EUROPEAN ORGANISATION FOR NUCLEAR RESEARCH
ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE

CERN – PS DIVISION

PS/CO/Note 99-22 (Tech.)

The PS Controls for Newcomers

Javier Serrano

ABSTRACT

This document is a 50-page effort to clearly explain both the internals of our control system and its interface to the outside world. The target audience is twofold. First, as the title suggests, it should provide newcomers with a quick way to get a complete picture of our system. Second, and most important, it is intended as a basis for understanding with our usual partners (operators, hardware specialists, etc.). People concerned by controls outside our group will get a clear view of the philosophy in use at the PS Controls Group so that discussions, requests and suggestions will be much more efficient.

The document touches upon a variety of subjects in an informal, easy to read style, including the network, hardware and low-level software, timing, application programs support, exploitation and current projects.

Geneva, Switzerland
17 September 1999

Preamble

This document is the result of an excellent initiative of Javier Serrano. It is the type of overall description of our control system every newcomer will love to find to understand quickly the system and get integrated quicker in our community. I also recommend the reading of this note to everyone who has somewhat to deal with the control system even if he thinks he already knows pretty well this system.

Besides technical descriptions, Javier Serrano gives his comments on our system, which might sometimes be considered as controversial, but I see them as beneficial as are all comments coming from a freshly engaged collaborator.

So, I would like to thank Javier for this document which is bound to be updated periodically to keep in tune with our fast-moving environment.

Bertrand Frammery

TABLE OF CONTENTS

1. Introduction

This document is an effort to provide newcomers with a quick introduction to the PS control system. This system performs the unbelievable task of controlling and synchronizing huge amounts of devices, but it turns out that understanding how this is done can be just as unbelievably complicated. I have tried to convey the excitement one feels when one finally realizes that it all comes down to some basic underlying concepts that are implemented in different ways around the complex.

People who, by their work and projects, interact continuously with the Controls Group will find here a synthesis of everything they need to know to emit realistic requests and to understand the proposed solutions. Also, people concerned with control systems in other divisions of CERN will see how their problems are handled here and will be able to suggest better solutions they may have found.

Making this document has been a very rewarding experience, but also a tough one. Fortunately, I have been helped by all my colleagues in the group who have generously offered their time and have shared their vast knowledge and experience with me. However, any mistake you find in this paper is exclusively my own responsibility.

The document starts with a small introduction explaining where the CO Group stands in the PS, much as the “YOU ARE HERE” sticker you find on commercial centers floor plans. Then we go on to describe the core of the control system, starting from the computer network and ending in high-level software issues after describing hardware, low-level software and timing.

Once the basics are clear, we devote a chapter to exploitation, maybe the most important people in the group. They can fix any part of the system at 4 a.m. on a rainy Sunday, they eat concrete blocks and they can fly, not to mention their infra-red vision capabilities. Thanks to these supermen the controls system runs smoothly all year round.

Finally, just as every Particle Physics book ends up with a “Beyond the Standard Model” chapter, we also go beyond and present a brief introduction of current projects in the last chapter, followed by a conclusion and some guesses on where we might be going in the future.

So! You bought a new control system and you want to start using it right away? Turn the page and find out all you can do with this no-frills guide to the PS-CO Universe.

2. Our place and mission in the PS complex

The Proton Synchrotron (PS) Division is one of the two divisions involved in running accelerators at CERN. It is committed to provide lepton, hadron and heavy ion beams to physics facilities and to the other CERN accelerators. Because it has such a variety of end users, each with a different goal in mind, the complex has to be very versatile concerning particle production. Reliability is of course another big issue; performance is measured by the percentage of time when a useful beam is provided. In order to keep this at the highest possible value, a great effort is needed to combine and synchronize the work of every group in the division.

There is no way to understand the role of the Controls group without an elementary knowledge of the other groups in the division: Circular Accelerators & Areas (CA), Hadron Production (HP), Lepton Production (LP), Beam Diagnostics (BD), Power (PO), Radio-Frequency (RF) and Operations (OP). The first three are responsible for the design, assembly and maintenance of accelerator facilities, and differ in the shape of the machines (circular or linear) and in the types of particles they produce. To achieve this, they rely heavily on the second three, which are specialized in critical aspects of accelerator-related equipment. Finally, the Operations group is responsible for running the accelerator complex in the most effective manner to maximize performance with the available resources. And this is where the Controls group comes into play. The Operations group must have tools that allow them to measure beam quality and to modify machine settings accordingly. The hardware and software necessary for controlling the accelerators in real time fall mainly within the responsibilities of the Controls group.

Needless to say, this can actually get quite complicated. Vast amounts of signals have to be monitored. Digitizers sample data lines continuously and produce events that trigger lots of actions. Timing information is distributed all around the complex to allow synchronization of hardware and real-time software tasks.

Application programs acquire data and modify them to present them in an operator-understandable way. These programs can be developed by people outside the group if the need arises, but the Controls group offers software platforms and support for these developments. The data travel though a computer network which is also managed by the group and which allows direct communication between the physical equipment and PCs or workstations in the offices.

Besides this, a section is devoted to the day-to-day exploitation and maintenance of the equipment. This is a necessary feature of any group that is committed to 24 hours a day of reliable operation.

3. General layout of the control system

By general layout, we mean a clear and concise description of how information flows from the workstations[1] in the offices and control rooms to the physical equipment. The architecture is mostly based on an Ethernet network where both workstations and front-end VME computers are connected.

VME modules are standard Europe-size printed circuit boards that are plugged into a VME crate ("chassis"), also called DSC[2] in the PS context. Each VME crate has a controller module[3] running a real-time operating system called Lynx-OS and which is able to communicate through the Ethernet link. Crates are housed in racks and are placed all around the PS complex, as close to the actual equipment as possible. When a VME controller receives an instruction through the network, it performs the corresponding action on the VME bus, telling the other modules on the bus what to do.

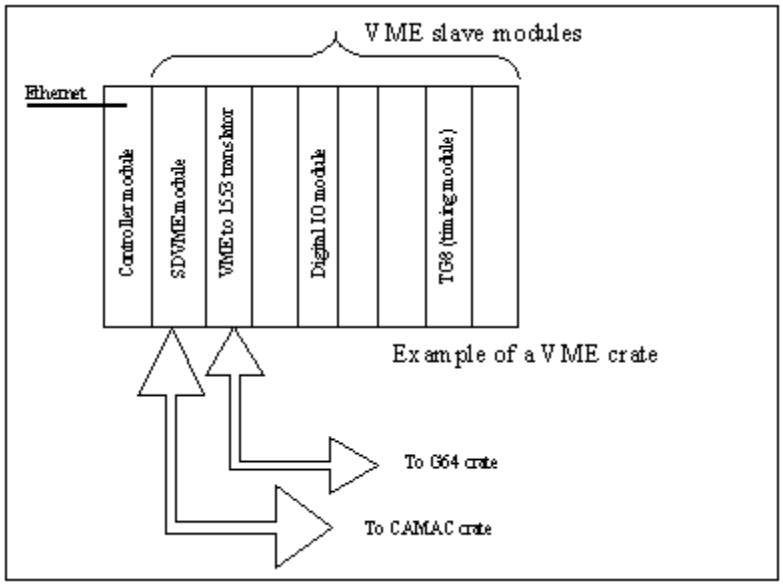

Figure 1. Example of a VME crate.

Besides this standard approach, there are cases where customized solutions have to be adopted. There are still over 80 CAMAC crates functioning in the PS, and which are being progressively replaced by the newer VME crates. These CAMAC crates are controlled by a VME module called SDVME that understands the Serial CAMAC Protocol.

[1] The reader must interpret the word "workstation" as "workstation or PC". Control from PCs is achieved through the gateway known as the "passerelle", to be described later.
[2] Device Stub Controller.
[3] Also called CPU module.

Power converters for the accelerators' magnets are generally controlled via another standard type of crate, the G64 bus. These crates rely mostly on 6809-based controller modules and communicate with VME crates through a serial field bus called MIL-1553. They can also be controlled by CAMAC crates through what we call "Quad/Single transceivers".

Yet another exception to the standard approach is the New Analog Observation System (nAos), to be described later. In this case, the VME eXtension for Instrumentation bus (VXI) has been used to allow control of fast oscilloscopes. VXI controller modules behave mostly like VME controllers and are linked to the same Ethernet network. The major difference is that they run locally a different real-time operating system called VxWorks.

3.1. The network

CERN's computer network is an extremely complex one. We will focus on how the PS Controls network fits into the general picture and then we'll go on to describe some of its internal features. A general overview of the data path to get from one of our computers to the outside world is depicted in figure 2.

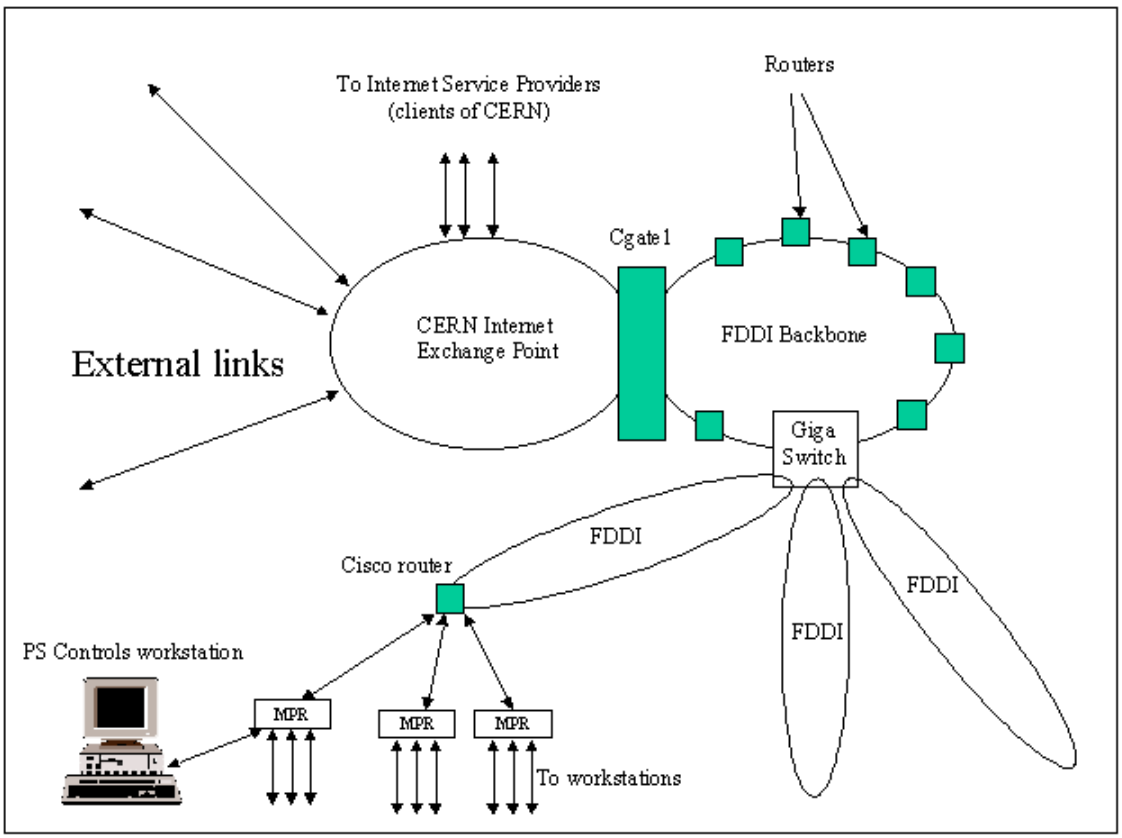

Figure 2. Data path from a Controls workstation to the outside world.

A little description is necessary to understand the jargon. CERN is connected via fast fiber optics connections to other important Internet sites. Information coming from say the U. S. enters CERN through what we called “external links”. Internet service providers (ISPs), are clients of CERN and offer their own clients connections to the Internet using CERN’s infrastructure. The Cgate1 machine controls network traffic from CERN’s Internet Exchange Point to the FDDI backbone. A backbone is a network made only of routers, while FDDI means among other things that the physical support is optical fibers.

One important part of the FDDI backbone, the Giga Switch, plays a special role: it interconnects several FFDI networks, one of which contains the Cisco router that controls access to the PS workstations. Multiple repeaters (MPRs) are then charged to deliver the information to each workstation.

What happens after the Cisco router is depicted without complete details in figure 3.

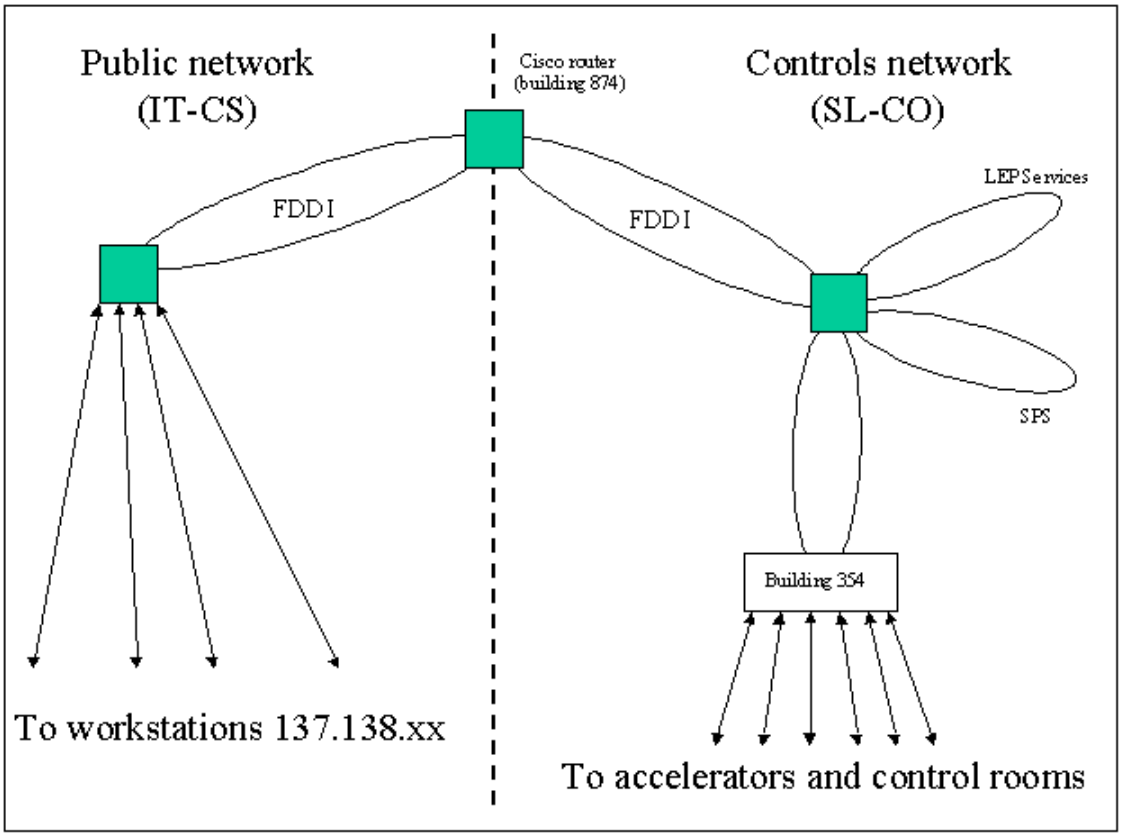

Figure 3. The public and controls networks.

The controls network is logically and physically isolated from the public network. In fact, the Cgate1 machine already prevents any information coming from the external links to reach the controls machines. This is done for security reasons and to ensure that the 24 hour a day operation is not upset by problems in the public network.

3.2. The workstations and AIX

The workstations in the control rooms and in the offices run on IBM's proprietary version of the UNIX operating system: AIX. By far, the most important concept in the network architecture at this level is that of *file server*. These can be described as belonging to four different groups:

- AFS services are offered by servers in the IT division. These contain the personal home directories, public domain software (e.g. the SNIFF applications development package, Frame Maker and Netscape), and the Oracle SQL Forms development software among others.
- NFS is used in servers in the PS to hold more critical information. The point of this is to avoid depending on the Public Network for data which are only used in the PS. The three servers used for these services are currently[4] called PSAS01, PSAS02 and PSAS13. They hold the sources of the software developed to control the accelerators and the development environment specific to the PS contained in the /ps/local directory.
- Another NFS service is devoted to hold the most critical information, that is the Operations environment.
- Finally, an Oracle server contains a database with lots of information concerning hardware modules installed throughout the complex, software running in each DSC and other accelerator-related data. The name of this server is PSAS12 as seen from the PS environment, or SLDB01 if called from the FDDI level. This server holds data for both the PS and the SL accelerators (much more on this later).

A fairly complete representation of what we described until now can be seen in figure 4.

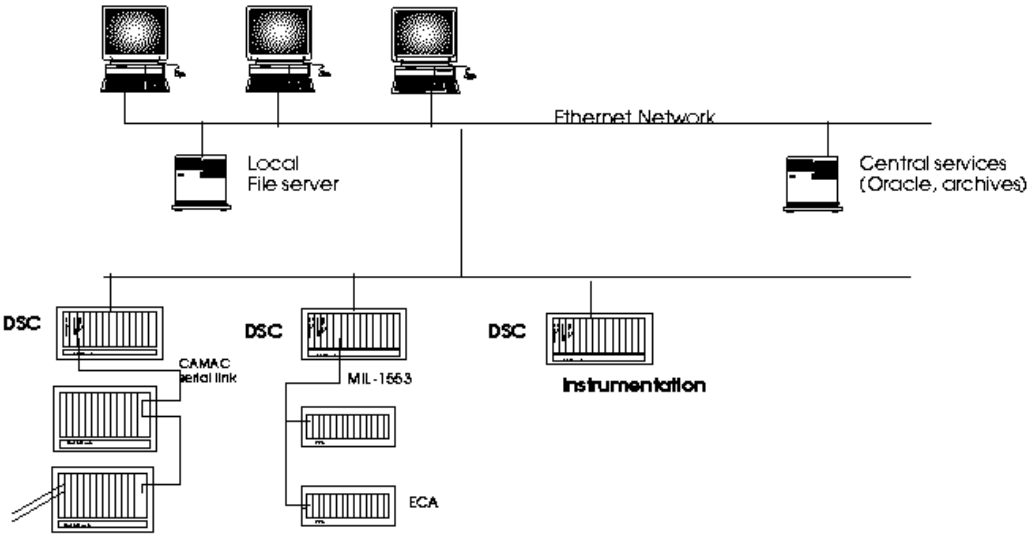

Figure 4. General view of workstations, servers and front-end computers.

[4] Notice that file server names are subject to quick change.

3.3. The DSCs and Lynx-OS

As we said before, DSC is the jargon name in the PS for a front-end VME computer. These controller modules run a UNIX-like operating system called Lynx-OS[5]. The main features of this operating system are:

- It is a real-time operating system. This means that any instruction is guaranteed to be executed within a certain time that the user can control in a variety of ways. Traditional UNIX-like operating systems guarantee that the instruction will be executed but they are unable to predict when or even to give an upper bound to the time when it will be processed. The UNIX community established in one of the POSIX standards the requirements that an operating system had to fulfill to call itself "real-time". The version of Lynx-OS used in the PS is 100% POSIX-compliant.
- It is a commercial package. This means that the source code is secret and that fairly high license prices have to be paid to use it in each DSC.
- The interface for writing device drivers has been carefully designed to allow writing these drivers completely in C. This was a problem with the preceding operating system used in front-end computers, OS-9, where most of the drivers needed some of the coding to be done in assembly language, thus increasing the complexity and the possibility of errors.

Lynx-OS was chosen as a standard for the PS and SL Controls front-end computers and it is used throughout the complex in all VME-based systems. For the particular case of nAos, as we said, the choice of the VxWorks operating system was imposed because no diskless controller modules existed in the VXI standard which supported the Lynx-OS operating system.

Indeed, another one of the major requirements for a CPU module to be used in the PS Control System is to have no hard drive in it. All controllers are booted remotely from servers and store local copies of the required Lynx-OS or VxWorks functions in RAM at boot time. This is done through a remote boot protocol called BOOTP.

Finally, in order to guarantee real-time answers, no swapping is used. This means that the memory size will always be a limitation in our VME front-end computers. Therefore, there is no point in trying to use them as standard UNIX systems.

[5] Not to confuse with the text mode Internet browser called Lynx!

4. Accessing the equipment

The general philosophy adopted for remote access of the equipment in the PS is that of maximizing modularity. The goal is to be able to fix most of the possible problems by replacing some part, be it hardware or software. This requires of course a big standardization effort, as we'll see in this chapter.

The PS standard for controlling hardware is the VME bus. In short, VMEbus is a computer architecture which is widely used in industry mainly thanks to the great number of off-the-self components one can buy. Developing a VME module is also fairly straight forward compared to other architectures. In fact, due to the special nature of accelerator control, more than 50% of the VME modules we currently use were developed here. The most salient features of the VME standard are the following:

- It is an *asynchronous* bus. This means that the speed on information interchange is only limited by the time two modules take to perform a handshake, that is to agree on how the exchange will be handled.
- The data bus width is 32 bits. Read and write cycles can be executed with 16, 24 or 32 bits. One can also transmit 64[6] bits at a time by multiplexing the data and the address buses.
- The address bus width can be set to be 16, 24, 32 or 64[6] bits by the same method as with the data bus.
- Slave modules can interrupt master modules with seven different levels of priority. This is of crucial importance for real-time tasks, as we'll see later.
- More than one master module can co-exist in one VME crate. This feature is not used in the PS environment.

The general architecture of the acquisition system follows the *standard model*, a distribution that is widely used in particle accelerators and specifically in the PS complex at CERN. As an example, the control of an HP5335A Universal Frequency Counter is depicted in figure 5.

[6] This feature is not used in the PS.

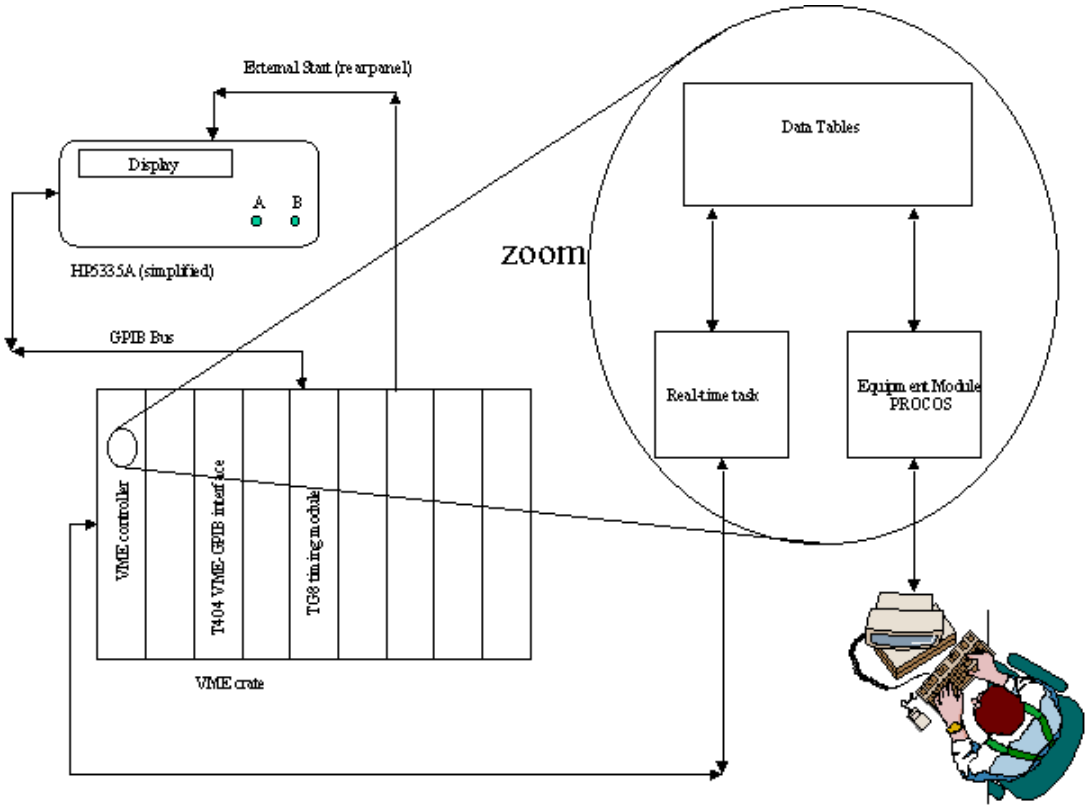

Figure 5. A general view of the control and acquisition system for the case of a frequency counter.

During normal operation, all the information concerning the counter, such as settings and acquisition results, is stored in a data table contained in the VME controller's memory. A real-time task running on the VME controller module performs *read* and *write* operations from and to the data table. The only way for the user to interact remotely with the counter is through this table. There are many ways to do this: one can use the *nodal* interpreted language, write one's own C program using a set of provided functions or use the *passerelle* to access the equipment with a PC through Excel or Visual Basic. All of these methods end up using a series of functions (also called *PROCOS* for PROperty COdeS) that communicate directly with the data table.

As a general rule, the user will write some specified values in some variables (also called columns) of the data table. These values will be read by the real-time task to decide the control values that will be sent to the counter. When one is interested in acquiring data rather than in actually controlling the device, the process is reversed: the real-time task writes continuously[7] the acquired values in some specific columns. The user must just decide which columns to read and when to read them.

[7] In fact, all these read and write actions are triggered by the arrival of VME interrupts.

The set of functions and data available to the user for controlling the device is called an *Equipment Module*. We'll describe these and the real-time tasks in more detail in the following paragraphs, but we must note that this standard model can suffer slight variations in practice. For example, sometimes the Equipment Module functions (called properties, as in the object-oriented paradigm) can access the physical equipment directly instead of having the real-time task to do it. As a general rule, however, it is always good to separate conceptually the Equipment Modules from the real-time tasks and let the latter be the only ones to call the device's driver to access the equipment.

As we mentioned before, the whole layout and configuration information of the control system is stored in Oracle tables contained in an Oracle database server. This is an extremely powerful method to handle such pieces of information. Startup files for every DSC are directly generated by programs using the database information. This also means that any replacement operation on the hardware side must be followed by an entry in the Oracle tables. Such entry operations are made easy by the use of Oracle forms where the user is prompted to enter information in different fields.

There is one key aspect we left out from the explanation of the counter's example above: timing. Synchronizing all processes that govern the accelerator is extremely tricky. To simplify things, a standard approach has been adopted that involves real-time tasks and TG8 modules. Very briefly, real-time tasks are programs that contain functions which are "awakened" by the arrival of VME interrupts to the controller module. The TG8 is a general-purpose timing module that generates these VME interrupts on the occurrence of external events.

Finally, an additional complication in timing comes from the sheer nature of the PS machine. While our client machines and physics facilities are always concerned with the same type of particles, the PS complex has to supply different beams to different machines at different times. That is, time multiplexing is used to make the clients look as if we were only working for them. The consequence of this is that the different types of beams to be delivered are organized in a sequence of cycles called "super-cycle" which is repeated continuously. Each cycle in this super-cycle can be concerned with different particles at different energies, so the whole complex has to change setup every time a new cycle is executed. This concept of time-multiplexing is called PPM (Pulse to Pulse Modulation) in PS jargon and we'll try to explain it in more detail in the chapter devoted to timing.

4.1. The hardware

Under the label of "hardware", there are many different kinds of equipment. The chain starts with the VME processor modules. The first series of these that was used in the PS controls environment was made up of Motorola's MVME147. Later on, and until 1997, we started using the MVME167, also from Motorola. Most of these boards only had 8 Megabytes of RAM, and this was clearly insufficient, so a replacement had to be found. The newest CPU modules in the complex are now the RIO8062 from CES. They were chosen because of their speed, their amount of memory and also to get the same processors as the SL-CO group and eventually share device drivers with them. The RIO8062 are based on a PowerPC microprocessor.

The second big group of modules concerns digital and analog input/output (I/O) modules. On the digital side, the ICV196, VMOD DOR and VMOD TTL are used where simple I/O is needed, while the workhorse on the analog side is the MPV908 sampler. A set of "standard" modules[8] supported by the group have been defined to avoid the proliferation of many different pieces of equipment that do the same thing, thus making maintenance easier.

The timing modules are important enough to deserve special mention. As we'll see later, timing in the PS is based on a multi-drop network where a special module called the Master Timing Generator (MTG) outputs timing-related information in the form of 32-bit serial words. These words are locally decoded by TG8 modules which produce front-panel pulses and VME interrupts accordingly. Both the MTG and the TG8 cards contain embedded software running on an on-board micro-controller, Motorola's 68332. The MTG card also uses a 68HC11 micro-controller to manage the serialization of the messages to send through the network.

As we have already mentioned, the VME standard is not the only bus architecture in use at the PS. For various reasons, some of them economical, some of them technical or historical, other control and acquisition schemes co-exist with VME:

- G64 crates are used for control of power converters and other simple applications such as stepper motor control.
- Leftover CAMAC crates are still of fundamental importance. Their complete replacement by VME crates is a clear goal but manpower and money stand in the way, so it has to be done slowly on a year-by-year basis.
- In addition to the ones already mentioned (MIL-1553, GPIB, Serial CAMAC), a variety of field buses are used to control more or less exotic equipment. These include CAN bus and RS-232.

[8] See the note by W. Heinze in the references for chapter 4.

- Finally, sometimes the right instrument just doesn't exist in the VME standard. Most of the times, oscilloscopes, spectrum analyzers and other state-of-the-art equipment are controlled via the GPIB bus.

Another important and sometimes forgotten topic on the hardware side is that of electrical standards for signals. Three main types are used in the PS for transmission of information: $\overline{TTL}$, TTL and blocking. Figure 6 shows what is meant when a PS hardware specialist talks about $\overline{TTL}$ signal transmission.

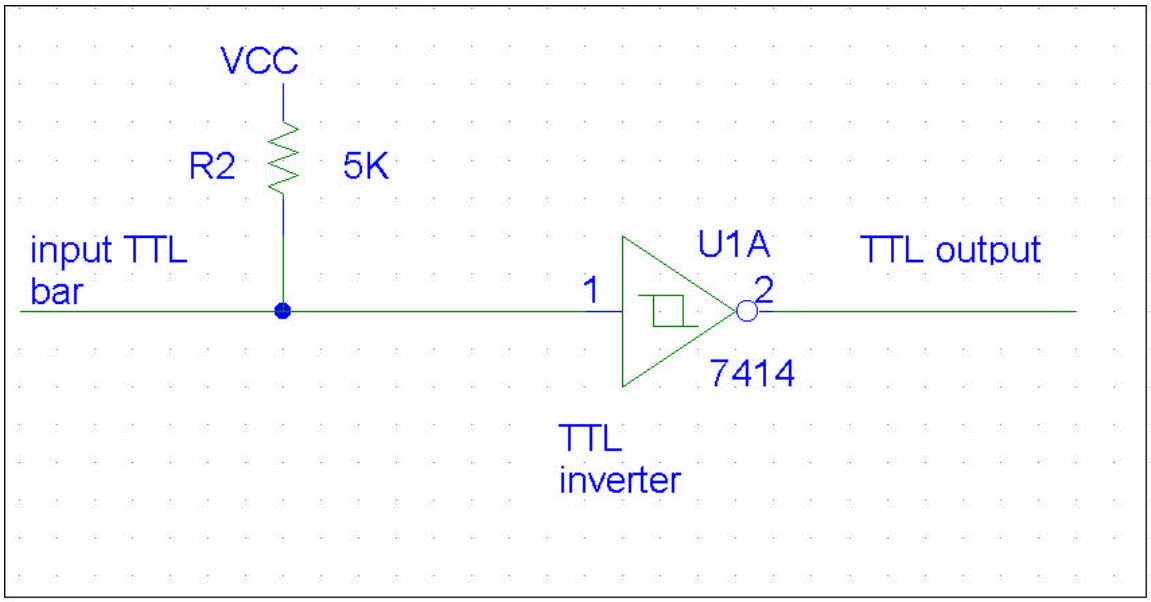

Figure 6. The $\overline{TTL}$ signal "paradigm".

A fairly high impedance (5 kilo-ohms in the figure) goes to the VCC supply before entering a TTL gate. The effect is that power consumption when a driver[9] pulls the input high is almost zero due to the big input impedance of the TTL gate, while current flowing through R2 when the input is driven low is higher but still acceptable (1 mA max. in this example).

[9] Don't confuse a TTL driver with the device drivers we talked about before! The first are chips that "drive" a line while the second are software entities that allow control of a device.

On the other hand, RF signals often need impedance matching at the input, so TTL transmission (figure 7) has to be used. In this case, a low-value resistor goes to ground before entering the TTL gate. This avoids reflections of fast signals, but has the bad effect of boosting power consumption when the input is driven high. Besides, this type of transmission needs a line driver circuit like the 74S140.

Finally, the "blocking" electrical standard is used for timing pulses which have to run through long distances. The only specification is that the voltage in the high state has to be larger than +5 volts. Most of the times it is between 12 and 24 volts. $\overline{TTL}$ to blocking signal adapter modules exist and are used all around the complex to interface signals coming from far away to local acquisition and control modules.

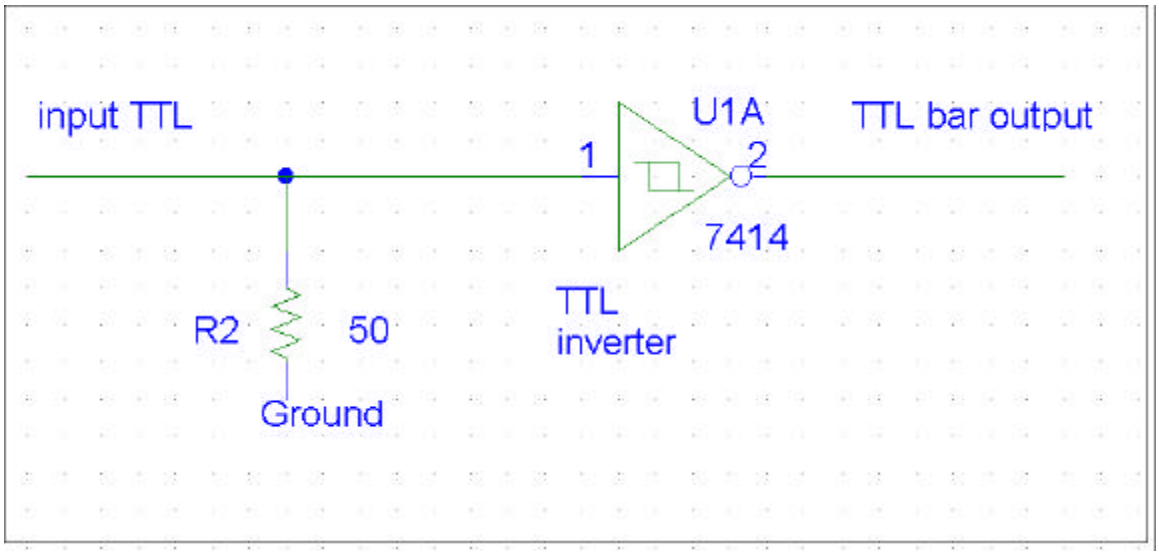

Figure 7. The TTL signal "paradigm".

4.2. Low level software

Many of the specific features of the PS Control System are on the low level software side. The aim driving the group's efforts was to hide unnecessary complexity from application developers. Configuration management is the PS jargon term to designate a process that prevents high-level programmers from getting lost in a mess of hardware addresses, interrupt vectors, crate locations and so on. It also ensures portability of software with respect to changes in platforms (e.g. transition from 68000 to PPC processors) or in physical address (the concept of logical address is used to achieve this). Thanks to this configuration effort, real-time tasks can be written concentrating only on the essential logical aspects of the hardware.

These actions resulted in a model based on the use of object-like software entities called Equipment Modules. In fact, one could argue that the Equipment Modules should be described in the high-level software chapters since they really are a layer that provides a uniform approach to access hardware from application programs.

This chapter contains information on all these subjects and also on the centralizing tool that allows a unified approach for all low-level software and hardware: the Oracle database. A small paragraph at the end is devoted to the specific case of the programs used to control the power converters.

4.2.1. Configuration management.

Each DSC in the PS control system is based on a specific hardware configuration and runs a dedicated set of real-time tasks and drivers. These features are described in the rc.local command file which is part of the startup sequence run by Lynx-OS after a reboot.

Due to the continuous increase in the number of DSCs, a management by hand of the rc.local file is not reliable enough and would lead to a waste of man power. Besides, any change in configuration can have unpredictable consequences and side effects.

To solve this problem, the group had to define a means to handle the DSCs hardware configuration and the associated startup procedure. This management is based on the use of an Oracle database containing all the configuration information for any given VME crate. The input of these data is made easy by the use of an Oracle Forms application called "hardware". The following are examples of information stored in the database through the "hardware" interface:

- The VME crate name.
- The CPU board used in that DSC.
- All VME modules present in that crate.
- The links to subsets supported by other hardware systems: 1553, CAMAC, …
- Information about drivers running on that DSC: associated installation program, driver name, hardware element controlled by the driver, …

With all these pieces of information, the "hardware" application is able to generate the rc.local file required by the DSCs at startup. This working method also makes exploitation easier and even possible if we consider that man power tends to decrease while the number of hardware modules to be controlled keeps increasing.

Another important topic is that of logical addressing. After declaring all modules in the database, the generation program inserts some comment lines in the rc.local file of the DSC. At the startup of the system, and before any application is launched, the ioconfigInstall program reads back the configuration description lines from the rc.local file and sets up, in a shared memory segment, the tables describing the declared configuration.

These tables allow some special routines to translate a logical address into a physical address, so if a real-time task or any other application uses these routines to access the equipment, they do not have to know about physical addresses at compile time. This process guarantees independence of platform and of physical address changes and makes the whole work of application developers much easier.

4.2.2. The Equipment Modules

This chapter could also be entitled "Equipment Access made easy". An equipment module (EM for short) is a collection of software procedures and data allowing to drive a certain type of equipment. Examples for actions performed via an EM are adjusting magnet fields, switching a device on and off or reading meters. There is one EM for each type of equipment, e.g. stepping motors, RF cavities, power supplies, etc. EMs can also be grouped in a composite equipment module (CM).

Like in the object oriented paradigm, every device is considered to be an instance of some class[10]. These devices, or objects, are accessible to the programmer via an interface which consists of access functions to properties, that allow acting on the device (algorithms) and changing data (variables) associated with it. Allowing access to the data

[10] In fact, the terms "class" and "equipment module" are used interchangeably in most contexts.

only through these functions allows the EM programmer to hide complexities and to ensure that the data are always in a coherent state. As we'll see later, there are ways to change the variables directly, but they are generally reserved for front-end specialists. We'll call the latter a "private" interface as opposed to the public interface made up of properties that are available to every application programmer.

An interface to the Oracle relational database machine (RDBM), called "genmod" allows the declaration of new classes with their corresponding variables and properties. One can also use it to generate instances of a given class, give them names and associate them with a given DSC.

The architecture used to implement the EM concept as seen through the public interface is depicted in figure 8.

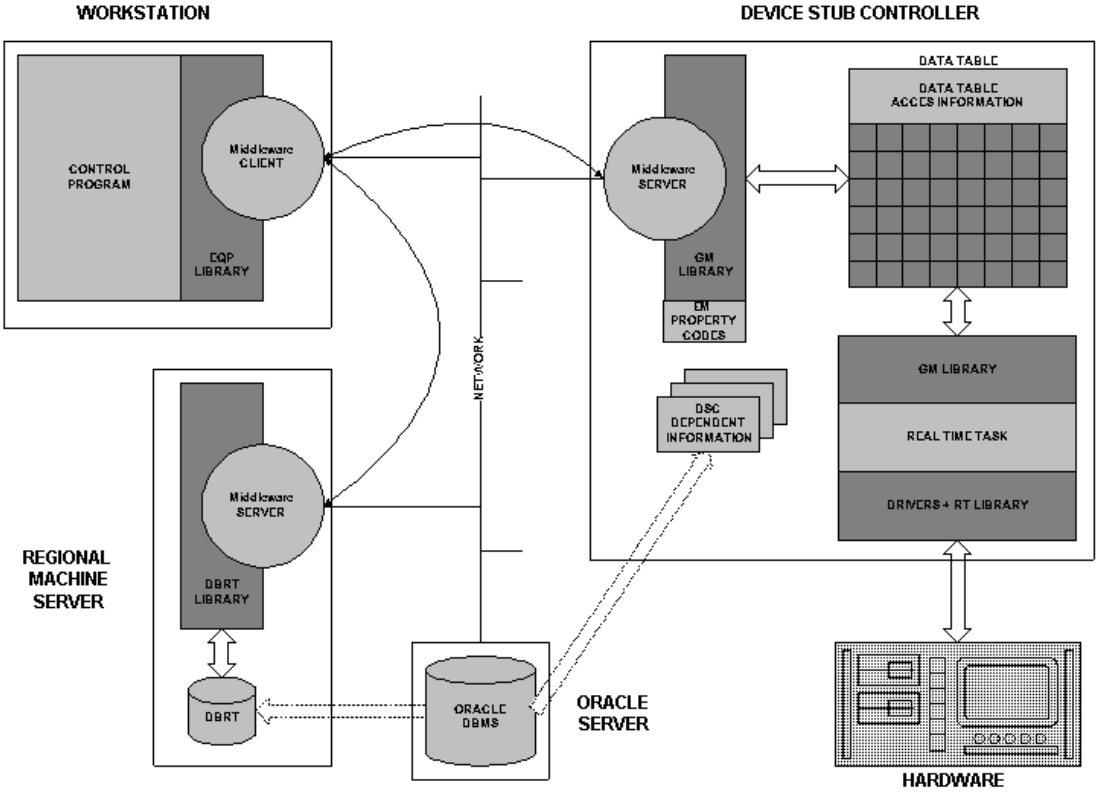

Figure 8. Communication path between the hardware and the control programs.

Some detailed explanations won't do any harm here. Regional machine servers exist for every accelerator in the complex: PS, PSB, ADE, LIN, LN3 and LPI. A control application running on some workstation uses the EQP library to access the properties associated to an equipment. The information then flows through the network using a middleware client-server mechanism.

In fact, what is designated with the generic word "middleware" is implemented using CERN's SL Division *Remote Procedure Calls* (RPCs) although this mechanism will

change in the future, as described in chapter 8. RPCs are a way for a client to ask a server to perform some function on its machine (here a DSC or a Regional Machine Server) and return the results to the client.

On the DSC side, the RPC server is a process which is able to service RPC calls by executing EM functions (procos) that act on variables stored in the data table. This table is just a shared memory segment that is reserved at startup time and that holds both static and dynamic data associated to the devices in that DSC.

The GM library is used by both the RPC server and the real-time tasks to access directly the variables in the data table. The number and nature of the different columns in the data table are defined in Oracle tables.

So now we have enough information to understand what happens from the moment one declares a new equipment module with "genmod" until an application program is able to communicate through the EM interface. The sequence is roughly the following:

- Declare, if necessary, the new class, its variables (class or instance variables, as in the object-oriented paradigm), and the properties associated to the class.
- Declare instances of that class in any DSC.
- Use the *gmake new_dtab* command in the source code environment of the DSC of interest. This command performs three main operations:

  It extracts information from the Oracle tables and translates it into C language files.

  It compiles them and links them to other programs, creating executables. These include *nodal*, the *RPC server* program and *creadt*.

  It installs the generated files in the bin directories corresponding to the DSC.

At startup, the *creadt* program is launched. Its role is to create the data table in the DSC and to fill it with the persistent information stored in a Data Table image file. The *RPC server* is also launched at startup, so from then on, any application program can control and acquire information from a device through the corresponding EM. The last program to be launched at startup is SYSGO, a nodal script that performs an automatic initialization of some devices by accessing their corresponding EM properties. The result of all these fairly complex operations is thus a very easy interface to high-level applications.

4.2.3. The real-time tasks

Real-time control of the equipment in the PS complex is generally achieved by means of software tasks running on a real-time Operating System[11]. The structure of these programs is often quite similar, since all of them can be divided in a first part where the device or set of devices involved are initialized and a second stage that consists essentially of an infinite loop where the task waits for interrupts and reacts to their arrival accordingly.

The job of connecting to interrupts and waiting for them is greatly minimized thanks to the existence of functions developed in the PS-CO group. A whole set of these functions starting with the prefix dsc_ exist. An example of their use follows:

```
/* Connect to three different interrupts, identified by integers ppmaqi, ppmcvi */

fd_int = dsc_rtconnect(ppmaqi);   /* Acquisition interrupt */
fd_int = dsc_rtconnect(ppmcvi);  /* Control interrupt */

/* Wait for an interrupt and react accordingly */

for ( ; ; ) {
                i = dsc_rtwaitit(fd_int, program);
                if (i == ppmaqi) do_acquisition();
                if (i == ppmcvi) do_control();
                …..
        }
```

The advantage of this approach is to provide a uniform and easy approach to wait for interrupts. Most frequently, the initialization phase of a real-time task involves several read operations from the Data Table of the DSC while the infinite loop generally performs many writes and a few reads from it.

There is currently an ongoing effort to improve the quality of the code used in real-time tasks. The goal is to use some basic object-oriented concepts to ensure encapsulation and code reuse as well as to improve readability and to make the set of all real-time tasks more uniform.

[11] This Operating System is Lynx-OS for the vast majority of cases.

4.2.4. The database

We have already talked many times about the ORACLE database in this document. In fact, it is such a crucial part of the control system that hardly any low-level software subject can be treated without referring to it.

The purpose of this chapter is to organize all the basic knowledge we have gathered about the database (DB from now on) into a complete coherent body. The first thing to know when speaking about the DB is that information is organized in different tables following the relational model. All the tables form the *dbco* database and are stored in the *orasrv*[12] database server.

We have already described the *genmod* and the *hardware* programs that allow easy equipment access as well as declaration of interface crates, modules, connections, and DSC programs. Both of these applications are based on tables containing the necessary information, but these are not the only tables sitting on the server. For instance, the EQUIP tables contain information about accelerator equipment, timing-related persistent information is stored in the PLS tables and so on.

Accelerator control programs need information about the accelerator hardware and software. They could get this from the Oracle relational database with embedded SQL[13] statements. However there are some disadvantages in this approach:

- The program designer has to know SQL and the table structures.
- The program must go through a pre-compiler to translate the embedded SQL into normal program statements.
- This gives difficulties with debugging programs.

The DBRT system was designed to give application programs a more simple and reliable access to the data. It is a simple database which is accessed read-only by normal procedure calls which return a single record when given a key. The data in DBRT are downloaded from the relational database.

Many application programs can live with the limitations of DBRT but, if you need write access to the database, or support for complex queries, you still need embedded SQL to the relational database. This is the case of the nAos system, to be described in chapter 6.

Finally, the problem of manually introducing data on the DB is solved by the use of ORACLE forms and menus. These are interactive tools that allow user-friendly interaction with the data tables.

[12] This is an alias to avoid continuous changes in applications, as explained before.
[13] Structured Query Language. It is a language that allows querying a relational database for information.

4.2.5. Specific software

This paragraph is devoted to a brief description of the control of power converters in the PS. Part of this control is done with the standard Equipment Module & real-time task approach while more specific software[14] is used to interface power converters with our standard control system.

A common feature of all power converters is that they have to be supplied with information to control the current they deliver and they all have some kind of status registers one can read. The control of current can be done either digitally or by means of an analog voltage sent to the ADC of a power converter's interface.

Very often, one of two standard setups is used to control power converters:

- In the first case, a VME crate contains an SDVME module that controls a CAMAC crate. The latter contains a Quad module that communicates with a Single module sitting close to the power converter.
- The second setup also involves a VME crate but this time it communicates with a G64 crate through a 1553 serial link. ADCs and DACs sitting in the G64 crate communicate directly with the converter.

In both cases, a real-time task runs on the DSC and it has an Equipment Module associated with it. However, in the case where a G64 crate is used, a real-time task runs locally on the G64 controller module and communicates with the VME-based real-time task through the 1553 link. The 1553 standard represents numbers in floating-point format, while the power converters' interfaces only understand numbers in fixed-point format.

Therefore, the G64 task, which is a Pascal / Assembly language program that runs with no operating system supporting it on the G64 controller module, has to carry out two tasks: the translation of numbers from one format to the other in both senses and the management of the 1553 controls protocol.

[14] This software is under the responsibility of the PO Group.

5. Synchronizing the whole thing: timing

The PS complex consists of six interacting accelerators working together, which, from cycle to cycle, produce beams varying in end user, particle type, energy, time structure and beam-geometry. Since the introduction of the new timing system, the sequencing of the PS accelerators now depends dynamically on their status, so that sequence changes in real time are now provoked automatically. This greatly improves the complex's response time to changing end user requests, and simplifies the task of the machine operators who no longer need to program it manually.

Coordinating this intricate time sharing particle factory is the MTG (Master Timing Generator), which broadcasts messages around the complex containing summary information on what each part must do next, and the timing needed to carry it out. These messages are received by TG8 VME timing modules which then provide nearby equipment with timing pulses, and the VME host processors with task synchronization events and summary information.

Timing is probably one of the most difficult subjects in the PS controls system. The jargon involved is specially tricky due to the fact that new concepts are often named after old solutions that dealt with the same problem. With this in mind, it will surely help to start off with a historical introduction describing how the timing system architecture has evolved since the PS was built.

5.1. The timing system architecture

At the beginning, an old IBM computer called the PLS[15] managed the synchronization of all the pieces of equipment in the whole complex. To achieve this, it used an output register of 256 bits, each one connected by a line to all devices. It was the devices' task to correctly interpret the contents of the lines to find out what to do at a given moment. The PLS machine contained a 256-bit wide parallel shift register and its only task was to shift the contents that had previously been edited by hand by an operator.

After a while, lines were divided in different groups. It became clear that some pieces of information needed more than one line in the register. For example, if one wanted to transmit the particle type to a power converter, three lines would be needed to make the difference among electrons, positrons and protons[16]. As more particles could be accelerated in the PS, it was decided to use 4 lines to encode the particle type. The other 252 lines were concerned with other types of information.

[15] Program Line Sequencer. The meaning of this will become clear in a minute.
[16] Only one of the lines could be asserted at a certain point in time. This is called an *exclusive* group of lines.

Later on, someone realized that some pieces of equipment needed some time before the information sent on the lines could be used. This could be due to loading times for pulsed power converters or to logic manipulation of the information before it could actually be used in the target equipment. To remedy this lack of time, two output registers started being used around 1980. One of them contained information about the current state of the system and the other one contained the next state. In that way, pieces of equipment concerned by this timing problem could use the *next* lines instead of the *current* lines to have the information in advance and thus have enough time to react.

A big step forward was taken when the mess of 2*256 line cables was replaced by a single cable where information traveled in a serialized manner. The whole system was made up of PLS encoders and decoders. The former were NIM modules with an internal 10 kHz quartz that serialized the 256 bits before sending them to the timing cables. This operation was therefore completed in 25.6 ms and was performed just before the beginning of a cycle[17] . On the other side of the cable, PLS decoders sitting in CAMAC crates de-serialized the messages and were able to route some software-chosen lines to their front panel outputs. Another CAMAC module used in those days[18] was the PLS receiver. Its role was to receive the 256 bits, also known as the *telegram*, and store them in memory so that the local CAMAC controller could have access to them. Most of the times, however, there was no CAMAC controller in the crates and Nord computers were used to control a set of CAMAC crates through the Serial CAMAC bus.

At the beginning of the 90's, a big thinking effort lead to the introduction of the *beam* concept. Basically, a beam in the PS sense of the word is a collection of cycles in different accelerators that are synchronized to achieve an end product. The MTG and TG8 modules were developed with this new philosophy in mind. The former plays the role of telegram emitter while the latter is a de-serializing receiver VME module capable of producing VME interrupts and front-end pulses related to the contents of the timing cable.

Telegrams are currently made up of a set of 32-bit words, each one corresponding to a group[19]. Each telegram is made of a number of words ranging from 20 to 30 depending on the accelerator. The first 16 bits of each word are descriptive: 4 for the machine (accelerator) concerned by the word, 4 for the type of event[20] and 8 bits for the group number. Groups such as particle type, end user, beam destination and cycle number are identified by these group numbers. The remaining 16 bits are the value for the given group. For example, in a PS telegram, the particle type group could have a value of 1 for protons, 2 for antiprotons and so on.

---

[17] A cycle here means a magnetic cycle. This will be explained in section 5.2. For the moment, you can think of it as a time interval devoted to produce a given beam.

[18] Actually, these modules are still used today in the LPI.

[19] A group here means exactly the same as it meant 30 years ago. Then it made sense to call it a group, since information about particle type, beam destination, etc. was coded in groups of n copper lines. The telegram is now transmitted through a single cable but the words "group" and "line" are still used.

[20] This tells us if the word we are reading is to be interpreted as part of a telegram or as another type of event.

Telegrams are broadcast in every cycle but the words that make them up can come at any time and in any order. Besides, the 32-bit words can have different event types. This means that they can be part of the telegram, but they can also be what we call C events. These arrive every millisecond and are used to synchronize both software and hardware. There are also date & time events, simple events (such as the one that signals the beginning of a cycle) and repetitive events. These come out every couple of seconds or so and contain a description of the groups in the telegram that is used by some non-VME receiver modules. Words that make up a telegram are validated by the arrival of a simple event called Ready PLS (RPLS). This means that receivers have to store the telegram information as it arrives and keep in mind that it will only start being valid after the RPLS event arrives. Three such telegrams (PSB, PS, AD) are produced and distributed to drive the six PS machines.

Now, we've come a long way since the 50's! There's no way to continue our timing discussion without a basic knowledge of cycles and beams as they are understood in the PS. So let's go for it.

5.2. Cycles and beams

As you surely know by now, coordinating the path of particles through the different accelerators in the complex is a complicated business. A very simplistic view of what happens when particles go from the Booster (PSB) to the PS is depicted in figure 9. Let's see how it works.

In circular machines, there is a linear relationship between the particles' momentum p and the mean magnetic field that the accelerator has to provide to keep them inside the tube. What we see in the figure's vertical axis can thus be interpreted either as the magnetic field B or as momentum[21]. Particles with higher momentum describe circles of larger radius if the magnetic field is kept constant. Unfortunately, there is a higher limit on the magnetic field that our power converters can supply, so the only way to increase the momentum of the particles once this field has been reached is to take the particles to a larger accelerator. This is what happens in the PS with the booster, an internally tangential circular accelerator that injects particles in the PS.

In the figure, particles start being accelerated in the booster at time T1. When they reach their maximum momentum at T2, the PS can start injecting particles from the booster. This is done in the small *flattop* between T2 and T3. Then the PS starts accelerating the particles until it reaches its top magnetic field. During the PS flattop, particles are ejected

[21] The relationship between momentum and energy is not as simple. Basically, $E = \sqrt{p^2c^2 + m^2c^4}$, so that E and p are almost the same at high energies for light particles such as electrons (in units where c=1) but they are quite different for heavier particles such as protons.

from the PS to the SPS or to other experimental areas. At time T4, the ejection is finished and the magnetic field in the PS starts to decrease to start another cycle. Different cycles can be aimed at different targets and contain different particles at different energies.

The length of the booster cycle is currently 1.2 seconds, while that of the PS can be any number of these 1.2 second basic periods. This means that after the booster has executed a cycle for the PS, it has time to work for other machines or physics complexes such as the ISOLDE experimental area, executing what we call parasitic cycles.

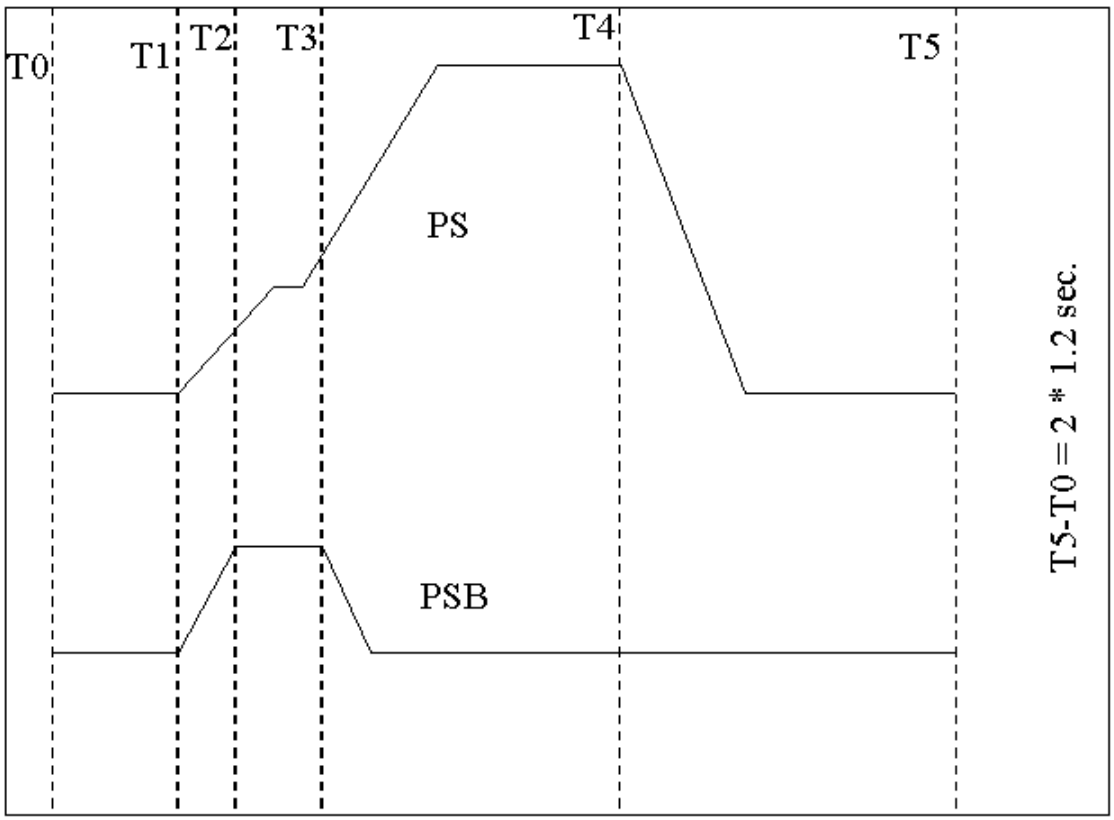

Figure 9. A schematic view of the PS and PSB cycles.

Now that we understand cycles, let's explore the concept of *beam* as PS timing specialists understand it. The need for a *beam* point of view arises when you deal with more than one accelerator. Particles must be transferred from one machine to another and some central intelligence (in our case the MTG) must coordinate these transfers. The whole situation can be schematically represented as a directed graph with nodes and arcs linking them. Each accelerator is a node in that directed graph and the arc between two nodes represents the physical transfer line from one accelerator to another. Particles flow through arcs and spend some time in each node.

In this context, a beam is a path through the directed graph, and the MTG deals with beams in this "PS controls" sense of the word. The *plsedit* program is used by machine

operators to change the supercycle, that is, the sequence of cycles to be repeated over and over again. This program is able to receive input in graphical form and to translate it into what we call a *Beam Coordination Diagram* (BCD), which is just data that will allow the MTG to generate telegrams. In fact, things are a bit more complicated. The complete sequence without much detail would look like this:

- First of all, an operator "draws" the supercycles with *plsedit*.
- Another program then uses ORACLE-stored data to merge the BCD generated by plsedit with information for the AD, which is a loosely coupled machine[22]. After doing this, it sends the merged BCD to the MTG's main task, a real-time task running on the DSC where the MTG[23] is sitting.
- With this information, the real-time task automatically generates a pseudo-assembly code program and downloads it to the MTG's internal processor every 1.2 seconds.
- The MTG card then starts interpreting the downloaded code, which is just information on what events to distribute and when to send them.

These operations need an external timing reference, so that the whole timing system agrees on what 1.2 second intervals to take. The general reference is a 10 MHz clock sent by a satellite and received by a GPS module. This 10 MHz is divided first to obtain a 1 kHz clock and then again to get the 1.2 second clock.

5.3. The MTG and TG8 modules

So how is all this handled from the hardware side? As we already stated, the timing system is based on both the MTG and the TG8 VME modules. We have also talked about how the MTG works: a real time task running on the DSC calculates the MTG's local processor program and downloads it to the MTG every 1.2 seconds. The MTG's role is to execute this program, that is to read these 32-bit words, serialize them and send them through the timing multi-drop network. The serialization is done by a second on-board controller, Motorola's 68HC11.

On the TG8 side, there is also a micro-controller with firmware running on it, but in this case the firmware is only downloaded at startup and keeps running until the module is reset. Its role is to receive the messages sent by the MTG. Encoded in these messages, one finds all kinds of timing information used to run the accelerator. The TG8 has to decode the 32-bit Manchester-encoded messages and use them to provide front-end pulses or VME interrupts accordingly. This is done in several phases:

[22] The main consequence of this for the timing system is that the length of the supercycles in the AD is not constant.
[23] In fact, there are currently four MTGs for security purposes, but we will limit the description to one MTG for clarity.

1. A Xilinx XC-3030 FPGA and an NS DP8343 chip are used to receive the serially transmitted 32 bits and to check for transmission errors. Once they have done this, the XC-3030 interrupts the on-board processor (an MC-68332) to transfer the received message in a double 16-bit parallel access. Some of the messages (called C-train messages) are distributed each millisecond by the MTG. After detecting this kind of message, the XC-3030 also generates an output pulse, called the 1 ms pulse. This can be used for equipment synchronization (through a front panel connection) and also as an internal clock for counters.
2. Once the message has arrived at the MC-68332 micro-controller, the information is treated by the firmware and control signals are sent to the counters accordingly.
3. The counters are embedded in two Xilinx XC-4005-6 gate arrays. These chips receive control signals from the MC-68332 and generate the appropriate output pulses in eight pins corresponding to the outputs of the eight counters. The first XC-4005-6 contains four down counters (1 to 4) and the second one contains the four remaining down counters (5 though 8).

As a result of this architecture, the TG8 is able to produce pulses and VME interrupts that take any event as a reference and introduce a programmable delay using the counters. The reference that triggers the counters can be either an event decoded from the timing cable or an externally supplied start pulse.

5.4. Timing information access through software libraries

When a programmer calls an equipment module from an application program, she is supposed to give the PLS line she wants to work with. This, of course is due to the fact that the whole complex works in PPM. These cryptic letters stand for Pulse to Pulse Modulation, i.e. the accelerator's parameters change (are "modulated") from one cycle to another (from one "pulse" to another).

Behind these Equipment Module calls is hidden a set of functions that allow the EMs to know at each moment what the current cycle is, what the next cycle will be and lots of other timing-related information. These C-language functions are grouped in the TGM library, also called the Telegram Access Library.

Most of the time, however, a programmer will not have to deal with them directly, since the GM library already uses them to handle EM calls transparently (have a look back at figure 8 if you don't remember where the GM library stands in the hierarchy). The timing system is thus fully integrated in the EM philosophy and contributes to it.

6. High level software

All software activities that use the low-level services described in chapters 3, 4 and 5 are labeled “high level software”. These include of course the development of application programs, be it on a workstation under the UNIX operating system or on a PC under Windows. This chapter is devoted to a clear and concise description of the high level software development process in the PS context.

The first responsibility of the CO group is to provide the developer with tools to do her job. On the UNIX side, this support includes a set of libraries and development environments to make her life easier. Also, in order to answer the day-to-day operation requirements, some *generic applications* have been developed by the group. We will briefly describe their role and their user interface.

Although the UNIX way is the standard and supported way to access the equipment from an application, it’s also possible to develop and run non-critical programs on a PC platform. The equipment access functions are provided by a service called the “passerelle” that links both the PC and the UNIX worlds through a server. A short description of its principle of operation is given in section 6.3.

Finally, a fairly complete description of an application program is given to illustrate how all these concepts are used in practice. The *AD Cycle Editor* has been chosen because it is an example of how far this architecture can be pushed to control virtually a whole accelerator from one application.

6.1. Support for application developers: the EQP library, UIMX and MOTIF

As we said, application developers can use a variety of libraries that let them concentrate on the design of their application instead of worrying about unimportant details. Figure 10 gives a glimpse of how this is achieved. The application program code sits in the center and is surrounded by libraries that let it communicate to the user on one side and to the control system and the archives & references on the other side.

All the Graphical User Interface (GUI) is done using the MOTIF library plus a set of four PS-specific widgets: Graph, Plot, Digit and Wheel-switch. These allow an easy implementation of concepts that are used frequently, such as graphically representing an array of points as a 2-dimensional plot. Besides, the whole process of building an application is simplified by the use of the *Frame* production environment. The Frame consists basically of three components:

- A GUI builder (UIMX), which has been customized to local needs: local widgets, make files, etc.
- An empty application frame, containing all the standardized components of the application interface. The programmer must add her own components and remove any standard components which are not required.
- A set of utility functions for the most widely used commands (e.g. displaying a warning message).

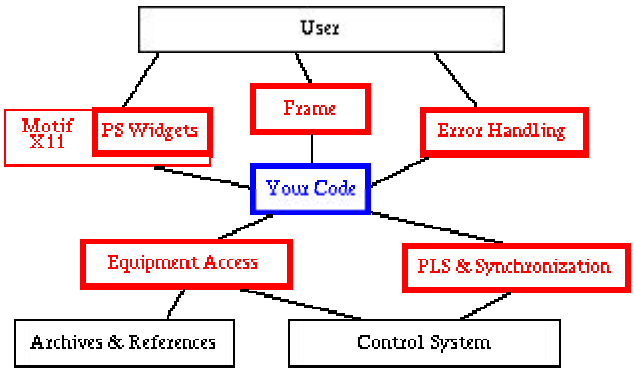

Figure 10. PS libraries for application program developers.

The Equipment Access (EQP) library is responsible for communicating with the GM library via the RPC protocol, while the PLS & Synchronization (PPM) library is used as a friendly interface to the low-level TGM library. The PPM library also provides additional services such as the possibility of subscribing to a certain PLS line, i.e. synchronizing one's code to the arrival of certain timing events. This service is also included in the Frame.

Finally, error handling is done via the Err library. All errors detected inside a library or an application must be reported by means of the *ErrLog* function. The error handling packages can be configured to send messages to an *Error Logger* server task. This is done by setting the environment variable *ERR_HOSTNAME* in the environment of the task[24]. The library adds host and task identifiers as well as a time stamp into the messages, and these are then sent to a file. The *Error Viewer* program can also be used to monitor these messages on-line.

[24] This configuration is active on every operation workstation.

6.2. The generic applications

While special applications are generally developed by anyone in the Operations group, the Controls Group is committed to provide a basic set of programs that are used all around the complex and which are central to the operation of the machines. These include the Console Manager, the Alarms program, the knobs server and the Error Viewer.

Mastering some operation concepts is necessary before discussing any generic application. From an operator's point of view, the PS complex is divided into machines, processes and working sets.

- A *working set* is a named software entity describing a sub-system, like the CPS Radio-Frequency, or a part of the machine, like the PSB to CPS transfer line. The display of the CPS "Positrons Injection" working-set can be seen in figure 11. Notice the name of the working set on top (SD92-E+_INJECT) and the different equipment names with some selected properties that we want to monitor for each one of them. These equipment names for each working set and properties for each class of equipment are defined using the *console* Oracle Forms application.

- *Processes* are defined in order to group together working sets and programs related to the same operation process, like "All Injections". A working set or program can belong to one process only, and each working set must belong to one process. There are two categories of processes:
    - "OP" processes are those presented by default to the operator.
    - "SPEC" processes must be explicitly selected before the associated working set and programs are made available. This is used to separate "specialist" working sets and programs (e.g. vacuum, RF...) from the default operation environment. "SPEC" process are specified by means of the following naming convention: the 1st letter of their name is ":" (e.g. ":VAC-SPEC").

- The complex is divided into a set of OP *machines*. Each machine describes a complete operation environment for the end user and is identified by an acronym, such as PSB for the Booster. Working sets as well as programs for each machine are defined in the configuration database using the *console* form.

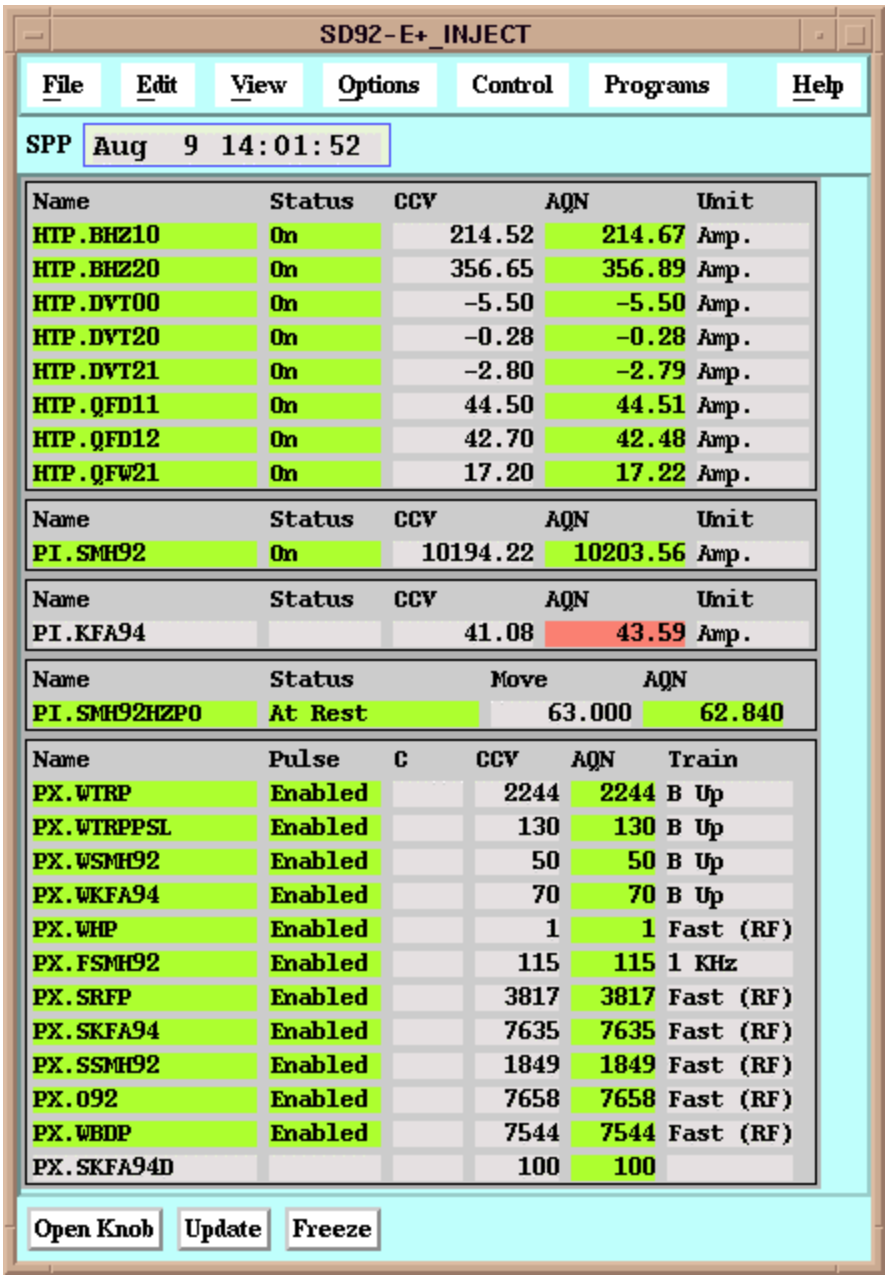

Figure 11. A working set as seen by an operator on a workstation.

Having acquired these basic operation concepts, we can go on to discuss the generic applications. The *Console Manager* is launched automatically whenever an operator logs in. The login name tells the system which is the machine to be controlled by the operator, so the Console Manager is configured according to the defined set of working sets and application programs for that given OP machine. Figure 12 shows the start-up screen for this application.

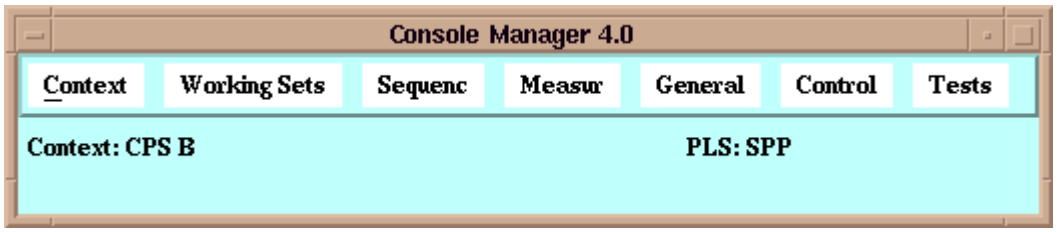

Figure 12. The Console Manager application.

Basically, the role of the Console Manager is to control the execution of all other interactive programs used in the operation of the machines. Calling a given working set display will result in a screen such as that of figure 11, which is more than a mere visualization of some parameters for a predefined set of modules. The data are interpreted and different colors are used to signal occurrences of some predefined conditions, such as big differences between the control and acquisition values. If the operator wishes to modify control values, she can do it using the knobs server. A knob can be called by clicking on any element on a working set display. A window such as that of figure 13 will pop out and the operator will be able to change a control value by clicking on the appropriate arrow buttons or pull-down menu.

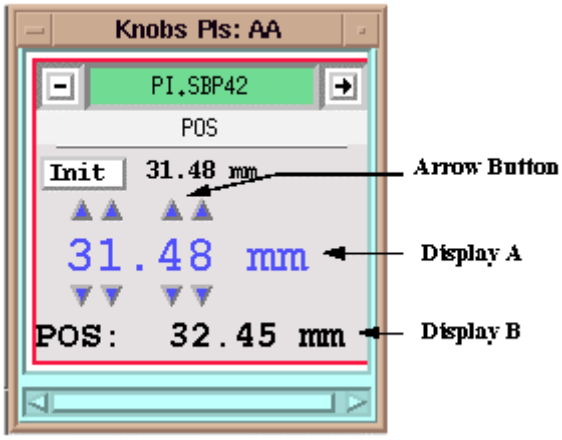

Figure 13. A knob for controlling the POS property of an equipment called PI.SBP42.

The other menus in the Console Manager are used to launch specific programs. These have to be declared using the *console* Oracle Form application in order to be accessible from the Console Manager.

An additional feature of the Console Manager is the possibility of storing current values in a database and recalling them at some later time. This allows the operator to change to a complete different configuration of the machine at the click of a button.

The two other generic programs currently in use are the Alarms program and the Error viewer. The former, which will be described in section 7.2., involves monitoring of a special property of some Equipment Modules called “alarm” and is used to make sure that a given piece of equipment is behaving properly. On the other hand, the *Error Viewer* offers a human-readable representation of error values returned by programs.

6.3. The “passerelle” approach

The PS Control System Access Gateway, known as the *Passerelle*, which provides a controlled access from the PC Office Network to the control system was developed in the early 90s. Initially it served the Windows 3.1 clients, but when Windows 95 was introduced at CERN, the PC environment became an attractive platform for tests in combination with some Windows-based tools like Excel and Visual Basic.

The *passerelle* is a software gateway running several processes on a IBM AIX workstation. It behaves as a bridge between the PS control system and the Windows 3.1 & 95 client platforms.

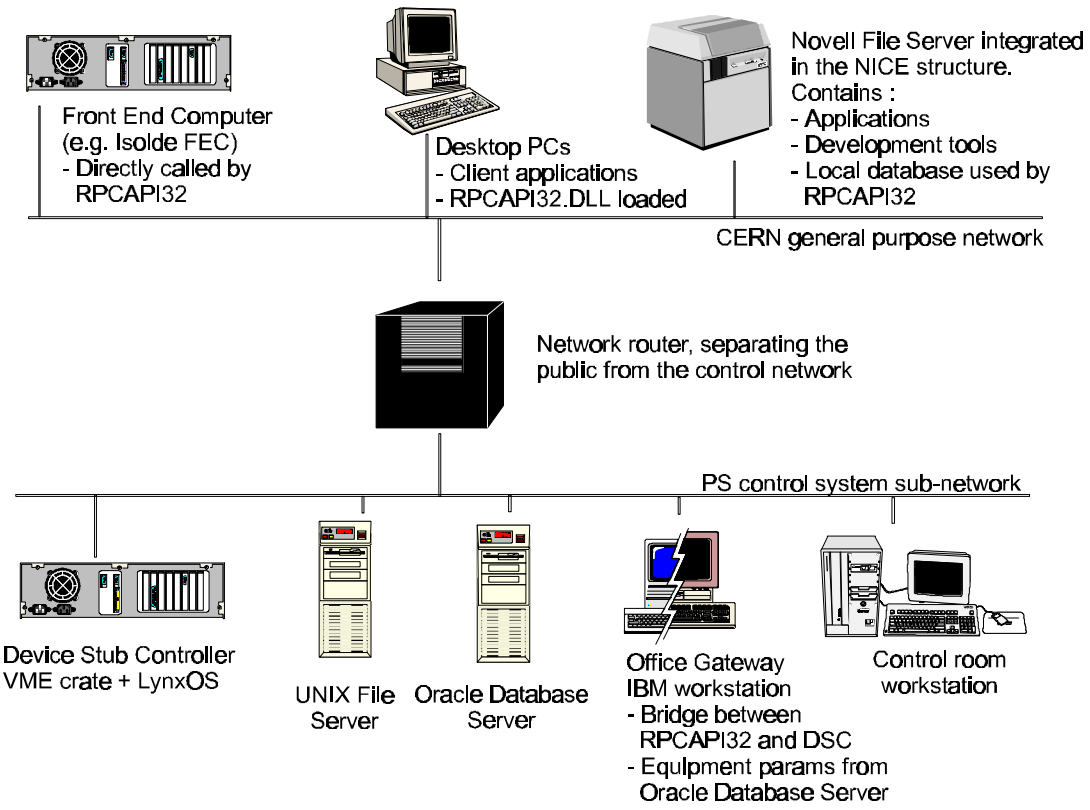

Figure 14. The passerelle architecture.

Figure 14 represents a schematic view of the *passerelle* architecture. Very briefly, it is based on a client-server model where the server is the IBM Gateway and the clients are the PCs sitting on the offices. These clients can run programs that call a Dynamically Linked Library (DLL) which handles the requests to the server through the network. The actual equipment access is thus done in the standard way by the workstation, but the passerelle interface hides most complexities to the user.

The passerelle also allows the user to establish “hot links”, that is on-line monitoring of a certain equipment property that may change every cycle. Another feature is access control: in order to have write access to equipment properties, a user must get registered in a database.

Although the Windows-based approach seems attractive to the occasional user, the CO Group has chosen the UNIX + Lynx-OS way to ensure reliability and to provide a standard frame to develop software that makes maintenance easier. However, the passerelle remains unequalled for quick tests and small non-critical applications involving equipment access, using Excel and Visual Basic programs.

6.4. An application example: the AD Cycle Editor

The goal of any control system is to end up providing a friendly and powerful environment that operators can use to maximize the productivity of some machine. In the PS complex, being productive means to provide a particle beam with a set of required characteristics as easily as possible. Ideally, an operator should have a very Physics-oriented view of the machine, and this should be reflected on the tools she uses to control it.

The AD Cycle Editor, conceived to control the new Antiproton Decelerator, is an excellent example of how application developers can use the underlying interface provided by the Equipment Modules and the associated development libraries to hide all complexities and present the operator with a clear view of physical machine parameters and an easy way to modify them.

What an operator has in mind when trying to visualize the global setup of an accelerator looks roughly as figure 15. The horizontal axis is time and the vertical axis can be interpreted as the current flowing through power converters or as the momentum of the particles in the accelerator. The AD Cycle Editor allows the operator to pass from one view to the other.

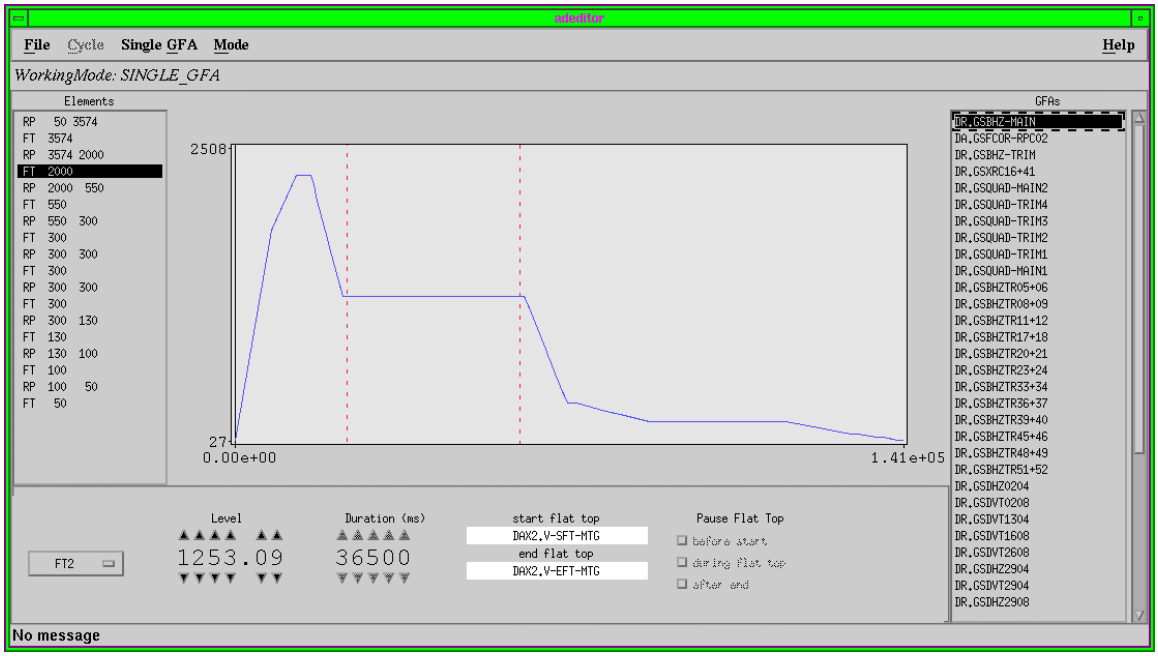

Figure 15. The main screen of the AD Cycle Editor.

The magnetic cycle is basically made up of ramps and flat-tops. The energy of particles stays the same during flat-tops and it increases (decreases) in ascending (descending) ramps. In the case of the AD, particles are supposed to be injected at a relatively high energy from the PS during the first flat-top and the end-product should be a beam with momentum as low as 100 MeV/c that will allow physicists to explore antimatter at conveniently low energies. To achieve this, one cannot decrease momentum linearly in a ramp because the beam has a tendency to disperse at low energies. Therefore, it was decided to insert "cooling" flat-tops to re-adjust beam dispersion at some intermediate energies. The techniques used to achieve this are called *stochastic cooling* and *electron cooling*, and they involve the use of external magnetic kicks and electrons to make the beam converge to a narrow section.

We can now appreciate what the AD Cycle Editor has done for us. By clicking and dragging one's mouse, flat-tops and ramps can be inserted graphically at times and energies to be decided by the operator. The very complex consequences this has on timing and power control equipment are all handled by the application, so that a perfect synchronization is preserved all around the accelerator.

It is difficult to overstress the importance of the development process. In the case of the AD Cycle Editor, the starting step was to come up with a complete model of the accelerator that was subsequently divided into different subsystems: Radio Frequency, Power, Stochastic Cooling, Electron Cooling, timing, etc.

A second task was to ask the users what operations they would like to perform on a cycle. These included adding and removing flat-tops and ramps, modifying energies, saving reference cycles and so on. The consequences of each of these "operator actions" on every subsystem were evaluated and translated into functions that accessed the equipment in the standard way, that is through the EQP library. The dialog with the timing system is handled with special tables sitting on the ubiquitous Oracle database and by direct communication with the MTG. In fact, the AD Cycle Editor goes as far as generating the whole set of telegrams and simple timing events for the AD machine and transferring them to the MTG module. The MTG then synchronizes all pieces of equipment by sending these messages through the timing multi-drop network. The whole process is of course transparent to the user.

The advantage of this approach is to provide an extremely intuitive interface for the operators and an easy way to modify vast amounts of configuration data at the touch of a button and with no risk of incoherence among the different pieces of equipment. On the negative side, some could argue that one cannot diagnose individual equipment failures with this application but the truth is that other tools exist for doing that. These tools are used primarily by the CO Group Exploitation Section, to which the next chapter is devoted.

7. Making it all work together: exploitation

After having devoted a great part of this document to the technical aspects involved in our control system, the time has come to talk about how all this complexity is handled from a human viewpoint. As we said, reliability and low down-time are the key words in the PS Controls (and in any control system for that matter). To achieve these goals, a team of five people form a picket service. Any user can call this service 24 hours a day, any day of the week, to report problems related to controls.

As we will see, exploitation people have a deep knowledge of the control system and are able to understand the user requirements. These two aspects are of paramount importance if one wants to provide a useful and reliable service.

From an administrative point of view, the Exploitation Section is the only official entry point for any external request concerning the controls system. This is essential to avoid any incoherence in the decisions taken by the group as a whole.

In this chapter, we start with a description of these organizational aspects and then we go on to discuss some of the most important diagnostic tools in use at the moment. The feeling we'd like to convey is that exploitation specialists are not only concerned with maintenance, but they use their broad skills to develop diagnostic tools, participate in the group's new projects and give their educated opinions about any group decision.

7.1. Organization: the piquet system and the new requests entry point

In order to assure a smooth interface between the operators and the Controls Group, a team of five specialists takes turns to answer the calls of users confronted with emergency problems, either by in situ interventions or by remote access interventions from terminals, workstations or PCs. Each one of them is on standby duty during a week, which means a turnaround of five weeks. This seems to be a good compromise between standby load and maintaining experience and knowledge. During the rest of their time, which represents more than 50% of their working time, they participate in new developments or enhancement maintenance. As we said, their participation in new developments is very appreciated due to their maintenance experience and their excellent overall knowledge.

Besides emergency repairs, the exploitation tasks include upgrading, extensions and development of diagnostics tools. The section is also our interface to the end-user. In this context, exploitation specialists play three roles:

- Explaining system capacities and proper use of the control system to the end-user.
- Participating in the improvement of the system's performance.
- Evaluating the feasibility of new ideas thanks to their vast knowledge of the control system.

As a new requests entry point, the section keeps operators' wishes and controls' offers from getting lost in a mess of incoherent actions that would not satisfy the end-user. This means that the team must have a clear understanding of the user requirements and a general and detailed knowledge of the control system.

The picket system and the new requests entry point form the backbone of our exploitation strategy. The results so far are very positive, and the old prejudice that saw the exploitation specialist's task as pure maintenance has vanished to be replaced by a picture where these people are highly skilled, well trained controls specialists who contribute to the overall group evolution and are involved in all projects.

7.2. Tools for the exploitation: the alarm program and timing layout

The motivation behind the idea of a good set of exploitation tools is evident. When things go wrong, the task of finding where the error comes from can be extremely tricky. Years of experience are necessary to develop an intuitive sense of fault detection, and sometimes intuition alone is just not enough. Our control system is big enough so that the need for general approaches arises. This is also true for problem detection and diagnostics.

One of the main sources of information is the web interface to the Oracle tables. There we find all equipment properties and related information such as the control and acquisition interrupts they are linked to. Among the vast amount of specific diagnostics programs, we could single out the following due to their extensive use in the PS-CO Exploitation section:

- The *alarm* program is used to monitor the running state of all pieces of equipment on-line. How this is done will be clear in a minute.

- The *eqpinfo* program provides a friendly interface to visualize equipment properties and to modify them.

- The *video* program is used to follow the different cycles as they are executed in the machines. It uses the current decoded telegram and displays this information graphically.

- The *test_tgm* and *tg8test* programs are low-level tools to explore timing information related to a given DSC.

- The *SchemaDraw* application provides a friendly way to visualize relationships among different pieces of timing equipment and is interfaced to other programs that allow the monitoring and modification of their properties.

To get a taste of the kind of development effort involved here, let's look more closely at two of these applications. The *alarm* program, whose initial screen is depicted in figure 16, allows the user to view the state of a set of pieces of equipment by monitoring the ALARM property for each one. This property is included in many Equipment classes and the associated real-time tasks are supposed to assign specific values to it depending on a set of predefined malfunction modes.

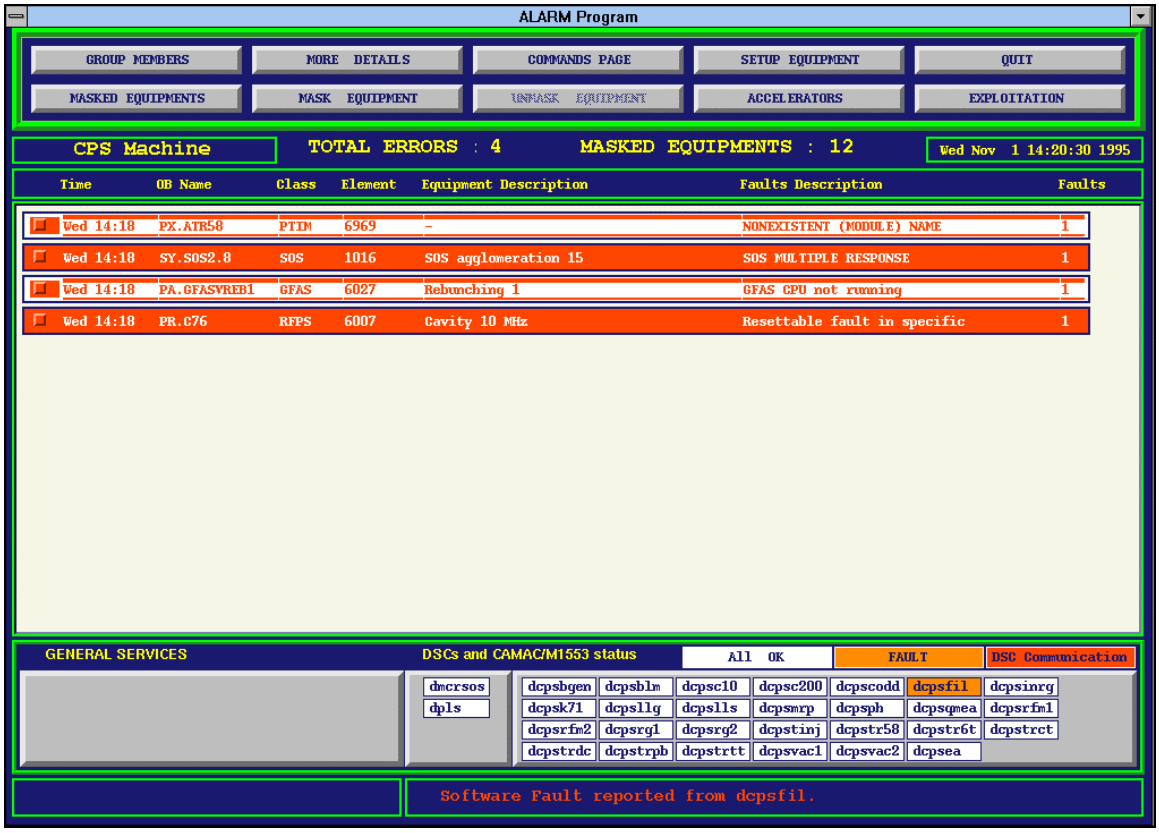

Figure 16. Screen shot from the alarm program.

The program can be launched from the Console Manager's General Menu. The pieces of equipment monitored by the program are those associated with the machine for which the Console Manager was launched. This association is done through a dedicated hidden

working set called <MACHINE>:ALARMS. The alarm display zone contains a list of the devices for which the property ALARM is not zero. Besides, a sequence of setup instructions to initialize some device can be launched via the “Setup equipment” button.

On the other hand, the *SchemaDraw* application is a graphical interface that lets the user visualize intuitively the relationships among different pieces of equipment and eventually modify parameters to tune up performance.

Figure 17 is a screen shot of this application where a set of injection-related timing devices can be seen for the CPS machine. Clicking on a TG8, for instance, will result in a screen where all parameters for that TG8 are displayed and where the user can change values for every read/write property.

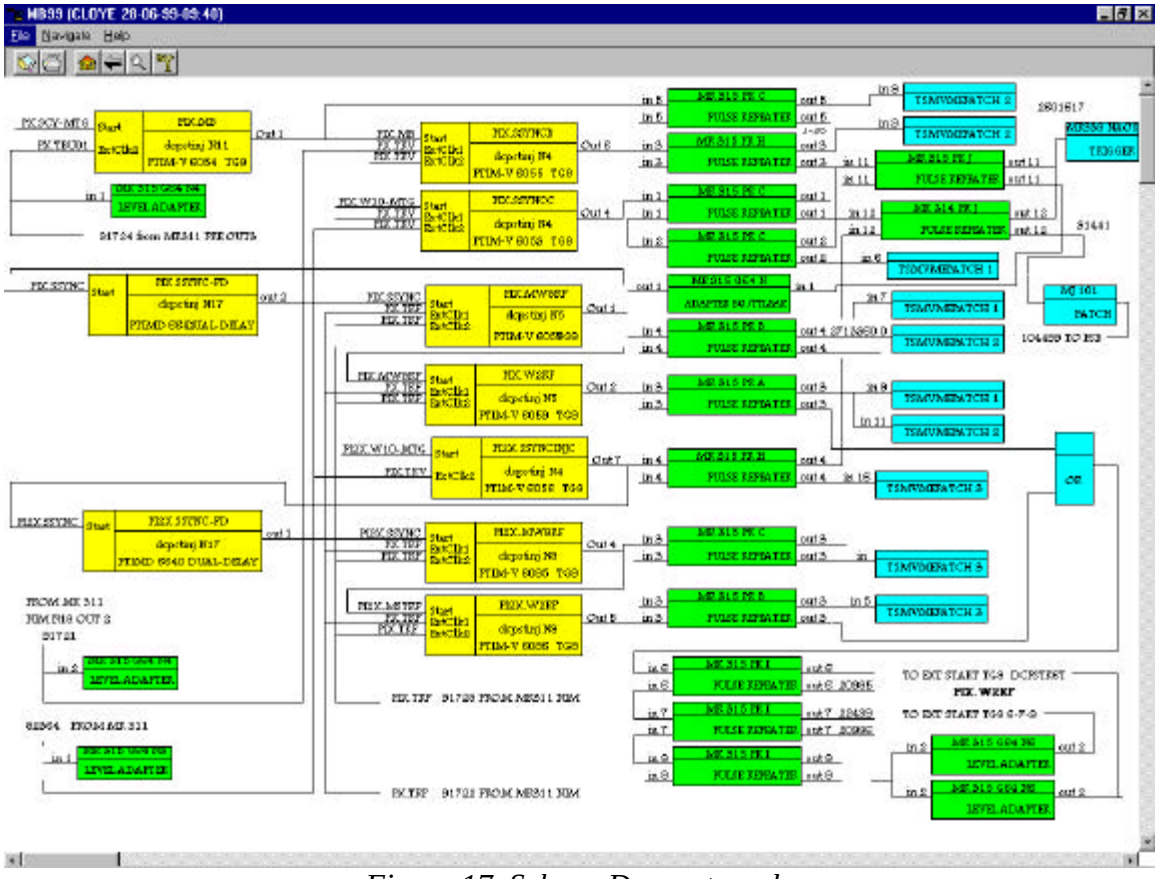

Figure 17. SchemaDraw at work.

The *SchemaDraw* application is built on top of the passerelle architecture and uses all of its services, including “hot links”, i.e. automatic refreshing of values on the screen for every cycle.

8. Current projects in PS-CO

Many of the current projects in the PS-CO group were launched in the context of the PS/SL Convergence Project, an ongoing effort to share and unify views about accelerator controls lead by the two accelerator divisions at CERN. Smaller projects exist in the group but their scope is limited to few people and it would take lots of space to describe them individually. Since the purpose of this document is to give only an overview of the PS-CO activities, three major projects have been chosen to illustrate current trends.

The Middleware project is intended to replace RPC and to provide a new set of services based on the object-oriented paradigm applied to inter-machine communications. The Java API[25] for accelerators control will give application programmers the opportunity to take advantage of the many features of Java: simple programs, built-in graphical capabilities, object oriented design and friendly development tools among others. Finally, ABS (Automated Beam Steering and Shaping) is just what the name suggests: a generic tool that will allow operators to automatically correct beam parameters without worrying about low-level details.

These three projects are all exciting opportunities to use the latest technology in software and communications. Besides, they are also innovative in that many people from different parts of CERN and even from different laboratories around the world have agreed to cooperate to develop solutions to their common problems. But what are these problems are how are they being solved? Let's find out.

8.1. The Middleware project

The RPC approach currently used in the PS has proved to be a reliable means for communications among different computers. However, RPC has several important limitations. Communications are synchronous and blocking, meaning that a process that sends an RPC request is frozen until the arrival of the response. Besides, if one is interested in monitoring the value of a certain remote variable continuously, the only way to do it is by sending requests at regular time intervals, a technique called *polling*, thus wasting bandwidth with requests that result in an unchanged value.

The Middleware project was launched to give an answer to these problems. Very simplistically, it consists of a software layer placed somewhere between the high-level API and the different pieces of equipment. A self-descriptive hierarchic view of the different layers is depicted in figure 18.

[25] Application Program Interface.

As suggested in the figure, the Middleware layer should allow programmers to use the *publish/subscribe mechanism*. This means that a program can be "awakened" by a predefined set of events concerning variations in a remote variable. The publish/subscribe mechanism therefore provides an elegant alternative to polling.

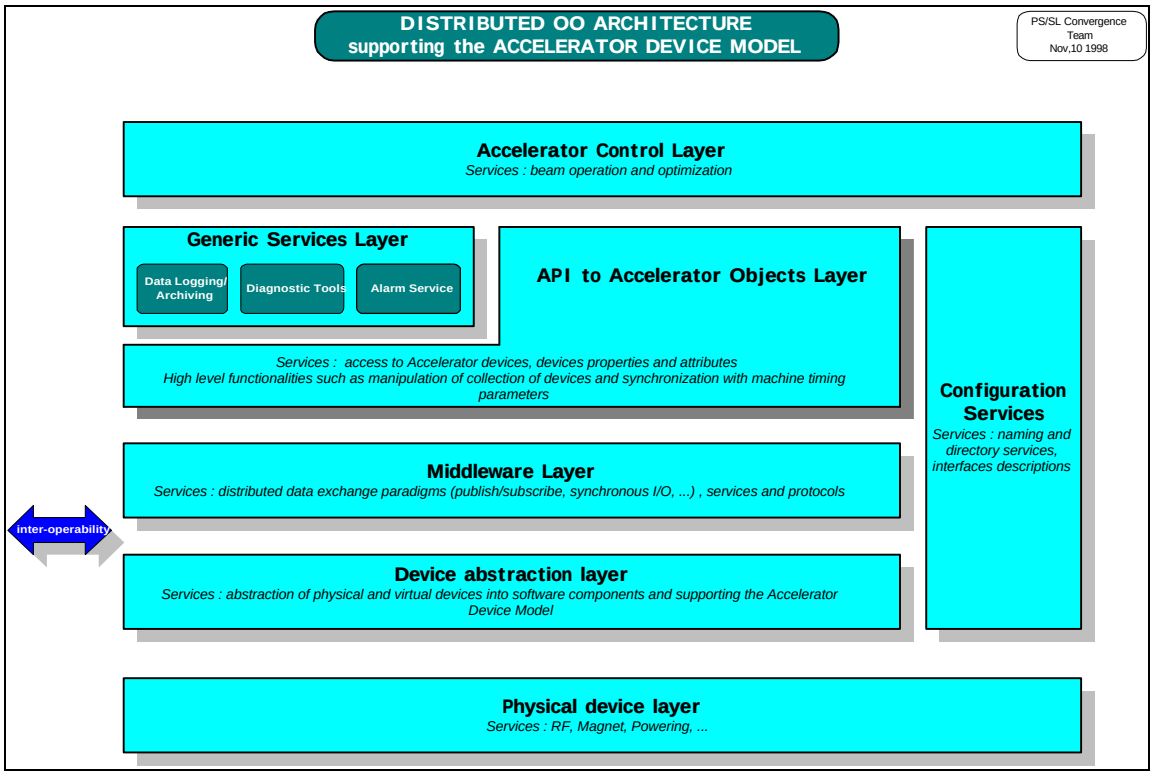

Figure 18. PS/SL distributed object-oriented software architecture.

Another basic requirement for the project is to support the common PS/SL accelerator device model. This model defines how accelerator components will be viewed and accessed from the software application level. On the other side, the middleware software will have to deliver a distributed controls architecture that also supports the technical specification of the Java API.

Yet another requirement is to be able to communicate with the now ubiquitous PLC-based industrial control systems. These are mostly based on OPC (OLE[26] for Process Control), a Microsoft standard, so the final solution will involve some kind of OPC interface.

The project has just gone through the requirements phase and the implementation will surely be based on one of the commercially available object-oriented packages that allow

[26] Object Linking and Embedding. Microsoft's robust means of integrating applications.

access to remote objects' methods. These include CORBA and Java RMI[27], but in any case a big effort for customizing these solutions to our control system is necessary.

8.2. The Java API for equipment access

The best way to place the Java API project in context is to see how it originated and how it developed through time. The force driving these project's beginnings was to unify the way equipment access was handled in the PS and SL divisions. While the PS application programmers saw what we call a "narrow interface", that is the very standardized and uniform equipment access approach we described in chapter 4, an application programmer in the SL Division had to know the pieces of equipment and their intricacies. Both ways of handling equipment access have their pros and cons, but it became clear that having a uniform approach for both divisions could only do good.

What the Java API intends to do is very schematically represented in figure 19. The specified API will be implemented by means of Java libraries (packages) that will rely on external services.

- Directory services, based on databases will provide the description of the control-system entities.
- I/O services will implement access to equipment parameters.

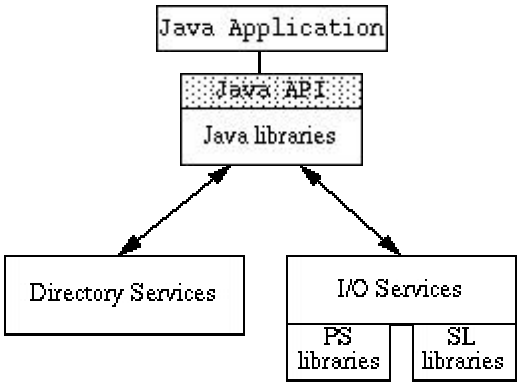

Figure 19. The Java API software architecture.

The current version of the Java API is based on CDEV, an object-oriented interface built on top of the EPICS control architecture. EPICS is a system for controlling High Energy Physics facilities that was developed by a collaboration of big American laboratories, such as the Los Alamos and Argonne National Laboratories. It specifies front-end (VME form factor with the VxWorks Operating System) as well as high-level software.

[27] Remote Method Invocation. A program can tell another program to execute a routine in some distant machine.

CDEV was developed at Jefferson Lab as a C++ API for application programmers. Pieces of equipment were accessed through objects of a class called "device". After a first Java version was released in the US, a CERN-TJNAF[28] collaboration developed version 2 of the Java-based CDEV API, which already implemented the architecture of figure 19.

As we already said, there are two services that the Java API uses. Directory services give programmers information about equipment names, their nature, location and so on. Configuration data for the PS are currently stored in Oracle tables but these have to be translated into a standard format and stored in the directory service tables, together with SL data. Once this is done, JDBC[29] routines are used to access the data in the API.

On the other hand, I/O[30] services perform equipment access, much as the EQP library does now. There are many ways to implement these services. One could just call the C++ EQP routines via JNI (Java Native Interface), a package that allows interfacing between Java and C/C++ code. Another way would be to implement a Java RPC client on the workstation side and a standard Sun RPC server on the DSC. Yet a third possibility is the use of the passerelle as an intermediate server between workstations and DSCs. All of these ways are being explored. The Middleware Project will be of major importance for these I/O services since it will act as a software entity between the Java I/O routines and the front-end computers. New I/O services such as the data subscription mechanism will then become available.

8.3. Automated Beam Steering and Shaping (ABS)

The ABS project is meant to provide a homogeneous correction environment for operators. In the past, every application program provided tools to correct beam parameters such as position and shape. With the advent of more strict requirements for the LHC era, a new solution had to be designed in order to take the same approach everywhere and to achieve the required precision.

The new way of doing things involves a generic correction program with an interface that lets it communicate with the outside world. This program generally receives acquisition data from measurement programs and sends correction values to the equipment. The correction algorithm needs to know machine parameters in order to compute correction values. In fact, a whole description of the accelerators is necessary, including characteristic curves for every magnet, physical definition of every active component and so on. This is the perfect job for a database!

[28] Thomas Jefferson National Accelerator Facility.
[29] Java Database Connectivity.
[30] Input/Output.

The Oracle tables contained in the ABS database include information about optics, layout and magnets that is used by the generic correction program when calling the specific routines that find a solution. These routines are implemented using Mathematica, a mathematical package that allows easy resolution of the linear systems used to model the optical elements behavior around a nominal set of values.

The measurement program calls ABS through a set of specific libraries. After the call, a screen such as that in figure 20 pops up. The user is then prompted for information about the kind of correction to perform. Using this information and data from the Oracle tables, control values are calculated and sent to the correction equipment.

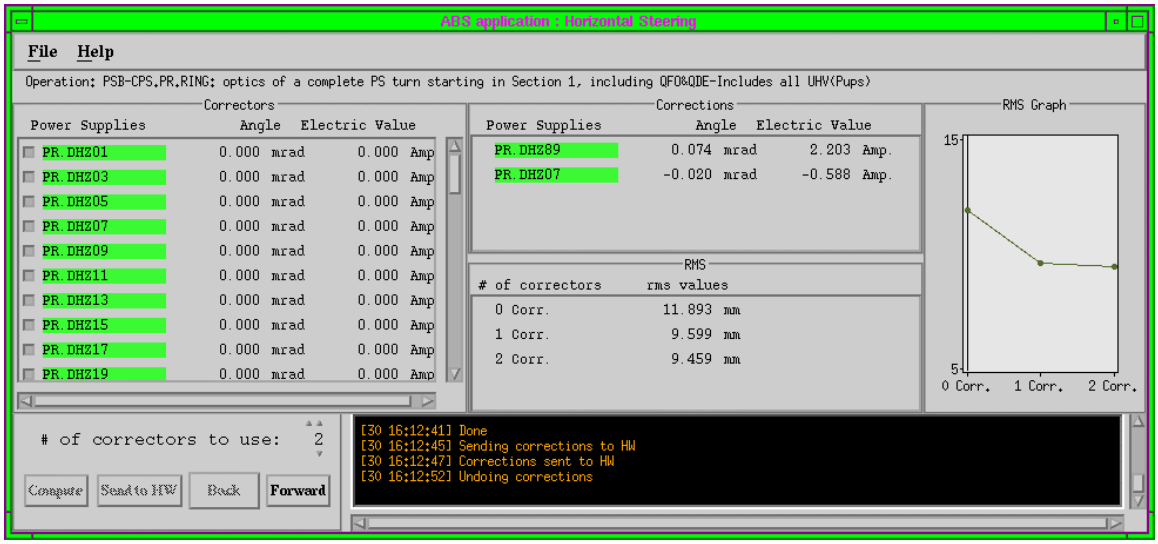

Figure 20. An example of a typical ABS screen.

Every active element in the machines is characterized by a transfer matrix that relates the output position and angle for a beam to its position and angle at the input. In theory, multiplying all transfer matrices in a subsystem will result in a matrix that represents the input-output relationship for the whole subsystem. The problem to be solved in ABS is just the opposite: we know the positions we want to obtain and we wish to compute the suitable angles for attaining those positions. Once we know them, we will be able to calculate the currents that make the magnets behave in such a way and to send the calculated values to the power converters concerned.

In theory (again), if a the matrix relation $\overrightarrow{position} = M \bullet \overrightarrow{correction}$ is known, we should be able to calculate the correction vector as the inverse of matrix M times the position vector. Unfortunately, the inverse of M does not exist in our context (M is not even a square matrix), so we assume that the system behaves linearly around a given working point and solve the corresponding linear system with Mathematica. The results so far are very promising and future extensions are foreseen that will enable ABS to become a very important general purpose control tool.

9. Conclusions and future projects

This is the end of our journey around our control system. During the trip, we explored how this intricate set of machines, electronic devices and software cooperate to control and synchronize the accelerators. Our final product is just a particle beam with certain intensity and shape characteristics, but now we can appreciate a part of the enormous amount of work hidden behind: that concerning the controls system.

If I had to mention only one driving force that motivates the kind of control we do, it would surely be the necessity to standardize all levels in the hierarchy going from hardware to applications software in order to maintain a very large system with limited manpower. In this context, there are also emerging efforts to improve the service to our "clients", mainly the OP group. We saw how the Middleware, the Java API and the ABS projects are under way and some of these new initiatives are already giving the desired results.

We didn't want to burden the reader with lots of uninteresting pages of details about the different activities in the PS-CO group (plus in fact we didn't have that much time ourselves!) but it turns out that there is a really friendly atmosphere in the group, and anyone interested in a specific topic will have no problem to find our specialist and discuss deeply over coffee.

We are extremely lucky to be in a highly active engineering environment. New requests come every year, either from the Physics community or from internal needs. The years to come will be no exception: we will welcome some new members to our beam family, including neutrons, and a new Central Beam and Cycle Management System (CBCM) will extend our timing system by linking it to the SPS accelerator in the SL Division.

The LHC requirements will grow in number and we will surely be confronted with new field-buses to control power converters and Programmable Logic Controllers (PLCs), such as ProfiBus and WorldFip. Also, as the LHC era approaches, the PS/SL Convergence projects will be taken into account for implementations in the new accelerator's control system. So as you can see there is lots of interesting stuff coming up. Stay tuned!

Appendix A. References

While there is no better reference than actually going to talk with the person who is responsible for a given area or project, there is a great amount of information both in the web and in published notes. The starting point for all web-based information is the group's official home page: http://psas01.cern.ch/Welcome.html.

A short guide for finding further information about each chapter follows:

Chapters 1 and 2

Go and have coffee with any of the group's members to know more about our role in the PS complex.

Chapter 3

A good starting point is http://psas01.cern.ch/user/Welcome.html, and for the more technical-oriented there is lots of information at http://psas01.cern.ch/sys/Welcome.html.

Chapter 4

A great introduction to the VME bus and its specification is:
Peterson, Wade D., The VMEbus Handbook, VITA, 1996.

The list of standard VME modules in the PS control system, along with a brief description (and prices!) :
Heinze, W., Standard VME modules in the PS Control System, PS-CO Note 99-04 (tech.)

Configuration management aspects are described in a PS-CO note:
Gagnaire, A., Introduction to DSC configuration management, PS-CO Note 93-80 (tech.)

The bible for equipment module programmers:
Sicard, C. H., Cuperus, J., Walter, O., Control Module Handbook, PS-CO Note 95-01
Also available at web address:
http://psas01.cern.ch/gm/FrontEndDoc/HandBook/Index.html

A very useful presentation on style recommendations to write real-time tasks:
http://psas01.cern.ch/gm/FrontEndDoc/RTTDoc/index.htm

The database is fully documented at http://psas01.cern.ch/db/Welcome.html.

Chapter 5

All sorts of information concerning timing:
http://psas01.cern.ch/timing/Welcome.html

This includes links to all relevant notes:
http://srv1ps.cern.ch/psco/mtg/notes/docgen/welcome.htm
Bau, J. C., Lewis J., MTG 99: comprendre son comportement, PS-CO Note 99-12 (tech.)

and to the MTG homepage: http://srv1ps.cern.ch/psco/mtg/

Chapter 6

The best address to complement the text:
http://psas01.cern.ch/console/Welcome.html (operator interface page)

The passerelle architecture is very well documented:
Deloose, I., Simultaneous access to the Controls of the PS & SL machines from the Windows 95 and NT Platforms via PS & SL passerelles, PS-CO Note 98-33 (tech.)

Concerning the AD Cycle Editor, PS-OP notes 99-11, 99-12 and 99-13 are a specification and implementation, a user manual and a system overview respectively.

Chapter 7

All kinds of exploitation information in the section's homepage:
http://psas01.cern.ch/expl/Welcome.html, including a link to exploitation tools documentation: http://psas01.cern.ch/expl/tools/Welcome.html.

The user's guide for SchemaDraw is:
Deloose, I., 'SchemaDraw' An MS-Windows based application to design and browse technical and functional diagrams, PS-CO Note 95-63.

Chapter 8

All three projects have web pages with information about specifications, milestones, meeting minutes and so on:

http://hpslweb.cern.ch/pssl/projects/middleware/middleware.html for the Middleware.

http://hpslweb.cern.ch/pssl/projects/javapi/javapi.html for the Java API.

http://psas01.cern.ch/abs/Welcome.html for the ABS.

Appendix B. Index

The idea of a glossary of terms had been suggested at the beginning, but after having made the effort of explaining terms as they appeared in the text, we think it would be redundant (and very difficult) to reproduce all those explanations here. If an explanation for some acronym or PS jargon word is needed, the best way to get it is through this index. After each one of the alphabetically ordered terms, we wrote the relevant page numbers one has to read to get an explanation. Also, page numbers where a figure is important to describe the concept are followed by the letter *f*.