INTEL machine which runs either MS-DOS or MS-WINDOWS 3.1.

2. Why PCs?

Looking at hardware, arguments concerning costs are compelling. Granted, an extra high-end PC may come in around the same price as a run-of-the-mill SPARC, but one seldom needs a high-end PC. In fact, since old "forgotten PCs" can frequently do the controls job we're interested in, hardware costs are sometimes negligible. Keeping in mind the age-old adage: "If we just want to go shopping, do we need to drive a Mercedes?" we should always ask ourselves if we are just going shopping.

Looking at software, it's not hard to beat MS-DOS and/or MS-WINDOWS as an operating system. The major shortcoming here is that the OS is not protected, and risks exposure to every vicious C program that tries to run on it. One should reiterate though, that neither MS-DOS nor MS-WINDOWS crashes by itself, and healthy programs will run on it indeefinitely (although development can be agonizing). Anyway, that's the down side. The up side is that Microsoft has around 80 percent of the software market and is in no danger of disappearing anytime soon. There is an enormous culture base built around MS-WINDOWS, and an impressive array of commercial software which already covers most of our needs. The overwhelming market share and user-feedback that Microsoft enjoys has essentially defined what is user friendly and what isn't.

3. When and Where to use PCs:

In truth there might be some times when we really do need a Mercedes. It's therefore important to 1) plan for an integrated control system and 2) use the right tool for the right job. So where is a PC the right tool for the right job?

There is generally no disagreement that a PC can be effectively used at the CONSOLE side of the control system (there might be some grumbling, but there is no major disagreement). This is true since the CONSOLEs are only displaying data and passing commands to the Front Ends. They are not directly steering control system hardware. Since real-time is never an issue here, MS-WINDOWS is fine. Furthermore if a rogue program gets loose on the CONSOLE and MS-WINDOWS crashes, you loose the display, but nothing more serious than that.

When can PCs be used at the Front End? This is a more contentious issue. Even the most ardent VxWorks enthusiasts will nonetheless concede that there are simple control tasks, which don't need Real Time by a long shot, and can be quite trivially controlled by a PC. For example, reading an ADC, which monitors the ambient tunnel temperature every few minutes. Such a task can be realized by a 286 PC running MS-DOS (cost: ~100 $). If the machine crashes (this will be a program bug, MS-DOS doesn't crash spontaneously), you are in no danger of losing the beam. Let's take that a step further, and claim that a good many diagnostic tasks can live quite happily on an MS-DOS machine with (for example) a

GPIB card. Such dedicated diagnostic Front Ends, whose temporary disappearance (although a nuisance) is not fatal, we shall refer to as simple FECs (Front End Computer). Simple FECs are excellent candidates for PCs. By the way, this "temporary disappearance" might occur on any platform. Whether a bug causes the process to "core dump" or the machine to hang is immaterial. You'll still have to have a way of dealing with it.

Beyond Simple FECs are Complex FECs, i.e. Front Ends with multiple tasks, and/or Real Time requirements, and/or crucial importance. At HERA, we have many PC FECs meeting all of these criteria (Note: there are several Real Time Kernels for MS-DOS commercially available). Nonetheless, we prefer to leave the case of Complex FECs an open issue, and assume that we have a multi-platform control system. Indeed, as we shall shortly see, we incorporate VxWorks FECs, and FECs running as servers on a variety of UNIX and VMS machines.

4. Networked PCs

Following the "Standard Model", the CONSOLEs and FECs at HERA reside on an ETHERNET. In addition, the PC FECs and CONSOLEs are attached to a NOVELL file server. Furthermore, all control system code is located on the file server (and not on the PC!), meaning that at boot time a PC logs in and mounts network drives to the file server, and then loads and runs its code. In the case of a CONSOLE or an FEC running WINDOWS (most FECs run DOS), WINDOWS itself is also loaded from the file server. As WINDOWS generally has several open files and does considerable swapping, this does introduce a weakness in that the connection to the file server is of paramount importance, since if the connection is lost the workstation is almost guaranteed to crash. On the other hand, all backup strategies can be entirely focused on the file server.   The weakness can be (and will be this shutdown) patched by mirroring all relevant software locally at login time, and always running from a local operating system.

Control System Model

So far, what we have described follows the so-called "Standard Model" of a control system, in that hardware-near FECs communicate with user-near CONSOLEs over the ETHERNET. We have not insisted, however, that FECs are VME CPUs or that anyone is running UNIX. On the contrary, we strive for a multi-platform control system, where CONSOLEs and FECs communicate with each other without concern as to what operating system is running at the other end. With that in mind, we make the following ansatz:

∑ An FEC should present an integrated device to potential clients, i.e. a client should see an object representation of the device to which it is speaking, be it a magnet, oscilloscope, BPM or whatever. A client should never deal with raw data, but instead receive data ready for display, and issue commands via mnemonic properties.

This goes hand in hand with saying that any change of state made by a CONSOLE should be seen by all other CONSOLEs.

Also important is that there should be no distributed control over the ETHERNET, requiring one FEC to be crucially dependent upon knowing the state of another FEC. To be sure, an FEC can be (and frequently is) a client of another FEC. However, the possibility that information might not come in on time or might not come in at all should be anticipated.

It remains to discuss specific communication models CONSOLE to FEC, and at this juncture we should mention our indebtedness to the ISOLDE project at CERN/PS and its developers. The ISOLDE concept was our starting point in 1991, and in some respects our system still bears a strong resemblance to ISOLDE although our needs and development began to diverge rapidly from the original concept in late 1992. Initially the communication model was pure client-server, in that a CONSOLE always asked an FEC synchronously for data. Polled data were requested from a timer on the CONSOLE. Also to be noted is that the original concept was a PC-only one and used only IPX-based protocols. In 1993, IP-based protocols were included, opening the door to non-PC communication partners. The fundamental model remained, however, "client-server". FECs were servers, and did not supply data unless specifically asked to.

Both IPX and IP communication channels were based on socket libraries, and specifically not commercial RPC products. This was largely owing to the fact that such products did not encompass both the IPX and IP worlds at the same time, whereas (working with sockets) the application layer and network/transport layers were easy to keep separate, facilitating development.

It soon became clear that certain data channels were of vital interest to a large number of CONSOLEs and FECs. These were items such as the energy and current of the proton and electron beams. And rather than having N clients request the present values of these parameters from one server (where N is a large number!) it was decided to make such values available via broadcast. To be efficient, the value is broadcast upon change (according to an appropriate tolerance) or at the system heartbeat of once per minute. In this way, no one has to ask, one only has to listen. Here, we are slipping over to a "producer-consumer" model of data exchange, where an FEC is seen as a data "producer," who is simply providing data to any "consumer" who is listening. An FEC should be rightfully regarded as a "producer" anyway, in that it should be in background constantly reading hardware and preparing data for display. Requests for data from the client side should in most cases end up fetching data from RAM and not initiating a hardware read.

This pure "producer-consumer" model was only applied to the most important machine parameters, however. Nevertheless, a crucial step in the "producer-consumer" direction was taken in 1994, by adopting a registered consumer model in which data links were now registered at the Front End instead of being polled from the CONSOLE. In other words, an FEC would keep track of a "mailing list" of consumers of pertinent data. A consumer could request values at a specific polling rate, or to be refreshed only upon change. The data would then arrive entirely asynchronously. The reduction in wasted CPU time, not to mention network traffic, in such a model is considerable! It is not uncommon, for instance, to have a popular control application, which might be getting a data update at 1 Hz, run on 15 to 20 different client workstations. Rather than having all 20 stations individually ask a server-FEC for data (one packet to the FEC, plus a function call) and receive data (one packet from the FEC) at 1 Hz (times 20), the producer-FEC keeps track of all 20 clients, makes one and only one function call at 1 Hz, and sends the data out to its mailing list (one packet from the FEC times 20). Furthermore, as the operation is connectionless and asynchronous, there is never any waiting on the part of producer or consumer.

Acknowledgments are required under only one set of circumstances. If a client has requested to be refreshed only upon change of data, and the data have indeed changed, then the producer-FEC asks to be acknowledged upon receipt of data. Otherwise, a consumer-CONSOLE knows very well if its requested polling rate has been met or not, or if the system heartbeat time of one minute has been exceeded. If warranted, it will sound an alarm and make an effort to relocate the FEC. Similarly, when a consumer-CONSOLE registers with an FEC, it subscribes for a certain quantity of data updates. When the subscription runs down, it must renew its subscription, or it will be removed from the FEC's mailing list. In this way, there can be no dangling consumers. Maximum efficiency regarding network traffic is maintained by packing together in the same ETHERNET packet all subscriptions destined for the same consumer at the same time.

Command-based requests of course follow a client-server approach. The client has the choice of sending commands synchronously or asynchronously. As the communication is connectionless, the turn around time for request and reply is typically 2 to 3 milliseconds (IPX is marginally faster than UDP) plus any time the FEC might need to read hardware.

These types of detail are of course hidden from the control system application programmer. What he sees is an API which tells him how to link data from a Front End into his control program, in the case of a CONSOLE application, or how to offer data for linking, in the case of an FEC application. Important is only that the Front End device shows up as a tag name in the API. The tag name is resolved at initialization into an FEC address and an equipment function living on the FEC, both invisible to the API. The FEC address will be either an IPX address, if both partners speak IPX, or an IP address.

At present, name resolution begins with a database file, which matches tag name to FEC name and equipment function name. This will be replaced by a name resolver during the 1996 winter shutdown. An FEC address is then established by the following: If the client is not a PC attached to the control system file server, then the local host table is scanned for the IP address of an entry whose alias matches the FEC name. If the client is a PC attached to the control file server, then the user list for the file server is scanned for a user matching the FEC name. If a match is found then an IPX address is established. If no match is found then the local host table is scanned for an IP entry. All CONSOLEs speak both IPX and IP. All PC FECs speak IPX, and speak IP only if necessary. And all non PC-FECs speak IP. We should also mention that there are non PC-clients to certain control system elements as well. These are principally machines outside the HERA control room, and serve to integrate the various sub-systems of the HERA machine and experiments.

An important element to the control system structure is the concept of the Data Server. This is a machine designed to acquire all important machine parameters at a sufficient rate so as to be able to act as a data gateway to clients outside the immediate control sub-system. It is both a client and a server with a large number of channels, and is in general very busy. This is nonetheless a much more efficient model for N clients to obtain machine data, than for the same N clients to form individual links to potentially many different front ends (largely owing to the data packing mechanism described earlier). The most important subset of machine parameters is broadcast from this machine as described above. Furthermore, as this machine always has an up-to-date record of the machine parameters, it also serves as the control system archiver.

As to archiving, machine state variables are regularly archived (with appropriate filters) allowing data retrieval and correlation throughout the year. Similarly, critical events (such as a magnet quench) can initiate archive dumps of the state of a particular subsystem at the time of the event. This information is of course of vital interest to the systems engineers.

The control system is also designed to be open, in that office client machines also have access to control system data and applications, but with certain restrictions. Namely, the FEC server itself always knows the identity of the caller and can pre-specify a list of users with WRITE access. READ access is generally allowed since data READs do not change the state of the hardware. Likewise, client applications can also choose to hide options from under-privileged users.

HERA: Current Status

So where are the PCs in HERA? As of this writing, CONSOLEs in the control room are either PCs running WINDOWS or NORD mini-computers running SINTRAN, in approximately equal numbers, with the NORDs living on borrowed time. The current generation of PCs in the control room are 486 33 MHz or 50 MHz machines. These will likely be replaced by PENTIUMs in the coming year.

Most components of the proton beam diagnostic controls, as well as some electron diagnostic controls, are incorporated on PC FECs running MS-DOS. A small minority run MS-WINDOWS, in cases where a user-friendly GUI is important at the front end. And there are also a number of non-PC FECs which are a fully integrated part of the HERA control system. These are summarized in Table 1.

As the data exchange mechanism described above has been ported to most platforms seen at DESY, a number of control sub-systems provide data server gateways and/or are clients to the HERA data server. The Proton Vacuum and Cryogenics systems, for instance, both of which are autonomous control systems in themselves, are persistent clients to the HERA data server.

HERA: Next Year

One of the last and most important steps in upgrading the existing HERA control system involves magnet control. As yet the 1200 electron and proton magnet controllers are driven by 4 NORD minicomputers. There are two promising approaches currently being tested at DESY. One involves scaling the all-PC PETRA control system to HERA, and the other involves using Symmetric Multi-Processing (SMP) on a multi-CPU SUN workstation (see Herb and Wu, “Accelerator Magnet Control from an SMP Workstation”, poster session this conference). In any case, the 1200 individual channels necessitate a multi-CPU approach. The SMP solution would control all magnets from one computer whereas the PC solution would span more than one. The PC solution appears to work well in PETRA. The number of magnets there is considerably smaller, and all magnets can be controlled from one PC. In HERA the horizontal and vertical correctors could conceivably be controlled from separate PCs.

Table 1. Front End Computers at HERA following the PKTR Control System Model.

FecName	OS	Description	Hardware
ADDA	DOS	Hera Tune Control	i386, GPIB
BEAMSCOP	DOS	Hera P X-channel/Y-channel Beam Scope Monitor	i386, GPIB
BPM	DOS	Hera P BPMs,BLMs,MTMs,Delay Modules,Alarm Loop Modules	i486, 4x SEDAC, V24
CMFL	DOS	Hera P Fast Current (Lopez) monitor	i386, GPIB
DATASERV	DOS	Hera P Data Server	i486, Watchdog
BEAMCURR	DOS	Hera P and E beam current monitors	i386, GPIB, SEDAC, Watchdog
FECSIM	DOS	Hera P development FEC	i386, Watchdog
HERA208	DOS	Hera RF (208 MHz)	i486, SEDAC
HERA52	DOS	Hera RF (52 MHz)	i486, SEDAC
HERAQ	DOS	Hera P X and Y tunes	i386, 2 DSPs
HIT	DOS	Hera Intgrated Timing	i386, GPIB
PETRA52	DOS	Petra RF (52 MHz)	i486, SEDAC
SCRAPERS	DOS	Hera P Collimator Steering	i386, SEDAC, Watchdog
OSZIS	DOS	Oscillascope Steering	i386, GPIB, V24
IPS104	UNIX	Orbit Correction	HP
MKI101	UNIX	Orbit Correction, Magnet Data	HP
SUN1	UNIX	Development	SUN SPARC
SUN2	UNIX	NORD Gateway	SUN SPARC
EMIT_PP	DOS	Petra P Residual Gas Monitor	i386, SEDAC
BLME	DOS	Hera E BLMs	i486, 4x SEDAC
EMIT_HP	DOS	Hera P Residual Gas Monitor	i386, SEDAC, ADC
ZLUM01	UNIX	Zeus Lumi Workstation	SGI
DORCAV	DOS	Doris Cavity	i486, SEDAC
BUNCHCUR	DOS	Hera-P Bunch Current	i386, GPIB
WIRE	DOS	Hera-B Wire Target	i486, SEDAC
DORQ1	DOS	Doris Q1 Sender	i486, SEDAC
HFPET_R	DOS	Petra Sender	i486, SEDAC
DORQ4	DOS	Doris Q Sender	i486, SEDAC
VWMASTER	UNIX	VxWorks (Development)	VME, CAN
VWSLAVE	UNIX	VxWorks (Development)	VME, CAN
PETRAQ	DOS	Petra Tune	i286, 3 DSPs
PETRAI	DOS	Petra Beam Current	i386, SEDAC
HFPET_L	DOS	Petra Sender	i486, SEDAC
TIC	DOS	Hera Intgrated Timing	i486. GPIB
TUNNEL_O	DOS	Tunnel Temperatures	i386, SEDAC
TUNNEL_W	DOS	Tunnel Temperatures	i386, SEDAC
DORFB	DOS	Doris Q2 Sender	i486, SEDAC
BCURRE	DOS	E Beam Current	i386, GPIB
LPSVAX	VMS	ZEUS Roman Pot Positions	VAX
PROXY	DOS	Middleman FEC: Quench Proxy, Netmex Proxy	i386, Watchdog
BLMDSY	DOS	DESY BLM FEC	i386, SEDAC
DESYGAS	DOS	DESY Gas Monitor	i486. GPIB
TUNEMOD	DOS	Hera Tune Modulator	i486, GPIB
LPS_H1	WINDOWS	H1 Roman Pot Positions	i486, SEDAC
WINFEC	WINDOWS	Windows FEC simulator	i486
DESYWS	DOS	DESY III Wire Scanner	i486, SEDAC
HWEST	DOS	HERA Sender (Development)	i486, SEDAC
HF_HE_SL	DOS	HERA Sender	i486, SEDAC
HF_HE_SR	DOS	HERA Sender	i486, SEDAC
HF_HE_OL	DOS	HERA Sender	i486, SEDAC

Table 1. Front End Computers at HERA ... (continued).

FecName	OS	Description	Hardware
HF_HE_OR	DOS	HERA Sender	i486, SEDAC
HF_HE_NL	DOS	HERA Sender	i486, SEDAC
HF_HE_NR	DOS	HERA Sender	i486, SEDAC
HF_HW_FB	DOS	HERA Sender	i486, SEDAC
MKI102	UNIX	Orbit Corrections	HP
IPS109	UNIX	Orbit Corrections	HP
VWWEST1	VxWorks	SPS Master	VME, CAN
VWNORD1	VxWorks	SPS Master	VME, CAN
VWEAST1	VxWorks	SPS Master	VME, CAN
VWSUED1	VxWorks	SPS Master	VME, CAN
VWWEST0	VxWorks	Transient Recorders	VME, CAN
VWNORD0	VxWorks	CPU 0	VME, CAN
VWEAST0	VxWorks	CPU 0	VME, CAN
VWSUED0	VxWorks	CPU 0	VME, CAN
VWWEST2	VxWorks	SPS Slave	VME, CAN
VWNORD2	VxWorks	SPS Slave	VME, CAN
VWEAST2	VxWorks	SPS Slave	VME, CAN
VWSUED2	VxWorks	SPS Slave	VME, CAN
ZIZI	UNIX	Zeus Lumi Data	SGI
EMIT_HPS	DOS	HERA Gas Monitor	i486, GPIB
DESYSCOP	WINDOWS	DESY Scope Gas Monitor	i486, GPIB

The PETRA model is worth mentioning here. For one thing, it is a much more homogeneous model where all of the players (with very few exceptions) are PCs running MS-WINDOWS. This in itself has merit, as we shall shortly see. Furthermore, the control system ETHERNET segment is kept isolated, with only a gateway interface to the outside. Most of the traffic on that segment is restricted to pure producer-consumer traffic, or rather “producer traffic.” Data are flushed from the front ends onto the net via broadcast at a cycle frequency of 1 or perhaps 2 Hz. The consumers only listen; they don’t introduce any extra traffic. Of course allowing commands from the console necessarily implies that client-server traffic also appears from time to time, but not in sufficient quantity to upset the overall loading.

Which, if either, of these solutions will be adopted remains to be seen. However, we should note that if the SMP approach is adopted, this will add yet another make of 'Mercedes' on the control system. Is this bad, good, or irrelevant?

There is much to be said for keeping a system as homogeneous as possible. The expertise developed in dealing with a specific platform is then shared by many, and no one person ever becomes indispensable. On the other hand, there is also a danger in putting all of our eggs in one basket (however mighty Microsoft or Wind River appears at the time). In this regard it is perhaps more prudent to relax our definition ofhomogeneous a bit. In the end, “How we got there” is more important than “Where we went.” In other words, if I can take the same code and compile and run it on another platform with minimal effort, then my learning curve on the new platform is not so steep. On the contrary, I am likely to feel at home in my new environment, and my general knowledge will increase steeply. I then also have the ability to “plug and play” at the front end, where (as long as I adhere to our initial ansatz of the control system model) I can completely alter the machine and reconnect it to the control system, without any of the other participants knowing. Rather, the application layer will know that something has changed and figure out how to deal with it, but the applications programmer still sees the same object on the other side of the API.

As to the data exchange mechanism, the homogeneity of the HERA system lies in the API, which is an identical C interface across all of the previously mentioned platforms. Reading hardware might of course require a completely different expertise on one system than on another, and in this light there remains a strong bias in sticking to the PC environment. Nonetheless, as UNIX offers a fairly standard working environment across many platforms, UNIX solutions to controls problems are not to be avoided, on the general principle of hedging our bets. Being able to easily incorporate a special UNIX solution to a controls problem and at the same time to allow an engineer who has tested and developed his hardware with a PC to trivially include his work in the control system proper are both good capabilities to have.

The waters will undoubtedly be muddied even further in 1996 as the control system API will be ported to WINDOWS-NT, i.e. to the WIN32 API.

C or C++ is the natural language of choice when interfacing with the control system API. The notable exception is under WINDOWS where, since it offers such an outstanding GUI, Visual Basic is used. Visual Basic itself fills a development tools niche so well that there has been over the past two years a flurry of activity from several vendors either to offer something similar (or better) or to port Visual Basic to non-PC platforms. The ease with which one can learn and program something useful with Visual Basic is astounding. This point should not be taken lightly, as a good many of our own CONSOLE applications have been written in this language by machine physicists whose last programming experience had been FORTRAN IV several years ago. This in turn enables the software engineers to devote more time at lower systems levels. Future incarnations of Visual Basic will almost certainly offer inheritance and polymorphism to its GUI objects, making it a legitimate object oriented language.

Acknowledgments

I would like to express my indebtedness to A. Pace and I. Deloose at CERN/PS for many of the thoughts and ideas presented here. I would also like to explicitly thank S. Herb, K.H. Mess, H.G. Wu, and F. Peters for many useful discussions and rounds of constructive bickering.