SL-Note 96-45 RF

The New RF Control System for the CERN SPS Accelerator
A detailed presentation

P. Baudrenghien, E. Bracke, H. Marty, J. Molendijk
Y. Pilchen, F. Weierud, U. Wehrle

August 20, 2002

Abstract

The old SPS RF control system designed in 1972 has been replaced completely, hardware and software. The new system has to control both RF equipment conceived during the last 23 years, and future equipment. Using information analysis methods, we derived a model of an RF command and designed a data base accordingly ($ORACLE^{®}$). Information from this database is used for command generation, processing, and also for archiving settings. The advantage is purely generic software, i.e. the same computer code is used for switching on an RF amplifier, or for setting a frequency synthesizer. New equipment is added very simply by entering new records in the data base. Additional features include a reservation scheme whereby a user can take private control of any piece of equipment, a reporting facility notifying the user of the simultaneous control activity by other users on RF equipment, and a capability scheme assigning a level of expertise to each user restricting action on the equipment.

1 Introduction

The SPS control system designed in 1972 consisted of a star network of NORD 100 computers: One central computer, located in the Main Control Room (MCR) and satellite computers for each subsystem. The operators could control the machine via a set of console computers located in the MCR (with some graphic facilities), while the expert of a subsystem could interact via the console located next to the corresponding satellite computer [1]. The programming language was NODAL [2].

In 1990, it was decided to replace the hardware and software based on these NORD 100 computers. This work was motivated by the technical obsolescence and the excessive maintenance costs of this old technology [3]. Working groups were created among the Controls Groups of the PS and SL divisions. It was expected that the equipment groups (such as the RF) take an active part in the development of their own software, while following the guidelines and using the architecture proposed and developed by the Controls Groups [4]. This note presents the resulting control system of the SPS RF.

2 The SPS RF equipment

The SPS is a 7 km long machine accelerating protons, electrons, positrons and heavy ions. The RF equipment consists of the following systems:

200 MHz TWC. 4 cavities at 200 MHz, of the travelling wave type, used for protons and ions acceleration.

200 MHz SWC. 21 cavities at 200 MHz, of the standing wave type, used for leptons acceleration.

100 MHz SWC. 6 cavities at 100 MHz, of the standing wave type, used for leptons acceleration.

352 MHz SUPRA. 4 supraconducting cavities at 352 MHz, used for leptons acceleration.

400 MHz SUPRA. 1 supraconducting cavity at 400 MHz, prototype of the cavity for the future accelerator LHC.

In addition to the above RF power equipment, the control system must also control the following:

Beam Control. Settings and acquisitions: radial loop, phase loop, frequency program. . .

RF Synchronization. Settings and acquisitions for synchronizing the bunch into bucket transfers from CPS into SPS, and from SPS into LEP.

352 MHz SUPRA beam dump and veto field. Acquisition and display of the faults causing a beam dump or a veto of the field in the cavity.

Measurements. Acquisition of data for diagnostics.

3 Architecture

Our control system follows the architecture proposed by the PS and SL Controls Groups [4] [5]. It consists of three layers (see figure 1). The *Control Room Layer* with its UNIX workstations and X-Terminals, is connected via a network (local Ethernet segments bridged to large Token-Rings) to the *Front End Computing Layer*. The Front End process computers are IBM type PCs with LynxOS operating system. They are called *Device Stub Controllers* (DSC) [4]. Each DSC can drive a set of MIL-1553 fieldbuses connecting it to the *Equipment Control Assembly* (ECA) layer. The ECAs are G64 crates with 8 bit microprocessors and AMX based real time operating system. The G64 crates are equipped with Input/Output cards connected to the RF equipment.

Communication between the Control Room Layer and the Front End Computing Layer is done via *Remote Procedure Calls* (RPC) on the network (Client-Server model) [6]. The DSCs communicate with the ECAs via *EQUIP* calls in so-called Command-Response mode: The DSC sends the command to the ECA and waits for the response [7].

4 Distributing the tasks between layers

4.1 Control Room layer

We have developed a Man Machine Interface program (MMI) meant to run on any workstation on the network. Less vulnerable X-Terminals have been installed next to the RF power equipment; and RF experts run the MMI program from there while working on their equipment. The program deals with command generation and display of replies: It guides the user through the generation of a valid command via graphics, menus and selections by the mouse. It displays the replies resulting from that command while it executes, and reports on possible modifications to the equipment triggered by other simultaneous users.

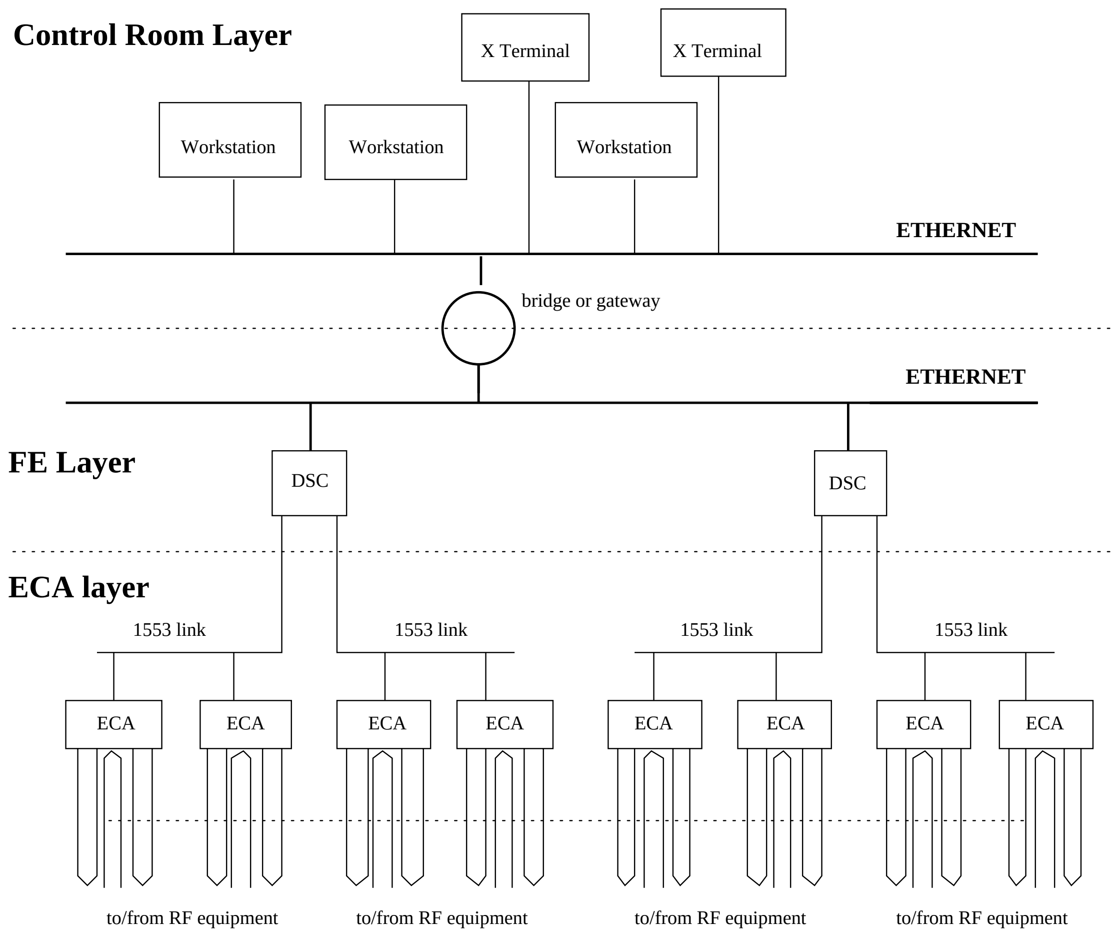

Figure 1: Three layers architecture.

4.2 Front End Computing layer

Refering to figure 1 we see that a given piece of RF equipment (say an amplifier) is connected via ECA's to one DSC. Any command sent to that amplifier will thus go through that DSC. This implies that the following tasks are best implemented at this level:

Filtering. Upon reception of a command, check that the user is allowed to request such an action on the equipment (this depends on the *capability* assigned to that user).

If the equipment has to be *reserved* before sending a command to it, check that the user is the current owner. (Capability and reservations will be discussed in section 6.1).

Sequencing. An action on a piece of RF equipment typically implies the execution of a *sequence* of steps, with the result of one step conditioning the execution of the next one. (For example, switching on an amplifier implies first the switching on of its cooling, then pre-heating of its filament, then the grid voltage, ...). This sequencing is done in the DSC, each step may involve one or several calls to the ECA(s).

Transmitting replies. Replies can be either data requested by the user (read command) or messages notifying the user of the execution of the steps in a sequence. A reply to a user is always the response to a command sent by that user.

Transmitting reports. Reports are messages notifying a user of the execution of a sequence called by another user. It is thus a copy of the replies sent to the user that issued the command. Using these reports, one can be notified of the simultaneous control activity of a portion of the RF equipment. One can ask reports on one or several systems as defined in section 2.

Notifying the Actif server. If a write command is successfull, i.e. it did modify the value of a setting, the DSC sends a notification to a process called the *actif* server. We have only one actif server (running on a workstation) for the whole SPS RF control system. It keeps an image of the last settings loaded in the equipment (see section 8).

4.3 Equipment Control Assembly layer

In our architecture an ECA cannot communicate directly with another ECA. It can only respond to a DSC. This implies that a sequence involving equipment connected to different ECAs must be implemented in the Front End Computing layer.

The following tasks are implemented in the Equipment Control Assembly layer:

Timing driven equipment control. The 14.4 seconds long SPS supercycle consists of 1 proton cycle, followed by two lepton cycles used for filling LEP. Synchronization of the various accelerator equipments during the supercycle is done by the *Master Timing*.

The RF ECAs have dedicated hardware, responding to this timing, and thereby sending cycle specific settings to the RF equipment. This solution was prefered to a software one because it guarantees real time operation without overloading the CPU.

Remote communication. The ECA first parses the command received from the DSC, it then checks that this command is allowed considering the present state of the equipment. Following that, it either executes a mini-sequence switching the equipment or reads the hardware or the RAM (stored data). It finally sends a reply back to the DSC.

Surveillance. During switching operations possible time-out situations are surveyed. In such a case the ECA returns the equipment to a safe state. Equipment error conditions are monitored by means of polled or equipment driven surveillance.

Initialization. The application RAM data area and the connected equipment are initialized into a safe state at start-up.

Logging. Fault logs are maintained and can be requested via the remote communication for analysis. Specific equipment settings are saved. The last state can thereby be restored after a power supply failure.

The last three tasks listed above are only implemented in the ECAs controlling RF power equipment, i.e. amplifiers, cavities, power supplies ...

The application software in the different ECAs uses generic code and is driven by *tables* that are generated during the development phase. The equipment specific code is limited to initialization and surveillance tasks.

5 Command Representation

The control system is mainly concerned with the manipulation of commands: Command generation, filtering, routing to the server concerned, transformation into a sequence of more elementary commands, etc ... Choosing an adequate model for a command is thus very important.

5.1 The Graph

The *RF Graph* consists of two types of objects: *nodes* and *leaves*, related by filiations (arrows). See figure 2. We impose the following restrictions on the graph

- there is a single root node,
- there is no recursive path,
- leaves have no child,
- each node has, at the most one child leaf.

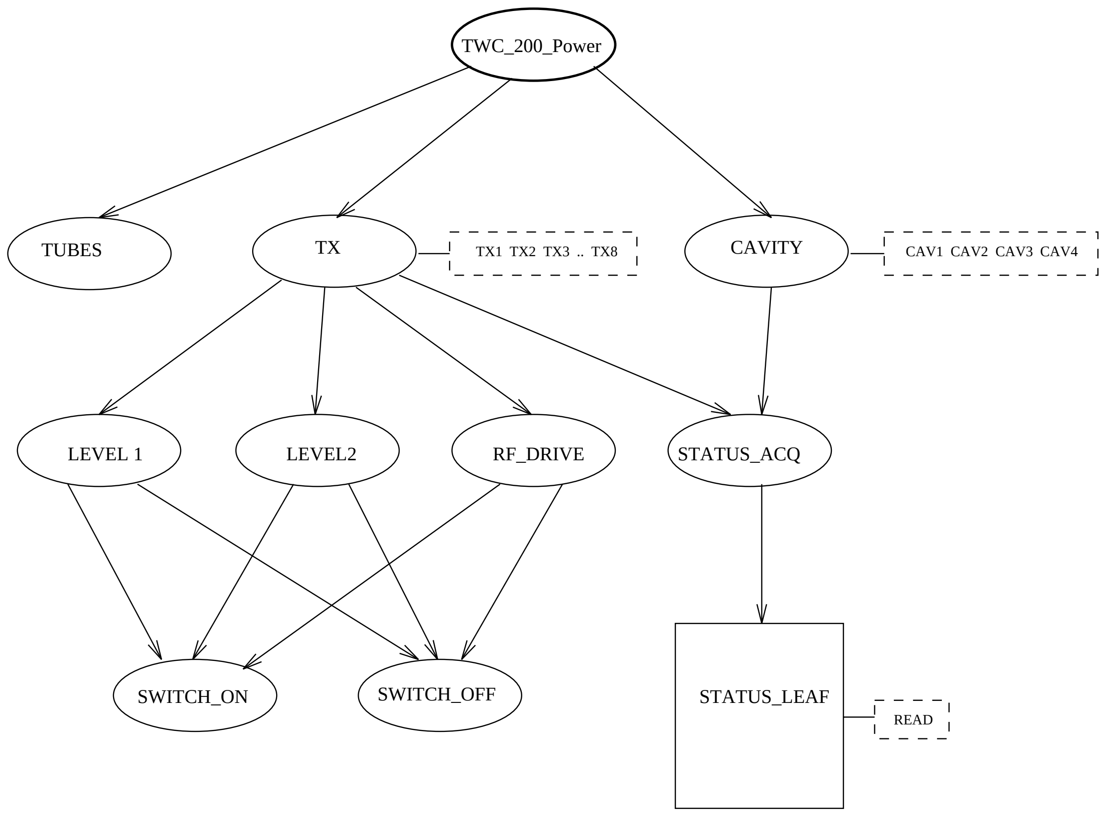

Figure 2: A portion of the TWC 200 MHz Power graph

We have one such graph per RF system (as defined in section 2). The root is the system's name. Nodes are meant to represent functionalities, with children nodes as sub-functionalities. Figure 2 shows a portion of the graph representing the four Travelling Wave Cavities at 200 MHz. Each cavity is driven by two amplifiers traditionally called transmitters (TX). Nodes are represented by small ellipses. The dashed rectangles represent a list of *targets* to which the functionality of the corresponding node can be applied: CAVITY can be applied to the four cavities (CAV1 to CAV4); TX can be applied to the eight transmitters (TX1 to TX8). Wherever a node has only one target, it is not drawn on the figure.

Leaves are meant to represent data. They are drawn as square box on the figure. The STATUS_LEAF describes the data returned by a status acquisition on either a cavity or a transmitter. The leaf can eventually be applied to a set of *transfer modes* drawn in a dashed rectangle. The STATUS_LEAF has only one transfer mode: read.

5.2 A Command

A command is a *path* through the graph, starting at the root, with a list of selected targets at each node, and, if the path ends in a leaf, a single selected transfer mode plus possible parameters and data (write).

Refering to figure 2, we derive the command:

switching on the RF drive for transmitters 1,2 and 3.

- node **TWC_200_Power** and selected target **default**
- node **TX**, selected targets **TX1, TX2 , TX3**
- node **RF_DRIVE**, selected target **default**
- node **SWITCH_ON**, selected target **default**.

This graph representation presents the following advantages:

Generic code. The graph model, with the corresponding data base (section 9), leads to generic applications: The same code can be used for all RF equipment. We have more than 100 different commands ranging from the switching on of a transmitter, to the modification of a setting and the acquisition of a large amount of measurement data. The model fits them all.

Treatment. Tree-like structures are optimum because they are visited very efficiently. We have a few hundred nodes. But, wherever needed, the search through the graph (for example the filtering defined in section 4.2) will, at each node, be limited to its child nodes only, leading to an exponential gain in speed.

6 The Man Machine Interface

The Man Machine Interface program (MMI) guides the user through the generation of a command via menus and selections. It is directly driven from the graph: Upon selection of a node, the list of corresponding targets is proposed. Then the choice of children nodes is displayed, etc ... until one reaches a terminal node or a leaf.

The MMI consists of two companion UNIX processes. When an operator starts the application, he starts `bulles.exec`, and this forks `hprr` (figure 3). These two processes communicate via RPC with the DSC layer, `bulles.exec` talking to the process `kernel` and `hprr` talking and listening to `rptrply`.

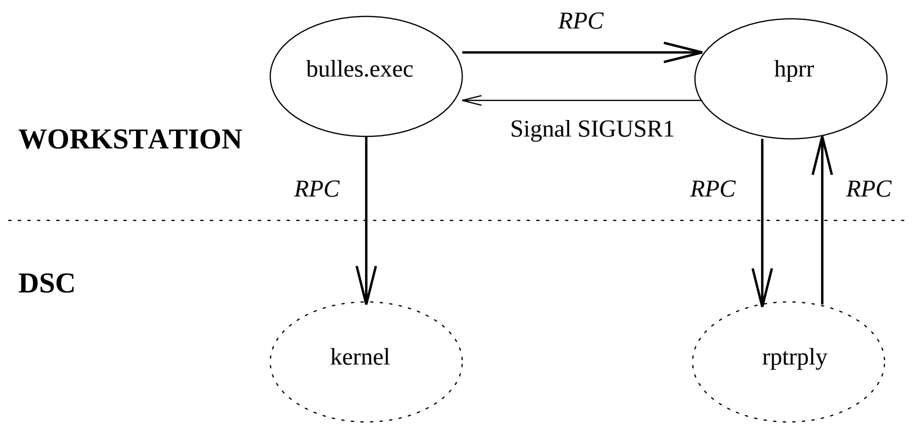

Figure 3: The Man Machine Interface program

6.1 Process bulles.exec

This process is the graphic interface between the user and the equipment. It has been developed using the Uniform Man Machine Interface model proposed by the Controls Group [8]. Fortyfour views (*synoptics*) have been created with the graphic software DV-Draw. Each synoptic contains a set of graphic objects. Following the model [8], an *Application Interface File (AIF)* describes all actions to be executed when the user selects (click via the mouse) an active object in the synoptic. This AIF is written in a *meta language* and is later translated into C-language using a Code Generator [8]. The resulting C code is then compiled, resulting in the program `bulles.exec`.

In each DSC connected to some RF equipment, a single UNIX server called `kernel` is running. This server is the unique entry point to access the relevant RF equipment.

Communication between `bulles.exec` and `kernel` has the following functions:

User identification. While server `kernel` is always running in the DSC, a new MMI program can be started at any time. Before sending any command, it must first call the login() service to be identified. From this identification the DSC will assign a capability to the recognized user, thereby restricting action on the equipment.

Reservation. Some equipment (a transmitter for example), can be controlled by only one user at the time. Using the service take(), one becomes the owner of a portion of the equipment. Service release() makes it available to other users again.

Command. When the user sends a command to the DSC its capability is checked and, if it is too low, his command is rejected.

Reload setting from archive. We allow a user to reload a setting from archive even if he does not have the sufficient capability to manipulate that setting via the service command(). To prevent an abusive usage of reload() in place of command(), parameters of the reload call are protected by a cryptographically secure hash function, MD5 (ref [9]).

A more detailed description of the graphic interface program can be found in ref [10]. A user's manual is also available (ref [11]).

6.2 Process hprr

Process `hprr` is the return channel receiving reports and replies back from the DSC. In each DSC connected to some RF equipment, a single UNIX process called `rptrply` is running (figure 3). This process is responsible for sending all reports and replies back to the `hprr` process of the MMI programs.

The mechanism is the following: As explained in section 4.2, execution of a command may result in a lengthy sequence of steps (lasting as long as 15 min. for some commands on a transmitter). During this sequence, intermediate replies will be returned, informing the user of the actions in progress. These replies are sent from client `rptrply` to its server process `hprr`. Process `hprr` will then send a UNIX signal (SIGUSR1) to `bulles.exec` informing him that a reply is available. As soon as possible, `bulles.exec` will then make a call to its server `hprr` to read this reply.

This complex mechanism was necessary because the graphic interface `bulles.exec` is only responding to signals (mainly clicks on the mouse) and cannot perform as a server of `rptrply`. An additional process `hprr` was thus designed as a server of both `rptrply` and `bulles.exec`.

The user can specify the *level of detail* for receiving intermediate replies: More elementary steps in the sequence will then cause no reply to be sent. A similar mechanism is available for the reports.

Communication between `hprr` and `rptrply` has the following two functions:

Asking reports. By asking for reports on a system, the user can be notified of the control activity of other simultaneous users.

Transmitting reports and replies.

For a more detailed description of process `hprr`, please consult ref [12].

Figure 3 shows a MMI program communicating with a single DSC. When controlling the RF equipment through several DSCs, `bulles.exec` will be connected with several `kernel` servers (one in each DSC), while `hprr` will also be connected with several `rptrply` (again one in each DSC).

7 The Front End Computing layer

Figure 4 shows the processes running in a DSC. The process `kernel`, responsible for filtering the command, and `rptrply` reponsible for transmitting the reports and replies have already been presented in section 6.

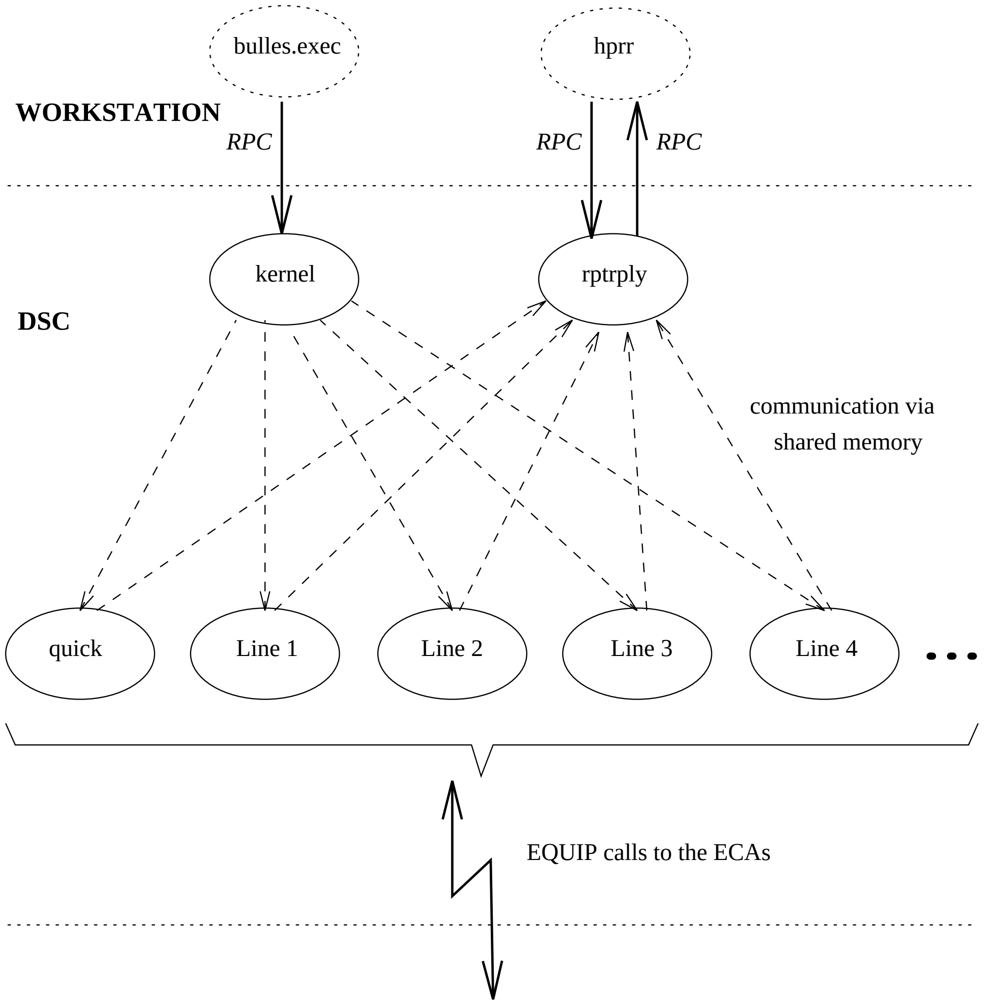

Figure 4: The processes in the DSC

The other processes (`quick`, `Line 1` ... `Line 4`) are called *executers*. They are all constructed from the same C code modules, each being responsible for the control of a portion of a RF system. They implement the sequencing mentioned in section 4.2. When assigning the RF equipment to executers we want independent devices to be controlled by different executers. Let us again take the Travelling Wave 200 MHz system as example: We have four independent cavities, each being driven by two amplifiers. We shall map this equipment into four executers (`Line 1` ... `Line`

4), each controlling one cavity with its amplifiers and accessories (hybrid, power supplies,...). Thanks to the parallel processing of LynxOS, we can then run simultaneous sequences on several cavities. Upon reception of a command, process `kernel` routes it to the concerned executer(s). During sequencing, this(these) executer(s) will send intermediate replies to process `rptrply`. Communication to and from the executers is done via a *shared memory* structured buffer system. The reservation mechanism (section 6.1) gives the user exclusive control of all the equipment under a chosen executer. Status acquisition commands on all four cavities are routed to an additional executer called `quick` (see figure 4).

When forwarding a command to an executer, the `kernel` labels it with a *Run Time Name*. To this Run Time Name corresponds a table (in the private data base of the executer) listing a series of steps (the relevant sequence). Each step can either be a reference to another table (a more basic sequence), or a reference to a program (such as a communication with an ECA). Customization of the control of a given RF equipment is achieved by the construction of these tables.

8 Actif - Archives

A single server *actif* is responsible for keeping an image (called the *actif file*) of the current settings of the whole RF equipment. For that task, the process `rptrply` in each DSC must notify it if a command modified a setting. This is done via the RPC channel shown on figure 5. Server `actif` keeps a copy of these settings in *shared memory*. Periodically, a copy is also saved on the disk.

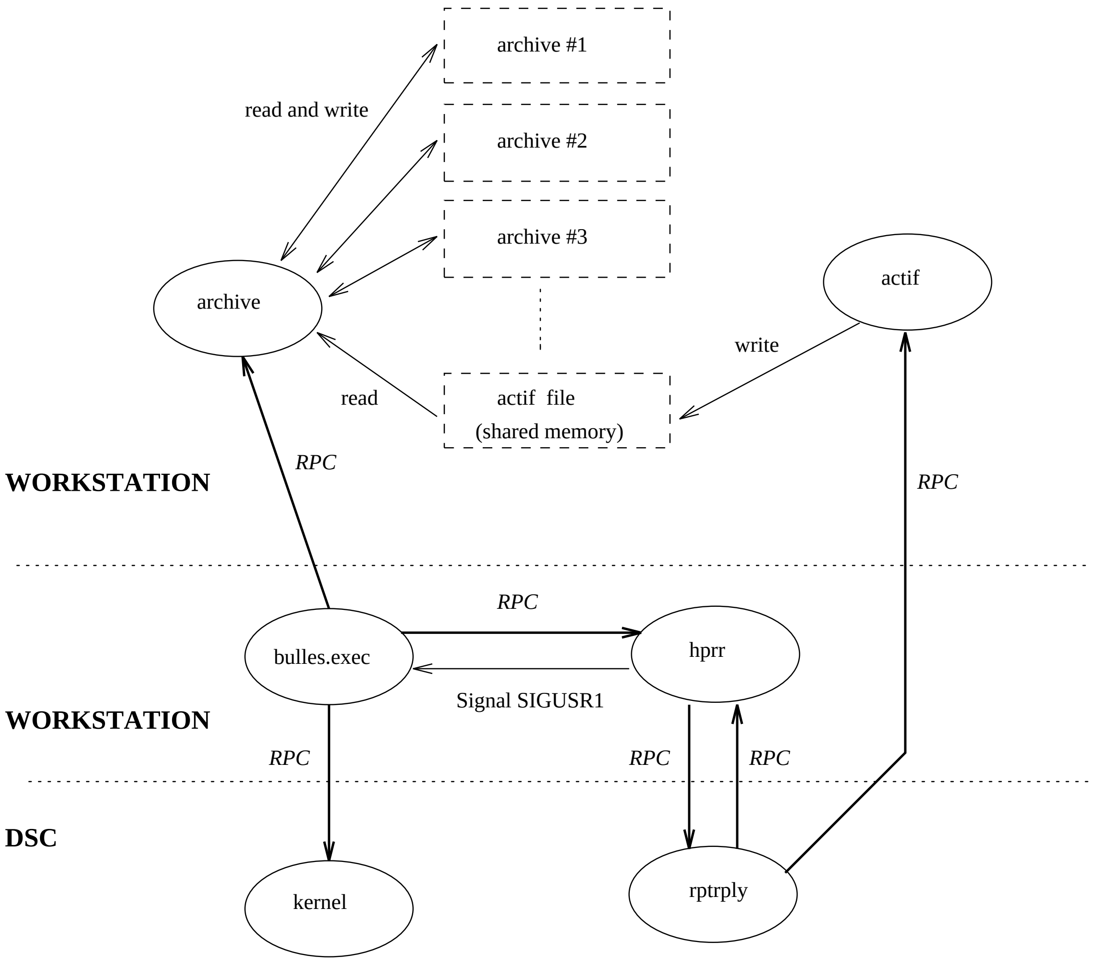

Figure 5: The servers `archive` and `actif`

The MMI program `bulles.exec` is a client of server *archive*. Using these services, it can archive the present settings (i.e. copy the actif file onto an archive), or retrieve old settings from an archive (in order to reload the hardware). For more details on these processes, consult ref [12].

9 The graph data base

9.1 Content of the graph data base

The graph data base describes all RF commands. For each command, it provides:

A representation. Node and leaf *names* are sufficient since a command is merely a path through the graph (section 5.2). In addition, nodes and targets have a *description* to be used by the MMI program for proposing commands.

The needed capability. A minimal needed capability is defined:

- For each node-node and node-leaf filiation. A user can follow this arrow only if his capability is at least equal to the specified one.
- For each target of each node. This restricts the set of targets available to the user.
- For each transfer mode of each leaf.

The reservation. For each target of a node, the data base indicates whether the user must reserve the executer or not.

The DSC. The address of the DSC serving that command.

The EXEC. The name of the executer responsible for the treatment (section 7).

The Run Time Name. The name of the command recognized by the executer (section 7).

The 1553 fields for EQUIP. For an *elementary* command (i.e. a command making a single EQUIP call), the 1553 fields FAMILY, MEMBER, ACTION, MODE, USOPT can be defined (see ref [7]):

- For each target of each node.
- For each transfer mode of each leaf.

The leaf. If a command involves data (read or write), its path will terminate in a leaf. The graph gives the possible transfer modes. For each transfer mode (read or write), the data base indicates:

- The types of data. We allow a subset of the three following types: List of longinteger, list of float, list of string. Whether the number of items in each list is fixed or variable, possible maximal number of items in each list, bounds, . . . are also defined in the data base. The MMI synoptic to be used, and for each data item, a description to be used by the MMI program for entering the data (write) or for displaying the data (read).

- The types of parameters. (At present parameters are used in the read transfer mode only). Again we allow a subset of the three following types: List of longinteger, list of float, list of string. Parameters can have fixed values defined in the data base, or be editable. In that case, bounds may be defined. The MMI synoptic to be used, and for each parameter, a description to be used by the MMI program are also defined in the data base.

A *conversion algorithm* may be defined (section 9.3).

In addition to the graph data base, a small *users* data base gives information on some privileged recognized users. Most important is the capability level assigned to each of these users.

Figure 6 shows the portion of the graph relevant to the command:

switching on the RF drive for transmitters 1,2 and 3.

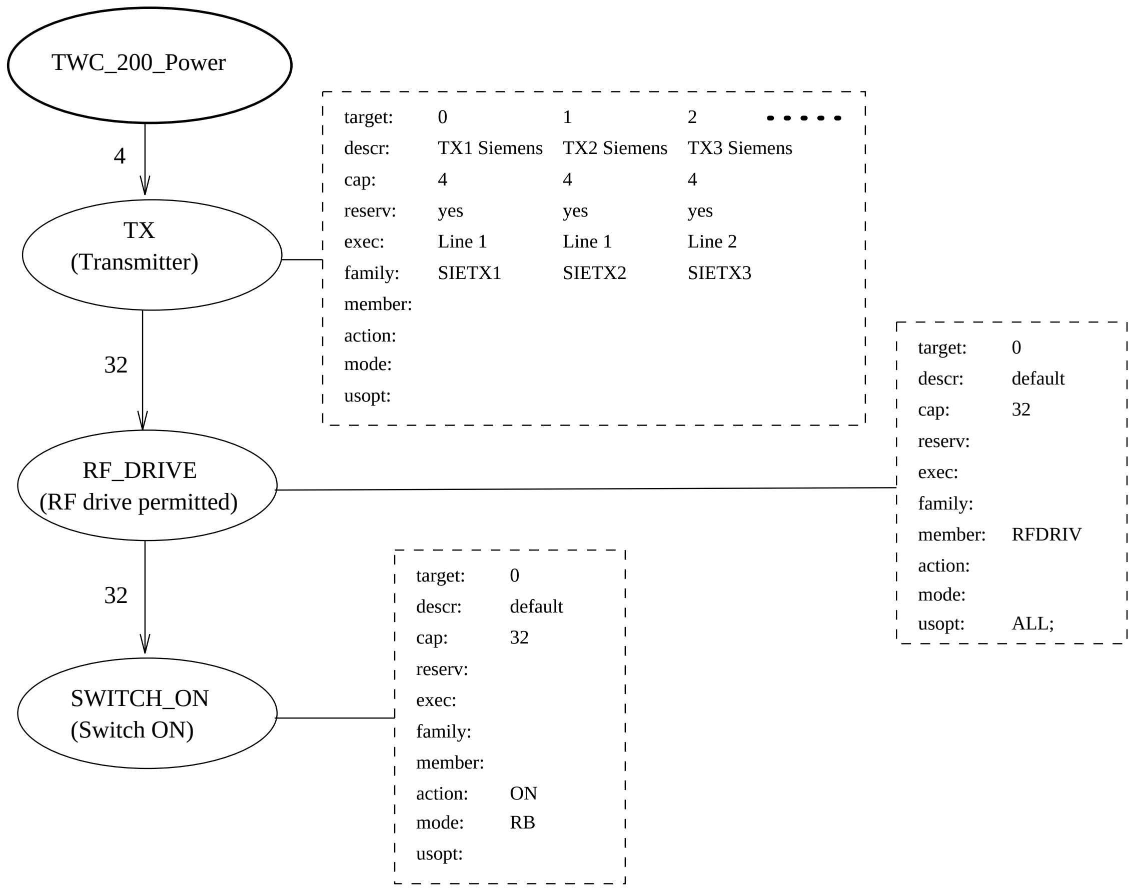

Figure 6: Content of the graph data base

In each node, we find the node description (between brackets) used by the MMI program. The numbers on the node filiation arrows are the required capability levels: 32 corresponds to the user Machine-Operation, and is the minimum required to use sub-functionality RF_DRIVE underneath TX. A user with a lower capability could only move to sub-functionality STATUS_ACQ shown on figure 2.

Each node has at least one target. (Only the first three targets of TX are listed). One possible attribute of the target is the specification of the executer: The first two targets of node TX are controlled by the executer `Line 1`, while the third is controlled by `Line 2`.

Finally, the 1553 fields are easily derived from the attributes of the targets in each node of the command path. (Wherever an attribute of a target is not defined, the corresponding field is left empty in figure 2).

Not represented on figure 6 are the specification of the DSC, (actually dependent on the system only, i.e. defined at the root), and the Run Time Name, dependent on the path in the command (and thus defined by the sequence of nodes).

9.2 Implementation of the graph data base

9.2.1 Data base design

The information analysis was first done using the NIAM method (Natural Information Analysis Method) [13]. This method consists in the representation of knowledge as *facts* between *objects*. Derivation of the proper facts and objects from the graph is straightforward: Most important objects will be *node*, *target*, *leaf*, *system*, *node_name*, *node_leaf_connection* ... And facts will read like:

- A *node* belongs to a *system*
- A *node* is the father in a *node_leaf_connection*

Using this method, one also defines *constraints* between these objects. For example:

- In the same *system* two *nodes* cannot have the same *node_name*
- A *node* is the father of only one *node_leaf_connection*

The product RIDL* [14] from IntelliBase is a CASE (Computer Aided Software Engineering) tool integrating a graphic interface for entering the NIAM diagram, an analysis package checking that the diagram entered does not violate the rules of NIAM, and finally a mapping package generating a SQL text file with statements creating the tables and defining the constraints for the chosen DBMS (Data Base Management System). We have entered the NIAM diagram of our graph data base into RIDL*, and have obtained the SQL text file for creation of our data base onto the Oracle DBMS (version 7). (For an explanation of the Oracle terminology, please refer to Oracle v7.1 user manuals).

The graph data base is mapped into 52 Oracle tables (a total of 235 columns) and 366 constraints. Of particular interest in the use of version 7.1 is the enforcing of foreign key constraints at the table level, thereby preserving the integrity of the data base. (For example one cannot delete a node if it has some targets).

9.2.2 Entering data into the data base

Seven *Forms* have been created, using SQL*Forms (version 3.0), to enter, query, update and delete information in the data base. By designing the forms with version 3.0, the integrity constraints have been automatically transformed into the appropriate triggers. This implements a first filtering of the modifications, at the forms level.

9.2.3 Extracting data from the data base

Information from the data base is needed by

- The MMI program: Process `bulles.exec` needs all the information from the graph except for the Run Time Names and 1553 fields. Process `hprr` needs to know which DSC is serving each system.
- Server `actif` needs information on all commands involving settings (i.e. ending in a leaf with a possible write transfer mode).
- Server `kernel` in each DSC needs information on all systems (served by that DSC) to filter the commands received, and to forward them to the relevant `exec` process.
- Process `rptrply` in each DSC only needs to know what systems are served by that DSC.
- Each `executer` needs a list of all the Run Time Names that it implements, plus the corresponding 1553 fields.

We do not use SQL*Net (real-time SQL calls) because this is incompatible with the desired speed of the control system. Beside, the RF control must be able to continue running without the data base server. Instead, we have used a philosophy similar to the one used in LEP [15]. We have developed a set of *data extraction programs*, written in SQL*Plus with PL/SQL blocks. These programs generate, for each of the process mentioned above, a dedicated set of *flat tables*, i.e. ASCII files with one record per line, and items separated by at least one blank character. These files are read by the process during initialization. The flat tables have the exact format desired by each process: Each line in a table contains the fields to be stored in a corresponding C-structure using the C function fscanf(). References between these C-structures (i.e. references between lines in different tables) are computed off-line by these SQL programs, so that initialization of the process is very fast.

Per system, 22 flat tables are created for `bulles.exec`, 7 flat tables for server `actif`, and 11 flat tables for server `kernel`. Process `rptrply` and `hprr` need a single flat table each (with information on all systems). Each `executer` needs 2 files.

The extraction programs also perform the consistency checks that cannot be easily described as Oracle7 constraints on the data base. These are typically checks involving a *path* through the graph, thus implying procedural capability (loop on the nodes, branches and exception handling) that are available in the PL/SQL language. For example, we check that for each path, the executer is defined once (only in one node); we also check that each 1553 field is not defined twice (in different nodes). The flat tables will be generated only if all these checks are positive.

9.2.4 Platform

The Oracle data base is installed on a DEC 3000 machine. SQL*Forms and the SQL*Plus data extraction programs run on this machine. The flat tables are then copied onto the workstations and the file servers of the DSCs via FTP (File Transfer Protocole).

9.3 The procedural data base

The graph is a *declarative* data base. At the very end of the treatment of a command, in the executer, one may need to apply a *conversion algorithm* to the data in order to match the format required by the specific RF equipment. This hardware has been developed over the last 23 years, thereby presenting a broad variety of digital access. (An example of conversion algorithm is a Binary to BCD - Binary Coded Decimal - conversion for a frequency synthesizer). Fortunately the same conversion algorithms can be used in many places in the RF. So, we have developed a small *procedural data base* consisting in a library of 23 conversion algorithms (C functions) to be linked with the executers. These algorithms are described in ref [16].

10 Acknowledgements

Many thanks to Krzysztof Kostro for his help on RPC calls. Marc Vanden Eynden provided guidance in the development of the MMI program. We were convinced to use the Oracle data base by the recommendations of Josi Schinzel, Paul Heymans and John Poole. Genny Ferran gave the support for the use of the CASE tool RIDL*. David Nathan (DaNathan Advanced Standard) wrote all the conversion algorithms under a specific contract. The shared memory buffer system was developed by Ann-Kari Amundsen and Hans Petter Christiansen. And finally, many thanks to our colleagues of the SPS RF for their enthusiastic encouragements during the commissioning of this new control system.

References

[1] The 300 GeV Programme, CERN/1050, 14 Jan. 72.

[2] M.C. Crowley-Milling, G.C. Shering, The NODAL system for the SPS, CERN 78-07, Sept. 78.

[3] PS and SL Controls Groups, Controls Users Meeting at Chamonix, CERN/SL/90-93(CO), July 90.

[4] PS and SL Controls Groups, PS/SL Controls Consolidation Project, CERN/SL/91-12(CO), April 91.

[5] P. Anderssen, P. Charrue, R. Lauckner, P. Liénard, R. Rausch, M. Tyrrell, M. Vanden Eynden, Interfacing Industrial Equipment to CERN's Accelerators and Services Control System, CERN-SL-95-42 CO, April 95.

[6] P.S. Anderssen, V. Frammery, G. Morpurgo, User Guide to the Network Compiler Remote Procedure Call (NC/RPC), LEP Controls Note 97, May 89.

[7] The LEP/SPS Controls Group, The LEP/SPS Access to Equipment, LEP Controls Note 54, Version 1.03, Feb. 86.

[8] P. Ninin, J.Ph. Regin, P. Sollander, M. Vanden Eynden, Uniform Man Machine Interface model for the control of industrial equipment, CERN SL/90-94(OP), Version 1.3, 23 Jan. 92.

[9] B. Preneel, Cryptographic Hash Functions, European Transactions on Telecommunications, Vol.5, No.4, Jul-Aug 94, pp. 431-448.

[10] P. Baudrenghien, H. Marty, The SPS RF Control System, Interface Homme-Machine, available from http://nicewww.cern.ch/SL/RFSPS/spsrfdoc.htm.

[11] P. Baudrenghien, E. Bracke, H. Marty, J. Molendijk, Y. Pilchen, F. Weierud, The SPS RF Control Application, User's guide, available from http://nicewww.cern.ch/SL/RFSPS/spsrfdoc.htm.

[12] Y. Pilchen, The SPS RF Control System, Report & Reply, Archives, available from http://nicewww.cern.ch/SL/RFSPS/spsrfdoc.htm.

[13] J.J.V.R. Wintraecken, the NIAM Information Analysis Method, Theory and Practice, Kluwer Academic Publishers, 1985.

[14] RIDL* (V1.2) manual set, IntelliBase nv/sa, Plantin en Moretuslei 220 bus 3, 2018 Antwerpen, Belgium, 1991.

[15] J. Poole, The Database Systems for LEP Control, CERN SL/90-65(MR), June 1990.

[16] P. Baudrenghien, E. Bracke, U. Wehrle, The SPS RF Control System, Conversion Algorithms, available from http://nicewww.cern.ch/SL/RFSPS/spsrfdoc .htm.