

# Data Format Design for Heterogeneous Environments

by *M.D.Geib Vista Control Systems, Inc.*

## Introduction

This paper discusses some of the issues involved in designing data formats for use in multi-platform computing environments. Issues include how to design file formats that can be read on any platform and how to transmit data across a network to support communication between different platforms. Additional discussion is presented on choosing or defining data formats to represent non-atomic data types that hold information such as time.

## Designing Platform independent files

Many systems make use of data external to the system, such as disk files. It may be advantageous to use these files on multiple platforms. For systems that make use of data files, one aspect of achieving platform independence is the support for these data files.

Many systems use read or read-and-write files that must be portable. For example, a utility on one platform is used to generate a file that other related utilities read and these other utilities may be running on various platforms. In this case the utilities must be able to read the file independently of which platform produced it.

### File headers

One method of making data files portable is by prefixing them with a standard file header which includes information indicating how the file was written. The header should include enough information to allow an application to determine the byte-ordering of data in the file, the format of the floating-point data, the character collating sequence and possibly the format of any time data included in the file. Once this file header is developed, any utility can be developed to support reading and writing of portable binary data. This method has the advantage that for files written and read on similar systems in a homogeneous environment, no type conversion is required, resulting in better performance.

A file header like the following has been developed at Vista to prefix any binary data files that must be portable.

- A two-byte integer with value FFFE
- A four-byte integer with hex value FFFEFD FC
- A four-byte integer containing the major version number
- A four-byte integer containing the minor version number

At this point the byte ordering can be determined. With the byte order established the file version information can be read.

The remaining data in the header includes a four-byte integer that indicates the data type and size of the data that follows. That is, each of the remaining fields is made up of a four-byte prefix, followed by the data. The remainder of the header follows:

- A four-byte integer that indicates the single precision floating point format
- A four-byte integer that indicates the double precision floating point format
- A four-byte integer that indicates the character collating sequence, ASCII or EBCDIC
- An eight-byte integer dependent on the version for validating the format for 64-bit integer values.

- A single-precision floating-point value to validate the floating-point format for single-precision values
- A double-precision floating-point value to validate the double-precision floating-point format
- A four-byte integer indicating the platform where the file was written
- And finally a four-byte integer which indicates the end of the header

Because only the initial portion of the header is fixed, the header can easily be modified in future versions of applications that read or write the file. Prefixing each data item allows for some flexibility in the format of the header and makes it easy for applications to support the reading of data files produced with different versions of the system.

The remainder of the data file is written in a similar fashion. All data items are prefixed with a four-byte integer to indicate the data type of the data that follows. Using this prefix allows for greater flexibility in interversion support. When possible, older versions of the system can ignore data that they do not recognize in new files, and newer version applications can supply defaults for values not supplied when reading older version data files.

The above method relates specifically to binary data files, but a similar approach can be taken with text files. A file header of a standard known character sequence can be written to provide enough information for an application to determine the collating sequence. With this information, the application can make the necessary translation from the data file to the native character values.

### **Using standard data types**

Another approach to producing portable data files is to adopt a standard data format for all required data types. Since a single format is used for all data applications one, simply converts to that type for writing and converts from that type for reading. For systems that have native data types that match the standard format chosen, no conversion is ever required. However, the use of standard data types prevents any performance gain when the platform where the data is written is similar to the platform where the data is read, if both have native types different from the chosen standard file format.

## **Data communication**

As was the case for portable data files, many systems must pass data between different machines while they are running. This requires that all the machines read and write data that other machines write and read. Like the data file, to handle this problem for data communication, a number of approaches are available. Note that since different platforms use different character collating sequences, the numeric value for a given character cannot be assumed to be fixed.

### **RPC**

A number of Remote Procedure Call packages are available that handle many of the problems associated with passing data between different machines. These packages automatically handle the conversion of data between the host machine and the client machine.

Typically, a tool is provided to define all the arguments that are passed in the RPC calls. A utility then reads the data definitions and produces code to handle the conversion of the data to and from the network representation.

Some RPC packages detect when the two communicating systems are of the same type and then use the native data types of the platform rather than converting to and from the standard network representation.

### **Low level data**

Using an RPC package for network communication may not always be appropriate, or available. When lower level data communication is required between machines, code must be written to support the conversion of data to and from different platform types.

One approach is to prefix the data with information similar to the binary file header so that the receiver of the data can convert the data to the native types if required. As with the file header approach, this technique has the advantage that for two similar platforms, no conversion is necessary.

Another approach is to convert the data to a standard type for transmission to the other machine. All parties involved must then convert to and from the standard type in order to communicate with the other machines. This is the approach normally taken by the RPC packages. In fact, many of the platforms that support RPC packages provide access to the routines used by the RPC for data conversion, so that users can use the same routines for converting data to and from a standard network representation for their own use.

## **User defined data types and constants**

When designing data for a system which supports many different platforms, one area to be aware of early on is the support of OS specific information. There are a number of data types that are specific to a platform or even proprietary in nature.

### **Time**

Time has a number of different formats depending on the platform. There are a number of supported formats that define a time with a multi-field structure. This is a very flexible implementation, but it tends to consume more memory than required. At Vista it was decided to adopt the same format for time used by OpenVMS, a signed 64 bit integer. This time format supports both delta and absolute times, and has sufficient resolution for real time systems. There are routines supplied with Vsystem to manipulate this time and to convert it to and from a string representation and a representation similar to the popular time structures.

### **Status values**

Status values returned from library functions are normally platform-specific. During a remote call, trying to return the status value from the remote system to the local one for display is useless in many cases. Vista developed a platform-independent facility for generating status values. These status values can be passed between different platforms, with the values always having the same meaning. To support OS specific errors, these are mapped into the Vsystem facility in a platform independent manner so that a similar type error on different platforms generates the same Vsystem error value and maps to a consistent text message.

### **Miscellaneous data**

A number of other types of data may or may not have to be supported in a heterogeneous manner. In networked systems, like those supported by Vsystem, there is some platform data, like process and user identification and callback routine addresses, that are passed between different machines at runtime. Even though this data is never used on the remote platforms in such a system, the data may be stored on the remote platform and then retrieved by the local platform at a later time. In this case, it is important that the data is never disturbed by the remote communication. Sufficient space must be allocated for the storage of this data generated on all the supported platforms. For example, callback routine addresses on a Digital Alpha platform require 64 bits of storage, while most other platforms currently only require 32 bits to store an address. In order to store addresses in such a system, all address storage must be at least 64 bits.

### **System Constants**

An additional problem created when supporting heterogeneous platforms is the value of system constants. One example is the use of the C constants normally defined in the header files `limits.h` and `floats.h`. When a data item is set to the value of `FLT_MAX` on one platform and then passed to a remote platform which uses a different floating point format, the resulting value may be illegal on the remote platform. An easy solution to this problem is to define within the system, constants that are valid on all the supported platforms. If a new platform is supported in the future, it is easy to change constant definitions if required. The code that does the type conversions for a system should detect this problem. The user code could also take the option of substituting the local platform's equivalent value if an illegal value is detected.

## **Recommendations and Summary**

The following is a list of some techniques successfully used at Vista to support the exchange of data between heterogeneous platforms:

- Define new data types for all external data and any internal data dependent on the external type to minimize the impact of incompatible implementation of data types by compilers.
- Define new data types for platform-specific types which will support the requirements of the system on all the platforms. These new data types include status values and error codes.
- Define new constants to replace those supplied by the local environments.
- Be aware of the supported range and valid values for all data types on all platforms.

Designing or choosing data formats for use with heterogeneous platforms requires knowledge of the requirements of the system being developed and knowledge of the data types on all the platforms to be supported. In addition, a well-developed plan must specify how new data types will be handled in the future.

# Real-Time Scheduling of Software Tasks\*

L.T. Hoff  
Brookhaven National Laboratory  
Upton, NY, 11973-5000, USA

## Abstract

When designing real-time systems, it is often desirable to schedule execution of software tasks based on the occurrence of events. The events may be clock ticks, interrupts from a hardware device, or software signals from other software tasks. If the nature of the events is well understood, this scheduling is normally a static part of the system design. If the nature of the events is not completely understood, or is expected to change over time, it may be necessary to provide a mechanism for adjusting the scheduling of the software tasks.

At the Relativistic Heavy Ion Collider (RHIC) front-end computers (FECs) provide such a mechanism. The goals in designing this mechanism were to be as independent as possible of the underlying operating system, to allow for future expansion of the mechanism to handle new types of events and to allow easy configuration. Some things taken into consideration during the design were the programming paradigm (object oriented versus procedural), programming language and whether events are merely interesting moments in time or intrinsically have data associated with them. The design also needed to address performance and robustness tradeoffs involving shared-task contexts, task priorities and use of interrupt service routine (ISR) contexts versus task contexts. This paper will explore these considerations and tradeoffs.

## 1. Introduction

The RHIC control system adheres to the standard model for accelerator control systems[1]. In this model, "front end computers" (FECs) provide access to accelerator equipment and generate reports to be sent to operator consoles. In addition to these tasks, FECs are expected to perform certain routine functions, such as periodically checking readings against range limits, logging data and closing local control loops in discrete steps. These functions are typically scheduled-based on the occurrence of physical (real-world) events. The event may be an interrupt from the accelerator equipment indicating that new data is available, or it may be a clock tick indicating the completion of a time interval, or it may be a software event indicating the completion of the execution of a software task. RHIC FECs use a real-time O/S (VxWorks) specifically so that FEC functions can be scheduled deterministically with respect to these real-world events.

## 2. FEC software

RHIC FEC software has been designed to allow rapid reconfiguration using a concept called an Accelerator Device Object (ADO) [2]. The "core" FEC software is identical in all FECs. Software objects known as ADOs are created within each FEC, establishing each FEC's unique mission. The ADO contains the methods for access to specific accelerator equipment and for performing range checking, closing control loops, etc. on specific accelerator equipment. As new accelerator equipment is added to the system, or as existing equipment or needs change, new or modified ADOs can be created, without changing other FEC software, or otherwise affecting FEC running.

In keeping with the modular ADO design, a flexible, configurable and extensible system for scheduling FEC functions with respect to real-world events was considered to be necessary. The ADO provides methods which know "how" to perform a function, but the event system determines "when" to perform the function. A flexible scheduling system allows for the possibility of tuning the speed of control loops, or changing the point in the accelerator cycle when data is read from accelerator equipment.

---

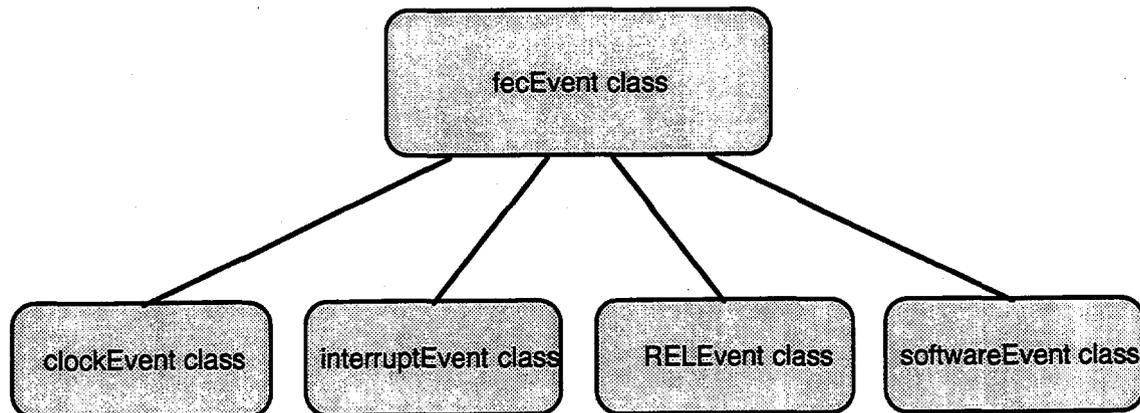
\* Work performed under the auspices of the U.S. Department of Energy

Using an object-oriented design approach similar to the approach used to design the ADO system, real-world events are represented by a software abstraction called an "fec event". The following requirements were established for the "fec event" software object; it:

- must be able to represent the following real-world events :
  - millisecond clock tick
  - interrupt from accelerator equipment
  - RHIC Event-Link (timing system) event
  - "software" event
- must be extensible to represent other real-world events
- must be able to schedule ADO and general FEC functions; an environment must be provided for running any C function or C++ class member function
- the relationship between real-world events and the ADO or FEC function which is executed in response to that event must be reconfigurable without otherwise disrupting FEC operation
- must be able to abstractly represent real-world events in such a way that the idiosyncrasies of the source of the real-world event and the operating system are well hidden within the object

### 3. Design

The fundamental design was developed quickly. The "fec event" class would be implemented in C++ (the only supported object-oriented programming language). An inheritance tree would be developed, with a single root or base class. This base class would handle the establishment of the relationship between real-world events and C++ class member functions and be responsible for executing the functions. Derived from this base class would be specific classes representing various types of real-world events. These specific classes would be responsible for sensing the occurrence of their particular real-world event and establishing an environment in which to execute a C++ class member function when that event occurs. The idiosyncrasies as to how the event is sensed and how the environment is established would be encapsulated within these specialized objects.



### 4. Tradeoffs

Although the fundamental design was developed quickly, it soon became apparent that the details of sensing each real-world event, and especially the details of establishing an executing environment, involved many tradeoffs.

The first consideration was how to execute both C functions and C++ class member functions. The event system was to be used to schedule basic FEC functions (coded in C) as well as ADO functions (coded in C++). Executing a C function from a C++ object is straightforward, using the address of the function. Executing a C++

class member function from an object of a different class is another matter. Traditionally, this problem has been approached using one of two different paradigms.

The first approach is to define a special C++ class, known as a functor[3]. This class has a single pure virtual function, which derived classes are expected to override to perform the desired function. There are several drawbacks to this approach. Firstly, since C++ classes may be part of another inheritance hierarchy as well as the functor hierarchy, it typically requires language support for multiple inheritance. Secondly, it only allows for a single function to be executed per object. It was envisioned that ADOs might have a collection of functions to execute for different real-world events. Finally, this approach has no C language analog, precluding scheduling FEC functions coded in C.

The second approach is to execute only static C++ class member functions. Static C++ class member functions are essentially C functions, so there is a direct C language analog. Furthermore, there is no limit to the number of static C++ class member functions per object. However, static C++ class member functions cannot access non-static class member data, nor can a static C++ class member function be virtual[4]. The latter limitation was not viewed as terribly severe for ADO functions, but the former would prevent ADOs from performing any tasks whose operation depended on data specific to that ADO. A simple workaround for this problem is to pass an object pointer as a parameter to the static C++ class member function.

This last design was adopted for the event system. The base event class was designed to store a list of function pointers (either C functions, or static C++ class member functions) and a list of parameters (either a pointer to a C++ object, or any arbitrary value for a C function). Whenever called upon to execute a function, the corresponding parameter would be passed to the function.

The next debate revolved around whether events are merely interesting moments in time or whether they intrinsically have data associated with them. Some schools of thought maintain that events have data associated with them[5]. The data is related in some way to the root cause of the event. If it is desirable for the event system to hide the root cause of the events, so that one event type can be transparently substituted for another event type, then a dataless design is more appropriate. The drawback of a dataless approach is that there is no straightforward method to provide detailed information about the purpose of executing a function. The executed function must be coded to assume that it is appropriate at the appropriate point in time. The dataless design was chosen for the RHIC event system, though provisions were put in place to easily revise this decision if it proves to be short-sighted.

Each specific "fec event" class is responsible for sensing the occurrence of its particular real-world event. The mechanisms for doing so may depend on the underlying operating system as well as the nature of the real-world event. In traditional timeshare operating systems, such as UNIX, real-world events are typically sensed via device drivers, which execute in "kernel space". In typical real-time operating systems events may be sensed more directly, e.g. a "user space" task may attach interrupt service routines (ISR) directly into the CPU's interrupt vector table.

Each specific "fec event" class must establish an environment in which to execute C or C++ functions. If the operating system uses a heavy-weight process paradigm, where each process has its own protected data space, and all event objects exist within a single process, then a valid executing environment may be provided by the operating system.

RHIC FECs use a real-time operating system which uses a light-weight task paradigm. Data is shared between all tasks including ISRs. Each task is assigned a priority, and the highest priority "ready" task preempts any other ready tasks (priority preemptive multitasking). Using such an operating system, one has latitude in choosing how many tasks are used as executing environments, what the priorities of the tasks are and even whether or not ISRs constitute valid executing environments. In general, the tradeoffs involve performance, robustness, and the need for consistency between executing environments for different event types.

The advantage of using an ISR as an executing environment is a minimum overhead and therefore maximum response speed. The drawbacks, however, are numerous. The executed functions must be ISR-compatible, i.e. they must not use certain operating system functions, they must be fairly short in duration (since interrupts are typically disabled during ISR processing) and they must not contain any bugs which might cause the

ISR to crash, bringing down the entire CPU. For these reasons, the RHIC event system always uses a task context as an executing environment.

Each specific "fec event" class is responsible for setting up one or more task contexts in which to execute the C or C++ functions. The choice of the number of available task contexts trades off overhead versus robustness. C or C++ functions which share task contexts may affect each other's execution. This effect becomes most severe if one function has a bug which halts the execution of the task or merely takes an inordinate amount of time to complete. Using separate tasks for each function avoids these problems at the expense of additional overhead. Most specific "fec event" classes choose a middle ground with a small set of task contexts sharing the load.

Under a priority-preemptive operating system, priorities assigned to the task contexts must also be considered. Events which need more prompt attention, such as interrupt events, should use higher task priorities than events such as clocks with a 1 second tick rate. The RHIC interrupt event class uses the highest task priority. The event-link (timing system) event class uses tasks of two lower priorities than the interrupt event. The clock event class uses yet a lower priority. As future specific "fec event" classes are designed, the task priorities used by them must be appropriate as measured against task priorities used by existing classes.

## 5. Conclusion

The event system has been in use in RHIC FECs for several months now, though not extensively. The system will get its first rigorous test during the injection transfer line test currently underway. Lessons learned during this test will be used to steer the design of the specific "fec event" classes. It is expected that the object oriented design of the event system will allow fine tuning of the tradeoffs used for particular "fec event" classes without affecting other classes.

## References

- [1] B. Kuiper, Issues in Accelerator Controls, Proc. ICALEPCS 91, Tsukuba, 1991.
- [2] L.T.Hoff, J.F. Skelly, Accelerator devices as persistent software objects, Proc. ICALEPCS 93, Berlin, Germany, 1993.
- [3] James O. Coplien, Advanced C++ Programming Styles and Idioms, Addison Wesley, 1992.
- [4] Bjarne Stroustrup, The Annotated C++ Reference Manual, Addison Wesley, 1990.
- [5] V. Paxson, Glish: a software bus for high-level control Proc. ICALEPCS 93, Berlin, Germany, 1993.

# Presentation of Data from the Fermilab Datalogging System

Iouri Ivanov, Kevin Cahill, Brian Hendricks  
Fermilab  
Accelerator Division Controls Department  
MS 347  
P.O. Box 500  
Batavia, IL. 60510 USA

Presentation of data from the Fermilab datalogging system is realized as an application program running under the Fermilab accelerator control system. This program utilizes a data acquisition system which is comprised of more than forty datalogging processes supporting more than 15,000 channels of which 9,200 are in use. Presentation of data can be performed for up to eight devices simultaneously, and is possible in two modes: time plot and X versus Y, where both X and Y are devices. Performance of the system allows retrieval and plot rates of several thousand points per second. Also the user may plot simple arithmetical expressions of devices, plot integrated curves for each expression (or one device), plot fitting functions, and display some simple statistical values about the data.

## 1. GENERAL

Presentation of data from the datalogging system is an important task which allows a user to spot long term trends for both technological and physical processes of the accelerator. The present work has resulted in the creation of a console program that reflects the wishes of accelerator operators, engineers, and researchers to study these trends. Flexible plotting modes coupled with retrieval and plotting rates of several thousand points per second encourage users to make maximum advantage of the datalogging system.

The program consists of two main parts which are mutually linked. They are the data update and the user interface.

The data update portion includes: retrieval of data from the datalogging system, mathematical transformation of data for graphing, mathematical methods of analyzing the data, presentation of data in both graphical and text forms and data exportation using e-mail communication.

The user interface includes all options and buttons which allow the user to define the setup of parameters controlling the session. Those parameters are as follows: time interval of interest, list of devices or arithmetical expressions comprised of devices, the view of graphical presentation of such expressions, etc. Moreover, there are means which allow the user to simplify the task of defining such a setup having from 15 to 100 parameters per plot trace. There are also provisions for saving previous setups for future reuse. There is both a circular buffer of the most recent plot setups as well as a buffer of files into which favorite plot setups can be saved.

## 2. DATA STRUCTURE AND DATA COLLECTION

The fundamental unit of data is a device or channel. The datalogging system supports more than 15,000 channels on more than forty nodes (9,200 channels are in use).

Presentation of data can be performed for up to eight channels simultaneously and is possible as either a time plot or an X versus Y plot, where X and Y are both devices. Devices have a number of attributes. These include a name, an array index number, a datalogger node, a number of points to skip, and a minimum time between points. The last two attributes are used to decrease the number of points read for densely populated devices.

Some devices (or even all of them) may be used as operands in simple arithmetical expressions. Each expression occupies a grid in the graphical window. In this sense, it is convenient to consider a single device that isn't part of an expression as a trivial expression. All grids for one session are overlapped in a graphical window, and two such windows are available to the program. Plot grids have attributes as follows: initial and final values for the Y axis, the type of Y axis scaling - linear or logarithmic, the type of trace, the fitting function type - polynomial (up to third order) or exponential, and the presence/absence of statistical information. Statistical information includes integral, average, and standard deviation values.

All the grids together comprise a single plot. A single plot has its own title, number of divisions for the X and Y axes (the same for all grids), and data point matching algorithm. The data point matching algorithm does not matter for time plots and the trivial expression case. However, when two or more devices are involved in one expression (like a nontrivial arithmetic expression for a time plot or any expression for the X versus Y case) the method of data point matching makes a difference. The menu of choices includes interpolation and matching with another menu for specifying accuracy.

The time interval made up of a start time and an end time is an attribute of the entire plot. So, it is common for all grids.

Data collection for each device on the page is organized by calling a library routine which receives as parameters the device attributes (or their derivatives) and the desired time interval. This routine sends a request to the datalogger process on the node specified and receives data from it. The returned data is packed in two arrays of the same dimension. One contains values in engineering units, and the other contains corresponding timestamps in seconds. These arrays for all devices of a single plot are kept in program memory and are freed just before the next plot session starts.

### 3. PRESENTATION OF DATA

#### 3.1 GRAPHICAL PRESENTATION OF DATA

Before collecting the data, the program creates up to eight overlapped grids on a graphical window and labels them with different, hard-coded colors which serve to identify and distinguish them.

After having completed the collection of all data involved in the expression, the program starts the plot. If there are two or more devices, the user has a choice of establishing a correspondence between points of different devices. The user may choose either "Interpolation" or "Matching". In both cases, the program identifies the device with the least number of points collected. In a loop for each of these data points, the program either calculates the interpolated point for any other device in the expression that corresponds to the timestamp of the selected device, or looks for the point with the closest timestamp within the limits of accuracy for that point in the matching case. Then the point coordinates are saved for plotting.

There are various plot trace options. These options include whether or not to connect points, what symbol to plot, symbol size, whether or not symbols are filled, and the number of symbols to plot.

The symbols available are circles, squares, triangles, diamonds, and crosses. Also, special ASCII-characters can act as symbols depending on the order number of the grid. These characters are X O I = N Z \$ %. If a character is chosen as the plot symbol, the size and fill parameters are ignored. The number of plotted symbols per trace can range between 0 and 65535. It is also possible to have every point plotted as a symbol. The color of the symbol plotted corresponds to the color of the associated grid.

Along with the curve of the expression, the integrated curve (multiplied by a user selected scale factor) and a fitted curve can be drawn. Both of those options are available for both time plots and X versus Y plots, where X is a device and Y is an expression. In the latter case, two options are supported. In the first one, X and Y are fitted independently, and their fitting functions define the equations of the parametrically defined fitting function. In the second option, the program fits just the resulting set of points for Y(X).

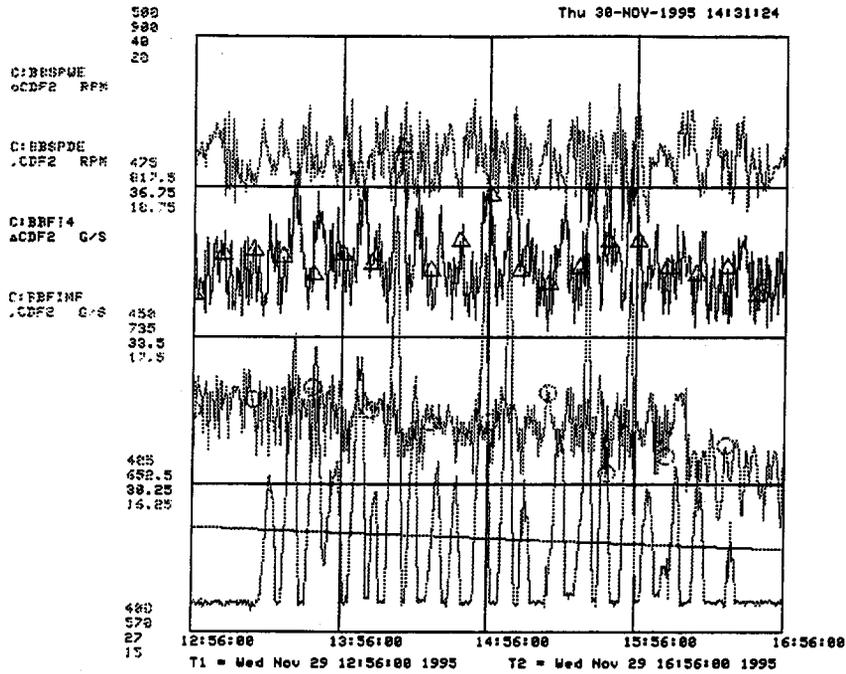


Fig. 1. An example of graphical data presentation with one exponential fit.

### 3.2 PRESENTATION OF DATA IN TEXT FORMAT

The numbers of points read for each device and plotted for each trace are displayed on the background text window. Also displayed there is the statistical information that includes the integral value, the average value, and the standard deviation value.

To explore the curve behavior in a manual way, the user may click on the "Calc Points" button. A popup window will be created to display the current grid coordinates of the cursor in the graphical window in the grid's units. The maximum and minimum values of expressions on the grid may be displayed as well after clicking on the "Min/Max" button.

After the plot has been completed the user may list the data in a popup window. There are options to list either the data actually read or the plotted data. To eliminate points from the data read, there exists an option to edit the collected data set (not the plotted data) and then to replot the edited data.

All statistical values are displayed on the graphical window and appear in the text window upon request.

The fitting function equations appear in a popup window upon request and can be displayed with different time units as the independent variable.

After the plot has finished the user may send the plotted data in pairs (time or X-value, expression value) to a user's e-mail address. The time values can be formatted in three different ways: regular format (day-time), seconds (since 1-Jan-72), or time in seconds after midnight on the first day.

## 4. USER INTERFACE

The user interface can be divided into five parts: individual attribute control, saving and recalling of plot setups, plot setup editor, presentation of text and graphical data and session control.

The first part presents a group of buttons which help the user to define attributes of individual devices, grids or the current plot setup.

The second part presents a group of buttons that help the user to work with saving and recalling setups that are associated with the circular buffer and the saved buffer of files.

The third part allows the user to edit, in a single action, several items of a single plot setup.

Presentation control consists of buttons which are responsible for plotting, listing and exportation of data and has been previously discussed.

Session control consists of two buttons named "Stop" and "Pause". The first of them cancels a plot, and the second suspends/continues a plot. These buttons only have an effect while plotting is in progress.

#### *4.1 INDIVIDUAL ATTRIBUTE CONTROL*

There are several groups of buttons designed to control different attributes of devices, grids, and single plots. There are some other attributes that are entered in special fields. These are the device name and the initial and final values of the grid. These attributes are read from their fields just after the command is issued to start the plot.

There are some individual attribute buttons for each device or grid. They control the datalogger node of the device and the type of fitting function. These attributes stay resident on the background text window. A shortage of display space makes it impossible to have an individual attribute button for each grid and for each attribute. Thus some attributes are not displayed on the background window and are displayed only if the operator clicks on the corresponding button. These are "Trace", "Symbol", "Skip", and "Units". The buttons related to devices and grids bring up a menu including all devices or corresponding grids and display the current state of the attributes. These buttons can be used both to display and to change attributes.

The buttons "Trace" and "Symbol" control the trace type of the plot and the type of symbols plotted, respectively. Their features have already been discussed.

The "Skip" button allows the user to choose a number of points to drop from the data read between accepted points. This removal of points is done by the application. There is another option to specify the minimum time interval between points. This work would be done by the datalogging process which has the benefit over the skip method of reducing network traffic.

The "Units" button allows the user to set a different Y axis scale for each grid. For user convenience, all "invisible" grid attributes can be displayed (and changed) by using a hidden button attached to each grid.

The buttons "X divisions" and "Y divisions" control the numbers of X and Y axis grid lines (for all grids).

Options to calculate statistical values are controlled by the buttons "Integrate", "Average" and "StdDev". Clicking on each button brings up a menu asking whether plotted curves are channels, integrals or both. When integral values have been chosen, the time units for the integrated curve and integral value should be input as well as a scale factor to adjust the presentation of the integrated curve in the graphical window.

There are several ways to change the time interval (T1, T2). First, the user can simply type in a new date and time over the old one and interrupt. Clicking on the "T2 now" button will immediately enter the current date and time for T2. Clicking on the "Interval" button allows the user to specify a time interval between T2 and T1 and enters the resultant date and time for T1. The "Inc/Dec" toggle button allows selection of the direction of time intervals.

#### *4.2 SAVING AND RECALLING SINGLE PLOT SETUPS*

There is a one to one correspondence between datalogger nodes and files containing circular plot setup buffers and save and recall files. These files are opened automatically after clicking on the "Data source" button and selecting the datalogger node desired.

Each plot setup is temporarily saved in a circular page buffer having thirty records. Each such setup can contain more than 100 parameters describing up to nine devices. The user can go quickly back to a plot setup used a few plots ago by clicking several times on the "Previous" button. Similarly, the "Next" button will bring the user to the next setup in the circular buffer.

More important setups can be saved in a save and recall file. Its capacity is sixty records. The current setup can be saved by using the "Save" button and inputting a descriptive title. To recall a record from the file, the user just clicks on the "Recall" button and then on the title of the setup desired in the file menu.

### 4.3 PAGE EDITOR

To give more flexibility for making setups, the concept of Enabled and Disabled devices was introduced. It provides an option to temporarily exclude (disable) or include (enable) some devices and their attributes into the current plot setup. This can be done for multiple devices at once.

If it is desired to have many devices with some of the same parameter values (node, initial and final values, etc.), the "Copy" button can be useful to accomplish this. The "Edit" button provides search and replace capabilities for device names. This is useful for setting up a plot when some device names contain common substrings.

```

D44 Lumberjack Datalogger
Plot Title = Refrigerator tests -- F14 --
X=TIME      Y=C:ESSPNE      ,C:ESSPDE      ,C:BEF14      ,C:BEF1MF      ,
            I= 400          , 570          , 27          , 15          ,
            F= 500          , 900          , 40          , 20          ,
            oCDF2          ,CDF2          ,oCDF2          ,CDF2          ,
            NONE          POLY=1          NONE          NONE          ,
            480           480           480           480           Read
            480           480           480           480           Plotted

            Y=
            I= 0           , 0           , 0           , 0           ,
            F= 10          , 10          , 10          , 10          ,
            .CDF2          ,CDF2          ,CDF2          ,CDF2          ,
            NONE          NONE          NONE          NONE          ,
                                                    Read
                                                    Plotted

T1= Wed 29-NOV-1995 12:56  T2= Wed 29-NOV-1995 16:56  ♦Inc♦ ♦T2 Now♦ ♦Interval♦
♦Skip♦   ♦X Divs 4♦       ♦Matching ♦       ♦Integrate♦ ♦Editor♦
♦Trace♦   ♦Y Divs 4♦       Accuracy ♦ 10♦     ♦Average♦
♦Symbol♦ ♦Overwrite♦     ♦Previous♦ ♦Next♦   ♦StdDev♦
♦Recall♦ ♦Save♦         ♦Fit Equations♦
Data Source ♦CDF2 (Crs48)♦ ♦List data♦ ♦Copy♦ ♦Min/Max♦
♦All Device Plot♦List = ♦ 1♦ ♦E-mail data♦♦Enable♦♦Calc Points♦
Messages
Page has been recalled, record N      29
controls_misc:lumjack12.dat

```

Fig. 2. Buttons and device attributes on the background text window.

## 5. SUMMARY

The program usage statistics show that this program enjoys great popularity among control room operators, accelerator physicists, engineers and administrators. This program covers most user needs for graphical data presentation, mathematical manipulation and data exportation, and its services continue to expand. A great deal of effort was expended to provide high speed data acquisition (thousands of points per second) for presentation. This was achieved by economizing the network traffic needed, modifying library routines to support the needs of the task, and optimizing the datalogging processes themselves. The next upgrades include the option to specify different time intervals for different devices and the option to display basic status values.

# XCON - A Motif Based Toolkit for Graphical User Interface Design for Control Applications

L. Kopylov, M. Mikheev  
*Institute for High Energy Physics  
Protvino, Russia.*

B. Hendricks  
*Fermilab  
Batavia, IL, USA*

The XCON toolkit has been created and used for the development of a number of applications for the Fermilab and IHEP accelerator control systems. It provides an easy way to create a sophisticated graphical user interface for control applications, in particular ones with complex synoptic diagrams. The toolkit is based on Motif and contains a wide set of control widgets in addition to a powerful graphical editor and has the capability of including GIF images as a screen background. Equipment access is implemented as a library layer which allows this package to be adapted to other control systems. A description of the XCON toolkit and a few examples of its use are presented in this report.

## 1. Introduction.

The Graphical User Interface (GUI) is an essential part of accelerator control applications. The large quantity of equipment and the complexity of physical and technological processes place serious demands on the operator's and engineer's professional knowledge to control and maintain the facility. An approach should be considered in application development which facilitates operation of the accelerator on the basis of a general knowledge of physics and technology. The solution is to develop an application with a powerful and comprehensive GUI that contains enough descriptive information about the facility to be controlled. The most advanced type in this direction is a synoptic-based GUI. It presents a schematic drawing of the facility or schematics of technological processes together with the set of the parameters to be controlled.

A number of commercial systems provide such solutions, for example Vsystem [1] or EPICS [2]. As a rule, those systems are a general complex solution of a facility control task including, together with the GUI, the internal database, equipment access and network communication. For a control system developed from scratch this is an attractive solution, while for an existing mature control system it is not the case because a complete set of services is already implemented which has its own local peculiarities. In such a case, the implementation of a complex package could be an expensive solution or not acceptable at all.

While developing the package called XCON we had a goal to provide a graphical tool to build a GUI for control applications that would allow powerful and comprehensive synoptic diagram development with minimal programming and which could be easily adopted into an existing software infrastructure. Despite a large number of commercial packages for Motif GUI development (eg. DecVUIT, FaceMaker, and X-Designer), such activity still requires special software support and remains very time and effort consuming. XCON liberates the developers from routine coding and allows them to concentrate on the technological and engineering aspects of application design.

## 2. Package contents.

An interactive graphical interface builder utility (XCON) and a runtime kernel for application execution (XEXE) are combined in the package together with a number of routines for file format conversion. XCON is a complete set of tools to create the GUI. It includes a graphical editor and control widget set for data representation and setting. The result of development in the XCON environment is a GUI description file that is executed later under the XEXE kernel. Such functional separation allows for the simplification of the execution kernel and an increase in the efficiency of its operation. Dedicated to GUI development, the XCON utility has an extensive user interface based on the Motif tool kit. A package FIG [3] has been used as a prototype of a graphical editor. An example of a GUI design session in the XCON environment is shown in figure 1. There are three main windows: command, mode selection and canvas window. In the last one, the developer is actually drawing the GUI components.

Conversion utilities allow the user to compile an input for XCON from different descriptive files. In these files, a position and an active element type should be specified. On the basis of this data, XCON builds a frame that can be combined with a desired background later on. This approach is suitable for quick GUI generation for large numbers of similar equipment described in tabular form. The same method was used to port an existing application to the XCON environment for Antiproton Accumulator control at Fermilab.

### 3. Interface development.

An application GUI consists of a passive background picture and a number of active graphical elements (AE) representing the equipment behavior. The background picture provides additional information to describe the operation of the facility. It could be either a synoptic diagram or a functional schema of the installation. XCON provides a wide set of graphical primitives to draw the background picture including lines, polygons, curves, circles, ellipses, and text. By grouping a few primitives, a complex graphical element can be produced. Once developed, an element could be saved in a library and reused later.

A set of functions is implemented to edit the elements once drawn. It includes Move, Copy, Size, Mirror, Rotate, Delete, and Group/Ungroup functions. The curves are editable by moving individual nodes. Files in GIF format can be included as a background. Multiple space layers in the element description provide the correct element superposition on the control screen. A grid with variable spacing and two coordinate side rulers simplifies the positioning of objects. The position of the object on the canvas can be snapped to the grid nodes. Drawing one selected primitive or performing a desired edit function is referred to as an individual mode of XCON. The mode of operation can be selected either by clicking on the corresponding button on the Mode Panel or by means of a cascading menu in the Command Window. For each mode, the corresponding attributes are indicated on the Command Window (line type, color, filling style, text font, etc.) where they can also be modified.

Active elements (AE) of the GUI present on the screen the state of equipment while the application is running. They show the values of measured parameters and device operational status and allow the setting of control parameters to desired values. In active (running) mode, all AEs are tightly coupled to their corresponding accelerator devices through equipment access library functions. A number of AE parameters should be specified during GUI design. Three groups of parameters can be distinguished: geometric, graphic and parameters belonging to the corresponding equipment. Geometric parameters determine the position and size of active elements on the control screen. The XCON graphical editor is used to specify this group of parameters (so called WYSIWYG style).

Equipment description parameters include the control device name and control device type. The first one defines the addressing of equipment through the control network, while the second one defines the method of device access which includes the data type and the corresponding equipment access routine name. In some control systems, a method is implemented to retrieve the device descriptive information from the database based upon the device name (like `dio_lib` at Fermilab [4] or `EqLib` at IHEP [5]). In this case, only the device name should be specified. The rest of the device specific information will be gathered from the database.

Active elements are implemented by the use of widgets. The graphical parameters of active elements define how the information will be presented on the screen (ie. the widget type and its resources). A set of widgets are included in the package for managing analog and digital data including text field, scroll bar, scale, plot, LCD style indicator, and wheel switch selector. The standard Motif widgets were used as well as specialized control widgets [6..8]. The resources of AE are specified through descriptive forms. An example of such a form for a plot widget is shown in figure 1 at the right. The resources that could be modified by means of XCON are the subset of the corresponding widget resources set. The rest are set to their optimal value for this application.

To show the status of a multistate device, a special active element was developed which is called a token. It changes the image according to device status. In the design phase, a set of graphical planes are prepared and are assigned to the corresponding device state. The device type defines how the conversion should be done from device state to the proper token plane. The token might be used for setting data. To do this, an action or a list of actions could be specified. It will appear on the screen as a menu of actions when the token is activated by clicking on it with the mouse. A set of standard actions allows the user to set the desired value to the device, open a new or close the existing window, open a Control Panel, or initiate the execution of the predefined action. The Control Panel (CP) is a combination of AE dedicated for the operation of one complex device. In the design phase, a dummy device name is specified for all AE on the CP. It will be replaced with a real device name during application execution. Figure 2 shows (on the right) the CP for the power supply of a magnet element.

There are two modes of the canvas window, development and simulation. In the first phase, the developer draws and specifies the GUI components using the XCON graphics editor and resource specification forms. Active elements are drawn as a named rectangle in this phase. In the simulation phase, the canvas presents a real view of the designed interface screen. All widgets pop up in their final appearance according to the resources specified even though they are disconnected from the equipment.

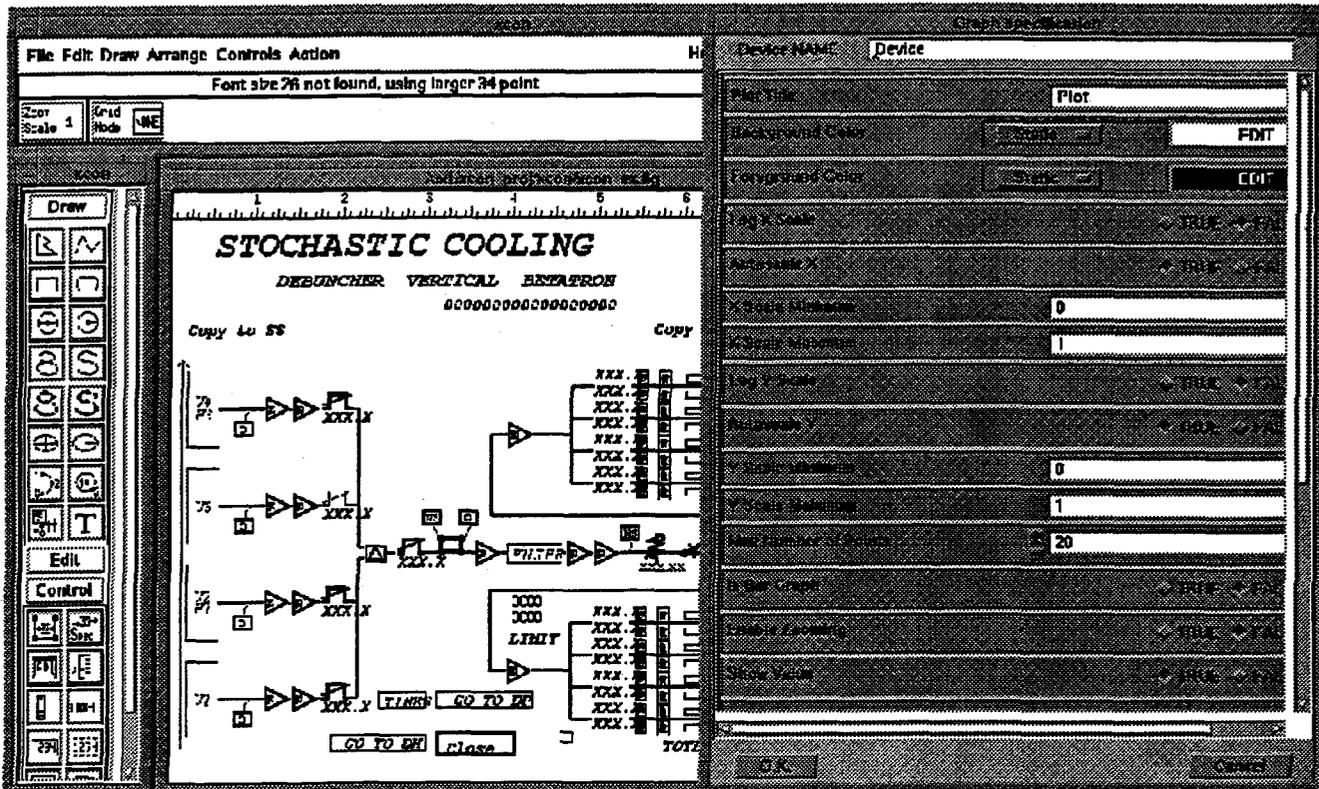


Fig.1 Example of XCON interface design.

#### 4. GUI execution.

For execution, descriptive files are supplied to the XEXE kernel. The kernel creates a window on the control screen with a background and active widgets connected to the accelerator equipment. The widgets reflect the data corresponding to the device states.

The communication is implemented through an existing equipment access library (`dio_lib` in the case of Fermilab and `EqLib` in the IHEP version). The executive kernel attaches to the library through the layer of Customization Functions (CF) which convert the specific kernel calls to the standard library functions. The number of specific functions corresponds to the number of different device types. Here is the place where the developer can put his own code creating specific or virtual device descriptions. For most "standard" devices, the communication is covered completely by equipment access functions, and CF serves just an interface purpose.

A refresh of the control screen is implemented by means of periodic polling with a rate specified by the operator. The kernel consequently passes the list of devices presented on the control screen. The polling list is updated each time when a new window appears or an old one is closed. Two specific aspects exist in the implementation with `dio_lib` at Fermilab. The first one is the extended set of routines in `dio_lib` that return values in engineering units. This significantly simplifies the CF layer. The second aspect concerns the possibility of requesting a list of devices in one call instead of with



# A Data Driven Graphical User Interface for Vacuum Control Applications

L.Kopylov, M.Mikheev, N.Trofimov  
Institute for High Energy Physics, Protvino, Russia

P. Strubin, M. Steffensen  
CERN, Geneva, Switzerland

## Abstract

It is always a serious challenge to develop a synoptic based graphical user interface for the large installations due to inhomogeneous equipment distribution, large variety of equipment and huge facility size. The solution proposed in the report is based on automatic configuration of the graphical user interface using the description of the equipment and of the topology stored in the database. This approach was used to develop the applications for vacuum equipment control for the SPS and LEP accelerators at CERN. As a result, an impressive quality of synoptics was reached with very efficient maintenance procedure.

## Introduction

The software described in this paper is intended to provide a uniform man machine interface (MMI) to the vacuum equipment of all main CERN accelerators (PS, SPS, LEP). The vacuum controls architecture is essentially similar for all three main parts of the CERN accelerator complex and employs a two layer model [1,2].

The control room layer provides the user interface to the system using graphics workstations and X-terminals. This layer also includes file and database servers. The front-end layer contains a number of front-end computers dedicated to a given geographical area and connected to the control room via the general control network. The front-end computers are used to run vacuum equipment servers: processes that allow access to the equipment over the network from the control room workstations.

The objective of this project is to provide the users with a flexible tool to monitor and control vacuum equipment using synoptic diagrams, pressure plots and other forms of graphical data presentation available on modern workstations. The main problems in developing MMI software for the CERN accelerators are due to the huge facility size, complex topology and inhomogeneous equipment distribution. Vacuum equipment of the CERN accelerator complex includes thousands of devices spread over many beam lines and accelerator rings and undergoes continuous modifications.

To achieve the required flexibility and maintainability of the MMI software, we decided to apply an approach based on automatic user interface configuration according to the accelerator description stored in the ORACLE database. The configuration data is read by the MMI software at run time and determines all equipment related aspects of the user interface, such as layout of synoptic diagrams, arrangement of pressure plots, etc.

## The database and configuration data

Although the formats and structure of the information in the ORACLE database are quite different for different accelerators, the database can extract the following essential information for each piece of the equipment:

- Equipment type identifier.  
The type identifier determines the shape of the icon representing the equipment on synoptic diagrams and for controllable equipment allows the selection of the correct data acquisition and data handling procedures.
- Equipment position in an accelerator ring or along a beamline.  
This information is mostly used to place the icons correctly on synoptic diagrams. Several different coordinate systems are used to specify equipment positions in the database. They are bound by referencing some elements to their global coordinates in the site survey database.
- For controllable equipment:  
The equipment address and some other information which is required to reach the equipment in the control system network.

Also, the ORACLE database provides information on the grouping of equipment into sectors and larger partitions (sextants, octants, rings, beam transfer lines). Unfortunately, the existing database does not contain any explicit information on how the partitions are related to one another, i.e. which partition is next to which, where a beam injection line joins an accelerator ring, or where a beam extraction line "forks". This information is supplied in a separate text file describing the

installation topology for the MMI purposes. An example of the topology description file is given in the Appendix.

Extraction of the information from the ORACLE database is done "off-line" by a special utility which preprocesses the extracted data, merges it with information from the topology description file and produces an indexed data file which is then used as the main source of the configuration data at run time for the MMI software. The purpose of this intermediate buffering is to provide:

- **Guaranteed availability and fast access to the data.**  
The data file resides on a control system file server and can be read within a fraction of second by any console workstation in the control network. This response is not guaranteed by the ORACLE database server which is not intended for real time control applications.
- **Uniform interface to the data.**  
Formats of the records describing partitions, sectors and equipment elements in the data file are identical for all accelerators. As far as possible, peculiarities of the original machine descriptions in the ORACLE database are removed during the preprocessing.

## The user interface

The MMI software has been developed to run on a control room workstation using existing control facilities at the front-end level and below. Despite the general similarity of the controls architecture, methods to access equipment are significantly different in all three accelerators. Due to the inhomogenous equipment interface, the software has been delivered as three separate application programs, one for each main part of the accelerator complex.

All three MMI applications have been written to run under UNIX (HP-UX) using the Motif tool kit and have a large common part implementing the core MMI functionality. At start up, the applications read configuration information from the data file, perform various initializations and enter the main loop displaying a main window (Fig. 1).

The main window gives a general overview of the accelerator on a top level synoptic diagram. All main partitions are shown together with icons indicating the general status of vacuum equipment in the area. The main window is also a starting point for navigation through detailed data views provided by the MMI software (Fig 2). From the main window users can invoke synoptic diagrams, pressure profiles or device tables for a selected part of the installation.

The selected part may include any number of sectors in a contiguous viewing area that can be defined using the main window options. The viewing area is highlighted on an upper level synoptic diagram. The equipment in the viewing area is represented on synoptics by icons placed along a beampath (Fig. 3).

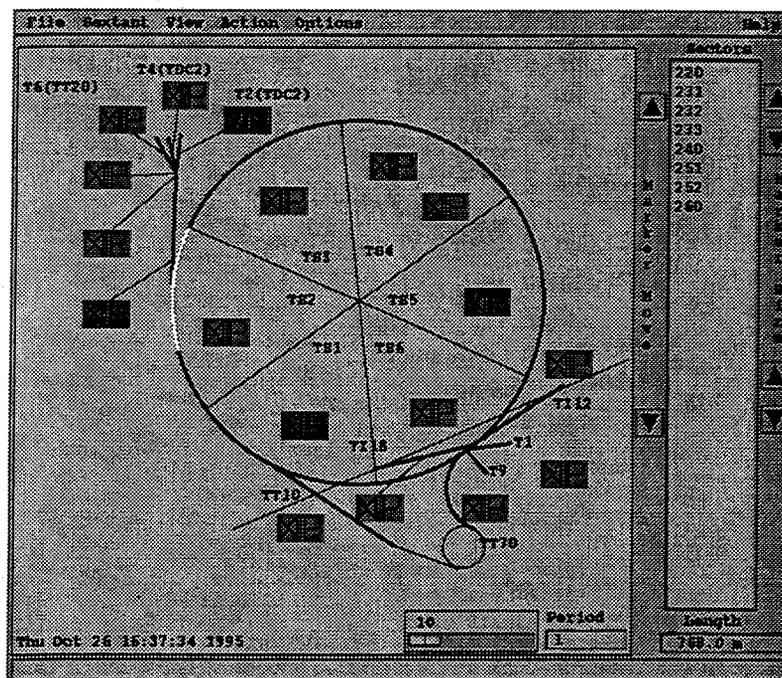


Figure 1. Main window of the SPS-MMI application

Both vacuum and non-vacuum equipment elements are shown on the synoptics; the non-vacuum equipment forms a background that helps users to locate the vacuum equipment of interest. Icon shapes for the non-vacuum equipment are derived from AUTOCAD drawings of the real equipment using a technique described in [3]. The size of non-vacuum equipment icons is roughly proportional to the real equipment size. The vacuum equipment is represented synoptically by fixed size icons according to CERN standards for icon shapes and colors.

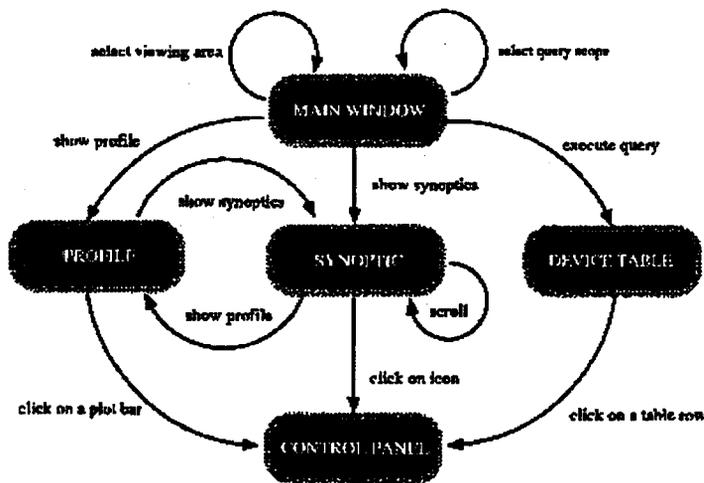


Figure 2. Simplified MMI navigation diagram

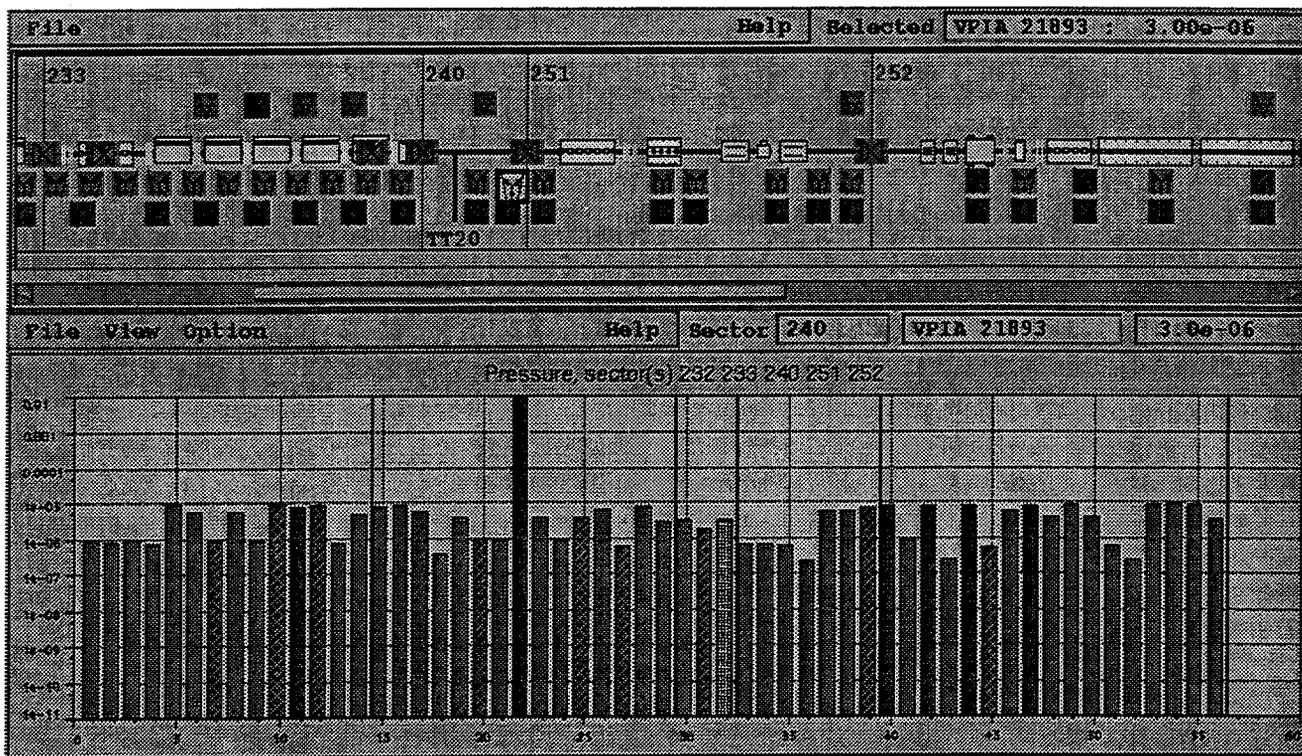


Figure 3. Synoptics and pressure profile for a part of the SPS ring

Relative positions of all icons in the synoptics are proportional to the real equipment positions, with the exception of some vacuum equipment elements which are located so close one to another that the corresponding icons overlap. In this case, a part of the synoptic is expanded to accommodate all icons in that area.

All icons representing controllable vacuum equipment are active: the icon color corresponds to the current status of the equipment and clicking on the icon activates a device control panel (Fig. 4). The control panel displays all device parameters

available in the control system and permits all device actions defined for a given equipment type (eg, open or close a valve).

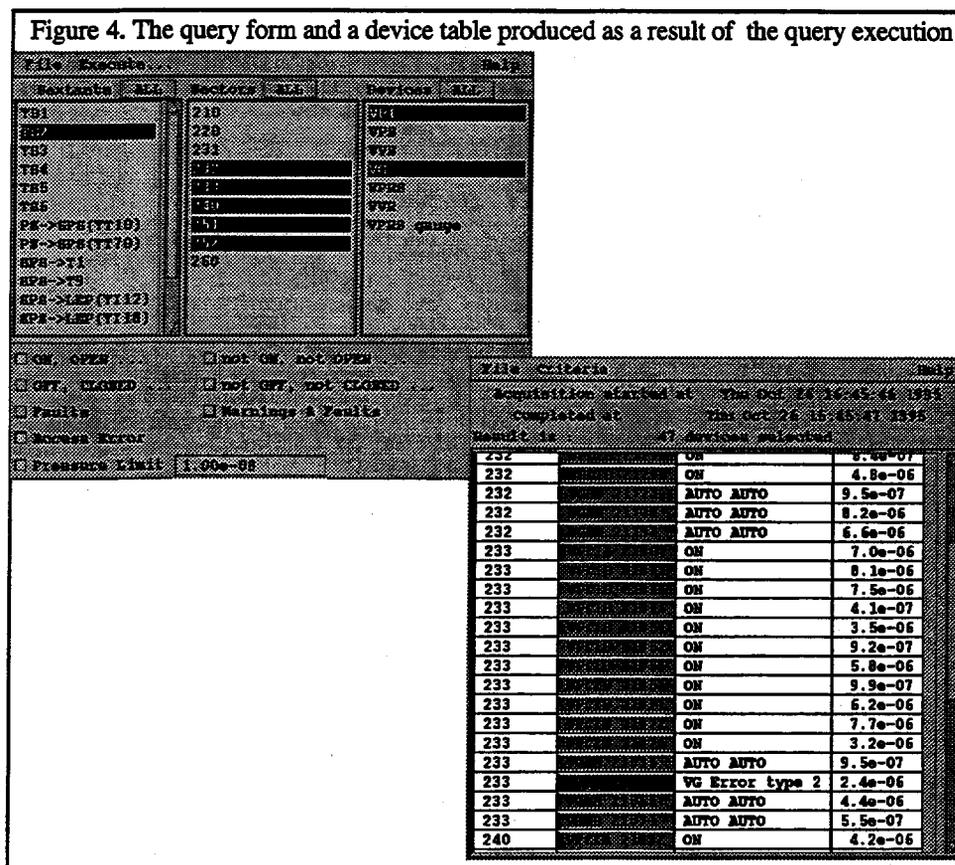
Control panels can also be invoked from a pressure profile. The pressure profile is a bar plot of the pressure measurement data collected from all measuring devices in the viewing area (Fig. 3). The sequence of bars in the plot is the same as a sequence of the devices along a beampath in the viewing area.

Information on synoptics, profiles and control panels is periodically updated to reflect a current equipment state. The data acquisition is performed by the work procedure which is executed when the main loop is idle (not processing any X events). Only the data required to refresh a picture currently on the screen is read from equipment and the whole process of data acquisition and picture refresh is optimized in such a way that the user interface is not blocked for more than one second.

The MMI software provides the users with a capability to execute actions "globally": for many devices of the same type. This option, which is especially useful in large installations, is complemented by a possibility to search within a given scope for the vacuum equipment elements with real-time parameter values that match certain selection criteria. A special form is provided to set the query scope and selection criteria (Fig. 4).

The query option can be used in conjunction with global actions to check the action results, or independently to locate equipment of interest. For example, to select all ion pumps in sectors 110 and 130 that are not ON or read pressure above  $10^{-5}$  mbar. When the query option is used to check a global action result, the last action scope is assumed and the selection criteria are set according to the action executed. For example, if a "close sector valves" action has been executed, the "check action results" option performs a search for the sector valves that are not closed within the last action scope.

Results of a query are displayed as a table where rows correspond to the devices found during the search and columns give values of the parameters used in the selection criteria (Fig. 4). Clicking on a row invokes a device control panel with the full information available on the device.



## Conclusions

The MMI software has been used to operate the SPS vacuum equipment for over a year. All functional requirements has been met and the software has proven to be both reliable in routine operation and flexible to equipment changes. The database driven-design enables easy maintenance of the software. The only thing the users need to do to keep the user interface up-to-date is change the equipment database. A large part of the SPS-MMI application is accelerator-independent

and implements general MMI functionality. As a result, it has become possible to produce similar applications for the PS and LEP accelerators in a rather short time.

## References

1. P.M. Strubin, "Controls for the Vacuum System of LEP", J. Vac. Sci. Technol. A5(4), July/August 1987
2. W. Koelemeijer et al., "Upgrading the Controls of the SPS Vacuum System", EPAC, 1992
3. L.Kopylov et al., "Tools for Man-Machine Interface Development in Accelerator Control Applications", Proc. of ICALEPCS'93, Berlin, October 1993

## Appendix - Topology description file

A part of the installation, or "beam line", is described in the file by a string containing the line name (preceded by keyword LINE) and coordinates of the first and last equipment elements in the line. A contiguous segment of the installation can be described by several consecutive LINE strings.

The installation topology is described in the file as a number of "root" lines from which other lines are branching. A root line description is delimited by keywords ROOT\_LINE (followed by a name of the line and keyword CIRCLE if this line is a circular accelerator) and ROOT\_LINE\_END. Descriptions of connected lines are enclosed in keywords CONNECT and CONNECT\_END and can be nested. Keyword CONNECT is followed by the name of the connected line, the position of the connection point and the connection type identifier. For example, beam injection line TT10 which joins the SPS ring from the left (LI) at the 735.4 meter point in the SPS coordinate system is described as follows:

```
CONNECT TT10      735.4 LI
```

A simplified (not all lines are included) example of the topology description file given below describes a part of the SPS accelerator complex. Comments in the file are preceded by the '!' character.

```

ROOT_LINE SPS      CIRCLE ! SPS main ring
LINE SPS          0.0 6912.0
! Injection PS->SPS, line TT10
CONNECT TT10     735.4    LI
LINE TT10        0.0    876.05
CONNECT_END      ! End of line TT10

! Injection PS->SPS, line TT70
CONNECT TT70     6261.2    LI
LINE TT70        0.0    665.0
CONNECT_END      ! End of line TT70

! Extraction to T2, line TDC2
CONNECT TDC2     1837.4    LO
LINE TT20 0.0    633.23
LINE TT22 0.0    89.0
LINE TT23 0.0    153.16

! Extraction to T4, line TDC2
CONNECT TDC2     722.23   RO
LINE TT24 0.0    142.74
CONNECT_END

! Extraction to T6, line TT20
CONNECT TTT20    633.23   RO
LINE TT25 0.0    258.95
CONNECT_END

CONNECT_END      ! End of extraction to TDC2

ROOT_LINE_END    ! End of line SPS

```

# C++ Library for Accelerator Control and On-Line Modeling

S.Kuznetsov

Kurchatov Institute, 123182 Moscow, Russia

## Abstract

A new object-oriented package has been developed at the SIBERIA SR source complex. This paper presents a formal approach to the interface between the control of a storage ring or transport line and beam modeling. A class library for on-line modeling is portable and uses the standard MAD input file interface for machine description. The object-oriented conception enables the realization of suitable parent-child class trees for control in "beam" terms. Simulation is based on the first order matrix formalism.

## 1. INTRODUCTION

SIBERIA-2 is a dedicated light source with an electron energy of 2.5 GeV that was developed by the Budker Institute of Nuclear Physics (Novosibirsk) for the Kurchatov Institute (Moscow) and commissioned in the beginning of 1995 in Moscow.

Using on-line modeling software complexes is very popular at modern accelerator facilities.[1-4] This type of software is very important during the commissioning of the machine. One characteristic of synchrotron radiation facilities is the continuous modifications carried out for new insertion devices. The software support for work in beam terms is necessary. We have tried to design an object-oriented library that combines control methods for specific equipment (power supplies, pulse generators, etc.) with control methods in beam terms: coordinates, angles, betatron tunes, chromaticity, energy.

Beam Objects	coordinate, angle, betatron tunes, chromaticity, energy...	$X, X'$ $Z, Z'$ $\nu_x \nu_z$ $\xi_x \xi_z$
Element Objects	bending magnets, correctors, quadrupole lenses, sextupole lenses, beam position monitors.	B1, B2 KX1, KX4, F1, D2, F2
Control channel Objects	ADC, DAC	IF1, ID2, ISx, ISz, IKX1, IKX4

Figure 1. Objects of an accelerator control system.

## 2. DESCRIPTION

To control the magnet system of a beam transport line or storage ring we use the classes: TMSControlChannel - magnet elements control, TPSControlChannel - power supplies control, TBMChannel - virtual beam-monitor channels control. These classes are derived from the abstract class TChannel. The hierarchical class structure is presented below:

```

TChannel
  TMSControlChannel
    TMSGGroup
      TLineGroup
        TRingGroup
  
```

```

    TMultiGroup
TPSControlChannel
TBMChannel
THandle
  TMatrixHandle
    TControlHandle
      TSimpleHandle
      TMSHandle
        TQXZHandle
        TSXZHandle
        TCorHandle
          T2CorHandle
            TX2CorHandle
            TZ2CorHandle
          T3CorHandle
            TX3CorHandle
            TZ3CorHandle
          T4CorHandle
            TX4CorHandle
            TZ4CorHandle

```

To realize control procedures in beam terms, on-line information about the magnetic structure of an accelerator is necessary, namely the transport matrix, input/output Twiss parameters for every element of the structure, etc. The class TMSControlChannel represents all necessary requirements for such on-line modeling.

```

TMSControlChannel  class
class TMSControlChannel : public TChannel
{
.....
    PYMatrix          XMatr, ZMatr;
    PYVector          Xin, Zin, Xout, Zout, Xoffset, Zoffset;
    PTwVector         TwXin, TwZin, TwXout, TwZout;
.....
    virtual void      FillMatrix(PTEnergy);
    virtual void      SetTwissIn( PTwVector, PTwVector);
    virtual void      TwissInOut();
    virtual void      SetXZIn( PYVector, PYVector);
    virtual void      SetXZOffset(PYVector, PYVector);
    virtual void      XZInOut();
.....
    virtual void      NewAdd(PTLineGroup);
    virtual void      Copy(PTMSControlChannel NewChan);
.....
};

```

The class TMSGGroup is a set of magnetic elements (e.g., objects of the TMSControlChannel class) and on the other hand it is a magnetic element itself, for which all the necessary modeling procedures are realized. The first group of methods provides the standard 'List' implementation for TMSControlChannel objects. Another method supports calculations for orbit and Twiss parameters.

```

TMSGGroup  class
class TMSGGroup : public TMSControlChannel
{
protected:
    TMchanNode        MChannelList;
.....
    virtual void      Add( PTMSControlChannel);
    virtual void      AddAt( PTMSControlChannel,int);
    virtual void      Free( PTMSControlChannel);
    virtual void      FreeAt( int);
    virtual PTMSControlChannel  At( int);

```

```

        virtual int                IndexOf( PTMSControlChannel);
        virtual PTMSControlChannel FirstThat( TMCondFunc, void *);
        virtual PTMSControlChannel FindWithName(char *);
        virtual void                ForEach( TMActionFunc, void *);
        virtual void                ForEachIter( TMActionIter, void*);
        virtual void                Copy(PTMSControlChannel*);
        .....
        virtual void                FillMatrix(PTEnergy);
        virtual void                TwissInOut();
        virtual void                XZInOut();
        .....
};

```

The realization of the method which calculates standard Twiss parameters for a group is:

```

void TwissLine(PTMSControlChannel AMchannel, PTMSControlChannel BMchannel, void* PS)
{
    if(PS)
        BMchannel->SetTwissIn(PTMSGGroup(PS)->TwXin, PTMSGGroup(PS)->TwZin);
    else
        BMchannel->SetTwissIn(AMchannel->TwXout, AMchannel->TwZout);
        BMchannel->TwissInOut();
}
void TMSGGroup::TwissInOut()
{
    ForEachIter(TwissLine,this);
    TMSControlChannel::TwissInOut();
    TwXout->mu = MChannelList->Mchannel->TwXout->mu;
    TwZout->mu = MChannelList->Mchannel->TwZout->mu;
}

```

This is the complete code for the method . One can see the readability of the code. In this manner we can build a beam transport line or a storage ring structure and get information about the machine functions and update it after any variation in the settings of magnetic elements in real-time.

Class THandle and its derivatives are used to provide linking between control channels of different types. Objects of this class convert beam parameters (for example coordinates and angular deflection of the electron beam at a given accelerator azimuth) to control settings for magnetic elements (e.g., corrector magnetic fields) and then to appropriate power supply settings. The local bump, betatron tune control and chromaticity control classes were implemented using this technique.

```

THandle class
class THandle
{
    .....
    PTChannel*    InVector;
    int           sizeIn;
    PTChannel*    OutVector;
    int           sizeOut;
    .....
    virtual void  Connect(void* );
    virtual void  Control();

    virtual void  Compute();
    virtual void  InvCompute();
};

```

Let us consider an example of a control procedure. We construct an object of the TX4CorHandle class. The information about the magnetic elements (4 X-correctors and a virtual beam position channel) is used and objects of TMSGGroup class are constructed to calculate the transport matrix between elements.

Then executing the SetValue() procedure of the beam position channel to change the beam coordinate TMSControlChannel::Control() method involves appropriate methods of TX4CorHandle. Then the necessary simulation information is calculated (transport matrix and matrix binding the beam coordinate and

angle with the strengths of the correctors in this case). After that we can execute the Control() procedure for these correctors. This procedure uses objects of the TSimpleHandle class for binding to specific power supply control channels. The TX3CorHandle and TX4CorHandle classes support local bumps using both 3 and 4 correctors.

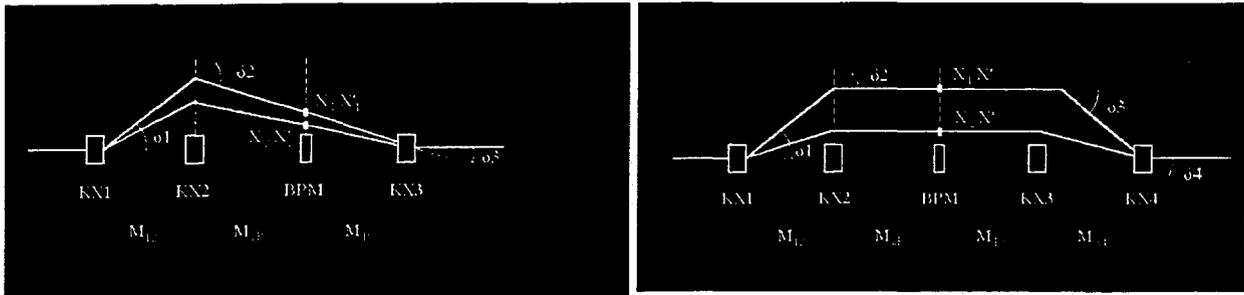


Figure 2. Three- and 4-corrector local bumps.

### 3. EXAMPLE

The library was successfully ported to the HPUX platform at the NSLS facility. The software complex was developed for on-line modeling. It is based on the GLISH software bus [2] which provides interprogram communication via events. The programs include:

"Readdbf" - reads the description of a MAD file and sends ring structure information to GLISH;

"Model" - gets a structure from MAD, calculates Twiss parameters of the ring (for each element in-out or for a selected group of elements with defined step in s-direction), gets readback information from the control system for energy and values of the quadrupole lenses, updates Twiss parameters, sends information to GLISH;

"QHandle" - the manager for tune on-line modeling - requests the MAD file, sets on-off status for readback, displays working-point tune diagram;

"ReadMes" - receives requests for getting readback of tune measurement values, sends information to GLISH;

"Dtwiss" - the manager for Twiss on-line modeling - requests the MAD file, selects the part of the ring, gets information from GLISH, displays graphics of the Twiss parameters and the structure of the selected part of the ring. An example of a graphic implementation using Motif is shown in figure 3.

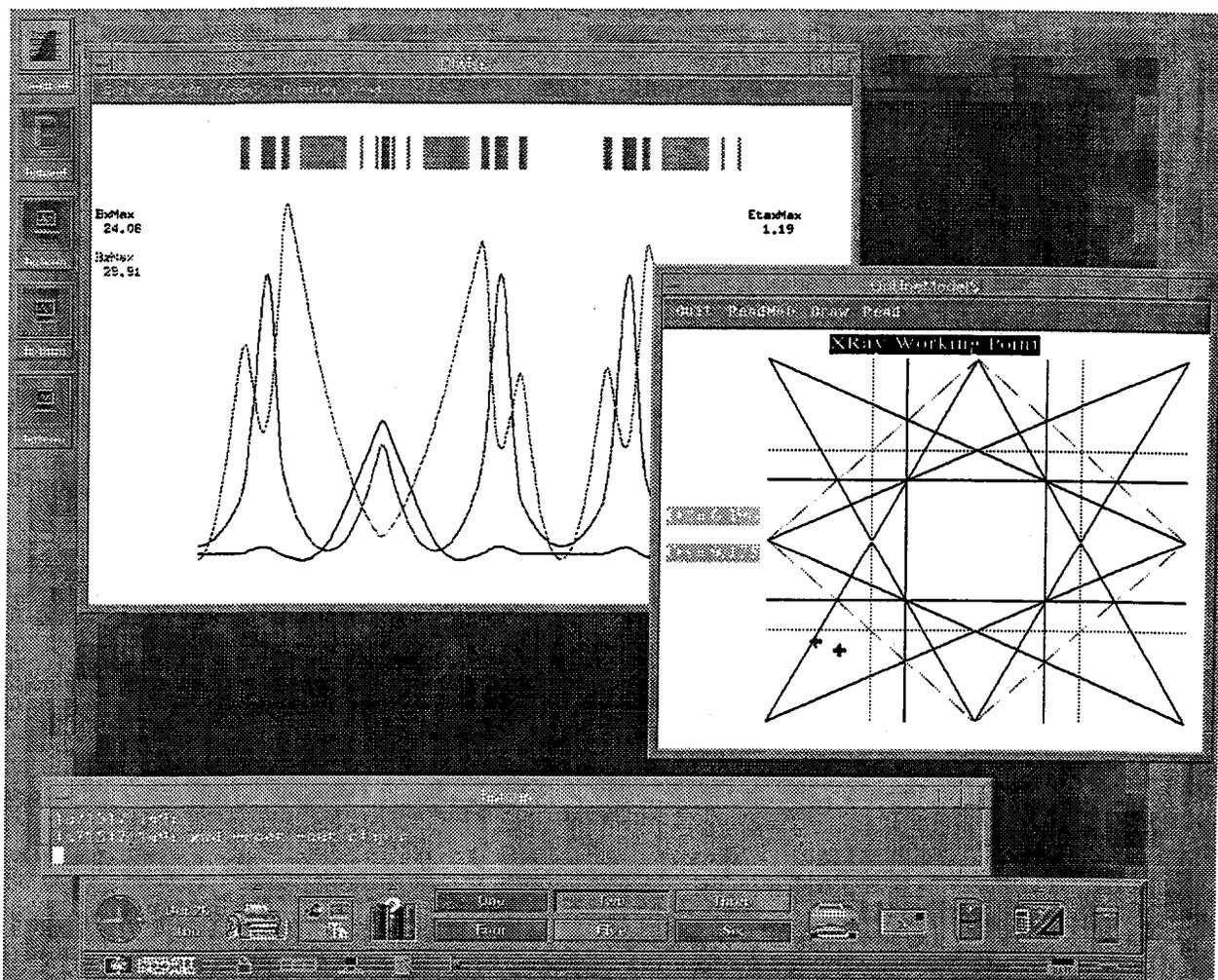


Figure 3. Example of on-line modeling.

#### 4. REFERENCES

- [1] H.Nishimura, "Dynamic Accelerator Modeling", Proc. of PAC'93, Washington, 1993, pp.111-113.
- [2] L.Schachinger, V.Paxon "A Software System for Modeling and Controlling Accelerator Physics Parameters at ALS", Proc. of PAC'93, Washington, 1993, pp.1940-1942.
- [3] M.Bickley, et al., "Orbit Correction Implementation at CEBAF", Proc. of PAC'93, Washington, 1993, pp.1895-1897.
- [4] M.Plesko, "A Complete Data Structure for High Level Software of ELETTRA", Proc. of EPAC'94, London, 1994.

# A RULE-BASED CONSULTANT FOR ACCELERATOR BEAM SCHEDULING USED IN THE C.E.R.N. PS COMPLEX

J. Lewis, P. Skarek, L. Varga<sup>1</sup>  
PS Division, CERN, CH-1211 Geneva 23, Switzerland

## ABSTRACT

The CERN PS accelerator complex consists of nine interacting accelerators which work together to produce particle beams for different end users, varying in particle type, energy, time structure, and geometry. The beam production schedule is time sliced and depends on the current operational requirements and dynamically on the accelerator status, so that production schedule changes occur in real time. Many potential schedules are not valid due to various system constraints and these constraints vary over time as new operational modes are introduced. In order to ensure that only valid schedules are given to the complex, an automated tool has been developed to indicate whether a potential schedule is valid or not. This presentation describes the method by which the validity of a beam schedule is determined and how this method was implemented using a rule-based approach based on SQL, avoiding the use of an expert system shell. Both the data to instantiate the rules and the rules themselves are kept in an Oracle data base. The SQL interpreter provides the inference engine for this knowledge-based system. A few examples are presented and the running experience with the tool is discussed.

## 1. INTRODUCTION

The CERN PS complex is a network of nine accelerators, which has been described as a factory for producing particle beams [1], [2]. Each accelerator is scheduled in 1.2s time slices called basic periods and works on the same beam for one or more of these periods, which constitutes an accelerator cycle. Every cycle requires a characteristic setting of the accelerator parameters (Pulse to Pulse Modulation [1]), which is chosen according to a telegram message sent to each accelerator at the beginning of the cycle. The beams produced by the PS accelerator network may thus be considered as lists of cycles to be executed sequentially on a set of its accelerators. These beams can be replaced in real time by an alternative beam set, depending on a set of external conditions representing each accelerators hardware status, and the current operational requests. The operational requirements for the complex are input to an editor via a Graphical User Interface as a time-ordered set of beams and alternatives called the Beam Coordination Diagram (BCD) [2]. The BCD is then executed by the Master Timing Generator according to the external conditions.

## 2. BEAM SCHEDULES

Although many BCDs can be built with the editor, not all of them can be executed correctly by the complex because the physical properties of the accelerator equipment pose certain constraints. Some constraints are concerned with individual cycles in isolation, while others are more complex and are concerned with cycle sequences called supercycles. The BCD Editor enforces some simple constraints on cycle and supercycle composition, but since this is not enough to guarantee an executable BCD, a second level of checking was required, Fig-1. For example:

- *A PS user LEAR (Low Energy Antiproton Ring) containing the PBAR (Antiproton) option should follow a cycle longer than 1.2s*

A constraint concerning the minimum ramp time for some power supplies involved in transferring the beam and hence a restriction on cycle sequence.

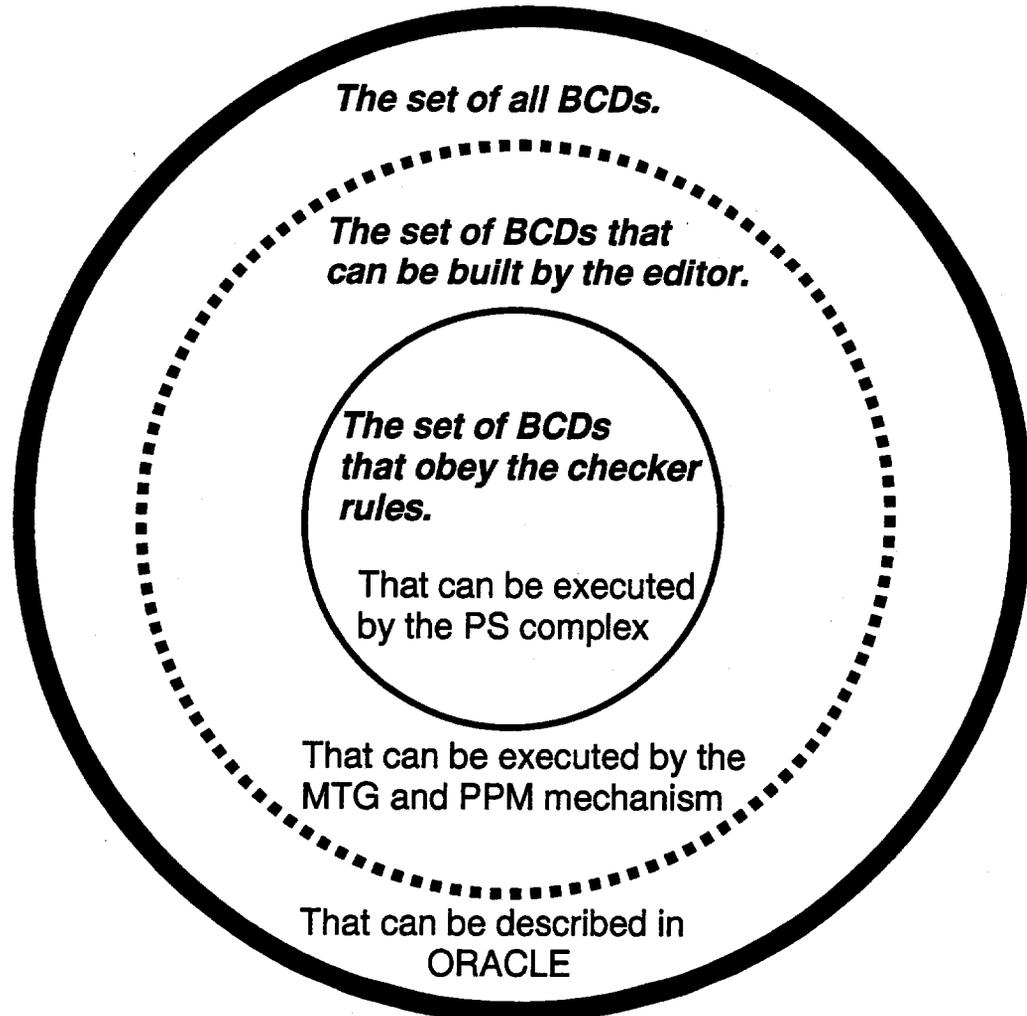
---

<sup>1</sup> On leave from K.F.K.I.-M.S.Z.K.I. , Budapest, Hungary.

- *Two PS users AA (Antiproton Accumulator) should be separated by a time difference which is a multiple of 4.8s*

A constraint concerning average power consumption and hence magnet temperature, giving a cycle sequence restriction.

**Figure 0-1 Different levels of constraints for the BCD**



### 3. A RULE BASED APPROACH

The requirements discussed above could not be satisfied easily by a traditional program, so we took a rule-based approach, but we did not want to have to maintain a full expert system shell and fortunately the constraints do not need rule chaining, because the rules are independent. The BCDs and the rules defining the constraints are stored in an Oracle database and we use the SQL language to check that the BCDs satisfy these constraints. However the direct use of SQL is not recommended for an operator and some concepts, such as the time delay from one cycle to another require complex SQL expressions. We developed a rule language which is similar to SQL and has special expressions for the BCD concepts. This approach led to the development of two programs: the BCD rule checker and the BCD rule editor. The rule editor allows the user to create rules using the formal rule language and compiles them into SQL expressions to be used later by the checker. The SQL interpreter plays the same role that an inference

engine plays in an expert system shell and the knowledge base, the other essential part of an expert system, is the set of rules stored in Oracle together with the properties of the BCD used in the rules to instantiate the rule variables.

The constraints on valid BCDs can be expressed by rules which have four text fields: the informal text, the formal text, the warning text, and the SQL text. [Fig-2] The SQL text is normally not editable, it is automatically generated by the BCD Rule Editor program and the BCD Rule Editor normally does not show this part of the rule to the user. The BCD Checker program uses this SQL text to check the BCD. The other three text fields can be edited using the BCD Rule Editor. The informal text and the warning text parts have no syntax restrictions. The warning text usually describes in one sentence the problem and is displayed with the results and the informal text describes and comments on the rule. The formal text field contains the rule itself and must satisfy a syntax which is checked when the rule is saved and the BCD rule editor generates the SQL text from it.

**Figure 0-2 An example of a Rule**

<p><b>Rule no:</b> 2</p> <p><b>Date:</b> 3rd Aug 1995 10:30 AM</p> <p><b>Enabled:</b> Yes</p> <p><b>Informal text:</b> A PS user LEAR contains a harmonic different from 10</p> <p><b>Formal text:</b> cycle set A exists where ('CPS' = A.machine and 'LEAR' = A.username and not 'H10' = A.plsgroup.HARMN) ;</p> <p><b>Warning text:</b> Wrong harmonic number for this user</p> <p><b>SQL text:</b> select A.cyclemachine, A.type, A.cyclestart from tmp1 A          where 'CPS' = A.cyclemachine and 'LEAR' = A.username and not 'H10' =          (select t.plsgroupvalue from temp2 t where t.bpkey = A.bpkey and t.plsgroup = 'HARMN')</p>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The formal text imposes conditions on a 'cycle set' and if there is a set of cycles that does not satisfy certain conditions then a warning has to be given. In the above example the name of the cycle set is 'A' and the cycles of this set can be referenced later in the condition part using this name. The condition is given after the 'where' symbol and it is a Boolean expression. Cycle properties used in equations are formed by concatenating the name of the cycle and the name of the property with a dot. For example A.machine is the machine in which the cycle is executed or A.username is the user of the cycle. An example of the value that the A.username property can take is: 'LEAR'.

#### 4. CONCLUSION

We have developed and implemented a technique for checking the validity of a beam schedule. This technique gives the flexibility of a rule-based system, but does not have the overhead of an expert system shell. The implementation includes a rule description language, a user-friendly rule maintenance and editor program and an inference program that applies the rules and presents the results in graphical form.

The first few months experience has shown that the users can create BCDs with a higher reliability. The rule-based approach has been proven to be sound and gives enough flexibility to express most constraints on BCDs. New rules can easily be created by the users. The speed of the inference program (in fact Oracle access and SQL interpretation) is a limitation, even though there are no real time requirements. Steps will have to be taken in the future to combine all the rules into a single optimized SQL query. For the future we consider it important that this

diagnostic tool become part of the general environment for timing diagnosis and recovery actions [Ref. 3].

## ACKNOWLEDGMENTS

We acknowledge the help of I. Campos during her stay at CERN on leave from DFTUZ University of Zaragoza, Spain, in helping with coding and documentation.

## REFERENCES

- [1] F. Perriollat, C. Serre: "The new CERN PS control system overview and status", ICALEPCS, Berlin, Germany, October 18-23, 1993, Nucl. Instr. And Meth. A352 (1994) 86.
- [2] J. Lewis, V. Sikolenko: "The new CERN PS timing system", ICALEPCS, Berlin, Germany, October 18-23, 1993, Nucl. Instr. And Meth. A352 (1994) 91.
- [3] P. Skarek, L Varga: "Multi-Agent Cooperation for Particle Accelerator Control", Accepted for presentation on the 3rd World Congress on Expert Systems, Seoul, Feb 5-9 1996.

## Good Screen Design

Richard Rothrock and Virginia Martz  
Vista Control Systems Inc.

### ABSTRACT

This paper reviews the elements of good screen design in relation to designing a Man Machine Interface (MMI). The key principles in accomplishing a successful design are knowing the user and empowering the user. Fast prototyping helps the designer know the user and the ability to create well designed screens empowers the user. Good screen design includes setting up local standards, proper use of color, using effective layout rules, ensuring adequate screen response, managing complexity, and maintaining consistency. This paper examines these elements and how they relate to factors such as intended operators and room environments.

### INTRODUCTION

Creating the operator interface or the man machine interface (MMI) for a control system can be a rewarding, creative and satisfying effort. A job well done contributes greatly to the productivity of the controls effort. A poor MMI becomes very frustrating, hard to use and may prevent the users from accomplishing their control system goals. This paper provides some guidelines for creating a good MMI. The two key principles for this effort as stated in the *OSF/MOTIF Style Guide* [4] are knowing and empowering the user.

### KNOW THE USER

Knowing your user is the job of every professional. Users are the operators, customers and people that will eventually sit or stand in front of your screens for hours and hours, keeping their machine running. In all endeavors we hear the same phrases: know your user, learn to communicate, write specifications, and iterate ideas so that users get a close approximation to, or maybe even more than, what they wanted. For good screen and MMI design, in order to know your user, you must use an iterative design process with the user during the development of the MMI.

Iterative design is essential to good screen design, especially when users are coming from an older MMI based upon character cell terminals. These iterations help teach the users what is possible with newer graphic-based MMIs and is accomplished most efficiently with a rapid prototype screen builder. When you show users what is possible, they can better communicate their requirements to you. Several iterative designs should be done even during the initial demonstration of the preliminary screens in order to show the users that changing screens is easy. In the past, changing control screens was a very time-consuming effort and was requested only when absolutely necessary. With new MMIs, users themselves have the ability to create preliminary screens which will enhance the design process greatly. Later on, users will also have the ability to maintain production screens.

### EMPOWER THE USER

The main reason for good screen design is to enable users to do their jobs. If your iterative design process is successful with the user, you could come up with an MMI that meets all the requirements but still is ineffective because the screens are busy, too colorful, or do not get to the heart of the job. If the screens distract rather than inform, or you need to navigate into too many screens to get the job done, the effectiveness of the MMI can be lost. The following sections describe some rules that make it easier to create an effective MMI. Well designed MMIs make a good first impression and still are easy to look at and use after a full shift and after years of running a facility. Remember that an effective MMI empowers the users by minimizing the scan time required to assimilate the information needed to do a task.

### MINIMIZE COLOR

One main rule to follow is to minimize the use of color by using grayscale in most of the windows. Resist the temptation to play with the many great colors now available on workstations. When the job is to control a physics experiment, the technicolor (wild random use of color), screen furniture and visual toys are very distracting. Bright, busy and cheerful images should be left to games; minimal use of color is the name of the

game for MMIs. An exception to this rule is photorealistic images, which may have many colors; this realistic use of color is very natural and not distracting.

To get a subdued look, do all preliminary screen design first using only grayscale, then add color only where it adds information and clarity. For example, a standard could be that all help screens have a muted green background and all operations screens have a light gray background. The muted green provides a visual cue that distinguishes help screens from other screens. If all other screens had randomly colored backgrounds, the green background would be meaningless.

#### **USE COLOR EFFECTIVELY**

Knowing how to use color effectively starts with understanding the physiology of the human eye. Color results from the interaction of certain frequencies of electromagnetic radiation with an appropriately designed nervous system. The eye is the sensor that perceives this radiation using the lens and retina. More brain power is dedicated to sight than all the other senses combined. A simplified diagram of the eye is shown in Figure 1.

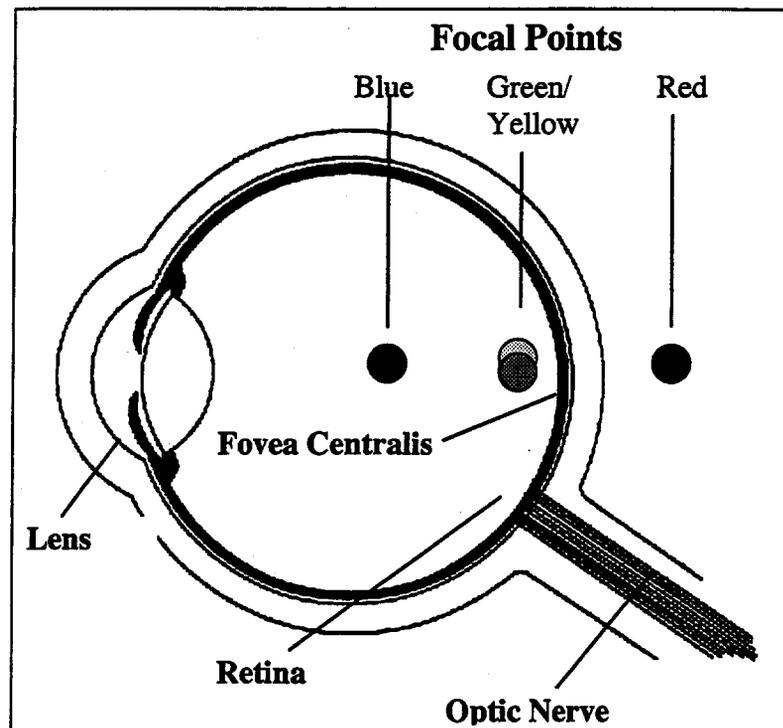


Figure 1. Diagram of the eye

Keep in mind the main function of our eyes is survival of the human species, not looking at a color monitor. At night our eyes become incredibly sensitive to light to allow us to see in the dark. Our eyes also are sensitive to rapid changes in light. Figure 1 shows that you can only focus on red, green/yellow, or blue at a single time. However, a color monitor presents all colors at the same focal depth, causing the eye to refocus to each color as the fovea scans the screen to perceive detail. These characteristics must be considered in developing an effective MMI.

Users find it easy to remember combinations of seven, plus or minus two, items, as in telephone and social security numbers. This rule can be applied to color as well. Meanings should be assigned to no more than seven to nine colors. To insure consistency across the MMI, develop a color standard to keep the meaning of colors consistent throughout the MMI. Users will make an implicit assumption that all things colored the same are associated in some way. When colors are used for meaning, the associated help screens should have color legends describing this meaning. When choosing the meanings of colors, keep in mind commonly held associations of color and meaning, as shown in Figure 2. The meanings of words must be carefully chosen, depending upon the application. Notice that the word "on" appears as a meaning for both the color Red and Green. In some applications, if the color is closely associated with a dangerous piece of equipment, "on" should

mean red to denote danger and green would mean "off" or safe. If "on" is not closely associated with danger, then green is often used for denoting "on," red is then reserved for an alarm condition, and blue or gray would

Color	Meaning
Green	Go, on, clear, safe
Red	Stop, hot, danger, fire, on, emergency
Yellow	Caution, slow, warning, warm
Blue	Cold, off, calm
White	Empty, closed
Black	Full, open

Figure 2. Common color meanings

mean "off."

Picking harmonious colors can be made easier by examining a color wheel and looking at different types of color combinations. The color wheel in Figure 3 shows relationships between colors. Similar colors, adjacent to each other on the wheel are useful for shading and showing changes in levels. Complimentary colors fall directly across from each other, and they make good combinations, because they are less distracting and easier to focus on than similar colors. Distinct colors, found in symmetrical positions around the wheel, are good for showing changes in state. Highly saturated, spectrally extreme colors (like pure red and pure blue) next to each other should be avoided. These colors cause eye fatigue, because the eye constantly needs to refocus on one of the two colors. If they must be used together, a black border should be used between them. Since blue is a hard color for the eye to focus on, it is not a good color to use for text, thin lines, and small shapes, but blue does make

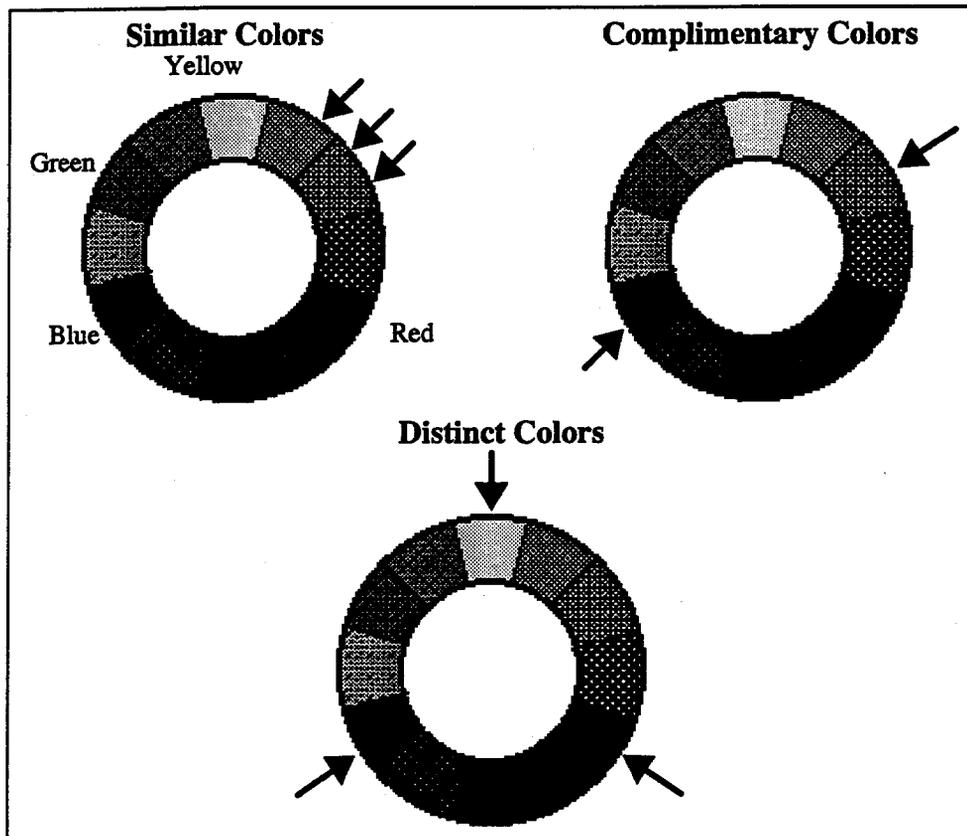


Figure 3. Color relationships

an excellent color choice for backgrounds.

Choosing a color standard often can be a difficult task in developing an MMI. You can obtain ideas for effective colors from many places. For example, look at the color combinations of basketball and football uniforms. Notice the color of text used with the background color on the jersey; it is black on light colored backgrounds, and white on dark colored backgrounds. Another source for good color combinations is Microsoft's window color combinations. Notice that the colors are subdued and very close to each other on the color wheel. The default colors, grays and blues, allow applications to stand out more in color. The "Arizona" selection uses complimentary blues and greens with gray. The "Black Leather Jacket" selection is a good example for darker rooms. The "Florescent" selection is hard to focus on for long periods of time, but it allows the user quickly to recognize the active window. The "Hot Dog Stand" selection has very strong edges made from very saturated colors taken from one side of the spectrum (reds and yellows and not blues), a selection resulting in less eye strain.

When choosing a standard set of colors, don't forget the occasional color blind users. The most common form of color blindness is the inability to distinguish between red and green. Color alone should never be used to convey a change. Use changes in color and text, or color and shape, or vary the contrast of the color as the color changes. If this idea is extended, a screen is not rendered useless if a color gun in the monitor fails. If printing is important, try printing your standard colors to ensure that colors are discernible in both color and grayscale.

#### **CONSIDER ENVIRONMENTAL CONTRAST**

Another factor to be considered when developing good screens is the room environment. Light conditions in the control room should be taken into account when locating monitors, to reduce glare on the screen and lower contrast. As the user's eyes move around the room and to the computer monitor, the light intensities between the room and the screen should be as close as possible to reduce eye strain. If you have control over this environment, the ambient light intensity should be set to around 500 LUX. A standard bright office is around 1000 LUX, which would be difficult to match with a computer monitor.

In other words, if you have a dark room, use dark colors for the majority of the screen area with light graphics and text. Similarly, with a bright room, use light backgrounds with black text. This method will lower eye strain by taking into account eye sensitivity to light. If the monitor is portable, determine all lighting conditions where the MMI may be used; if necessary make two sets of background and text colors available, and switch them dynamically to match the lighting conditions. The key colors that carry meaning should remain unchanged and be designed to work in all situations. Some MMIs have this capability by defining multiple color maps for use in different lighting conditions.

#### **USE TEXT EFFECTIVELY**

The proper use of text can do much to improve screen effectiveness. Select only two or three font families for use throughout the MMI. Use standard point sizes to show different types of information. Use a large point size for screen titles, with a slightly smaller point size for labeling groups of information and yet a smaller point size for labeling data. Point sizes should be based on how far from the screen the users will be. With few exceptions, text should be white or black. If it is necessary to use color for text, a bold font from the same family should be used to maintain a contrast similar to the normal text. When labeling data and titling screens, only the first letter of each word should be capitalized. Users tend to identify words by their shapes; therefore words made



Figure 4. Text shape

up entirely of capitals do not vary in shape from word to word and thus take longer to identify. Figure 4 demonstrates the added clues the shape gives by viewing only the top half of both sentences.

#### **USE LAYOUT RULES**

Consistent layout choices for objects also can enhance the effectiveness of an MMI. A standard should be developed for the placement of objects. Aligning rows and columns of objects reduces the complexity of a screen, which also reduces the scan time for a user. Another layout rule is to reduce the density of the screen. A window should be made up of 70% to 80% white space if at all possible. Again, this use of white space makes it easier for the user to scan the display and gather information quickly. Reduce information density by presenting lists of values in columns with appropriate labels and by grouping similar objects with separators such as boxes or lines.

### *CONSIDER SCREEN RESPONSE*

Operator feedback is another important part of an MMI. The computer should respond immediately to all actions by a user. This response can be a color change, a wait cursor (if the action will take time to perform), a popup informational window, or some other feedback method that informs the user that their action has been recognized. It is very frustrating to request an action and not have anything happen, even though the MMI has received the request. Often this lack of response causes the user to repeatedly request the same action and stall the requested action or toggle unnecessarily.

Consider, too, the update rates of information displayed on a screen. A user can recognize and process information changing on a screen from 5 to 30 Hz. Changes more frequent than this are unnecessary, because the user will not perceive data changing more often and the MMI will waste computer cycles. The movie industry has found 30 frames per second adequate to show smooth and uninterrupted change. It is recommended that the MMI throttle changes more frequent than 10 Hz to prevent overloading of the graphics server and to insure fast response for the user. Reserve higher rates (up to 30 Hz) for showing smooth animation when necessary.

### *SELECT TILED VS OVERLAPPING WINDOWS*

Another decision to be made is which windowing method should be used: tiled or overlapping windows. A computer-controlled tiled system determines where and what windows should be placed on the screen. The user is not allowed to size or move individual windows. This method is the least flexible, but the easiest to learn by inexperienced computer users. Tiled windows present less complexity; use them when users are nonprogrammers whose interactions with the computer are peripheral to their work. Overlapping windows can be entirely user controlled. Users can choose which windows should be opened, where they are placed, and how large they will be. Overlapping windows can be dynamically rearranged to suit the user's operational needs, depending upon what task is being performed and which windows will most empower the user to perform the task. A user controlled overlapping system is the most flexible, but the hardest to use and much less determinant.

The disadvantages of tiled windows are that the users cannot configure the windows to meet their needs. Overlapping windows have the disadvantage of giving the user too much control over the number of windows open at one time. This can lead to windows being lost behind each other, increased complexity and overloading the graphics server. Choose one method and use it for the majority of the MMI.

### *DESIGN ICONS*

When possible, give information to the user graphically instead of textually, or both together for more lexical-based users. Graphical information can be processed much more quickly, allowing faster recognition than pure lexical based information. Using well designed icons also saves space and reduces complexity. When choosing icons to represent data, select symbols that are meaningful and distinct from other icons in the MMI. Keep in mind that the computer culture is currently building a common set of recognizable icons (the trashcan, and the phone, for example). Use commonly recognized icons when possible, to give the user less to learn.

### *RULES FOR DIALOG WINDOWS*

There are two main types of windows in a control system: display windows where the users perform primary tasks and dialog windows which are secondary windows intended to appear temporarily. Dialog windows ask a question or show detailed information needed for a minor step in a task. They should have a consistency all their own. For example, the cancel button should put the dialog window away without any action taking place, and it should be in the same spot on all dialog windows. Dialog windows should be as small as possible and placed effectively in order to show the orientation of where it came from when the user answers the dialog. These windows should be designed so that the user must answer the dialog before continuing a task. Give careful consideration as to when an additional dialog window is necessary and when dialog windows can be combined to

limit the frequency of processing individual windows. If too many windows are thrown at the user too frequently, the user begins to ignore the messages and the meaning is lost.

#### ***PLAN FOR EXPERIENCED USERS***

As users become more familiar with the MMI, they become curious and exploratory. As users explore, they should be able to back out of their actions without making permanent changes to the system. Warnings about destructive actions allow the user to avoid making a permanent change and encourage curiosity. More experienced users tend to repeat familiar actions and desire accelerators for performing repetitious work. Excessive warning beeps from the MMI are ignored by experienced users. A single beep should be employed if the user tries a sequence that is not correct at the time.

#### ***TOUCHSCREEN RULES***

Touchscreens are becoming more common in industry with improved cost/performance. Touch panels have become easy to press and are transparent. The important design considerations to be aware of are touch area size and placement. Touch area size should be set to 2 cm minimum. Place most frequently used touch areas along the bottom and right side of screen, to minimize the amount of screen obscured during a press (remembering that most users are right handed). A good reference for touchscreens is the Sematech standard, which is an industrywide standard for semiconductor manufacturing [5].

#### **CONCLUSION**

Good screen design can be a learned skill; with careful attention to color and text choices, layout and environment. Plan on creating standards and prototyping early in the design process. As the iterative design process develops, it can be useful to keep a storyboard of the screens, to monitor your progress, see improvements, and keep track of your goals. The references are recommended as helpful tools for building a complete, easy-to-use and interesting graphical user interface.

#### **RECOMMENDED BOOKS**

- [1] Foley, James D. et. al. *Computer Graphics Principles and Practices*. Reading, MA: Addison-Wesley Publishing Company, 1990.
- [2] Fowler, Susan, and Victor R. Stanwick. *The GUI Style Guide*. Boston: Academic Press, 1995.
- [3] Horton, William. *The Icon Book*. New York: John Wiley & Sons, 1994.
- [4] Open Software Foundation. *OSF Style Guide*. Englewood Cliffs, NJ: Prentice Hall, 1989.
- [5] Sematech. *SCC User-Interface Style Guide*. Austin, TX, 1992.

# Object-Oriented Control System Development Using the Smalltalk Language

I. Mejuev, I. Abe and K. Nakahara

National Laboratory for High Energy Physics, 1-1 OHO, Tsukuba, Ibaraki 305, Japan

## Abstract

An approach to and the tools developed for accelerator control system creation are presented. The Smalltalk language environment provides a set of unique possibilities for solving this problem. The most important is the dependencies facility, which allows one to divide the domain model and user interface development. Our method uses the Domain Model Editor tool for developing an object-oriented model of an accelerator. The Chart Editor is used to define graphics for representing the accelerator objects states. The model and chart are stored in an object-oriented database. Using the database file, the Control Panel program scans the model conditions for the execution of operations and displays the state dynamics on a chart. The domain model can be linked to custom-user interface objects in order to implement control at runtime. The user interface objects are created using either the Control View Editor tool or VisualWorks environment. The problem of the control distribution over the computer network is also discussed.

## 1. Introduction

This paper represents our approach to creating accelerator control systems. Any control system that meets modern requirements should give users the possibilities to easily create and modify their accelerator model using convenient interface tools. For this purpose we have developed the Domain Model Editor, which describes an accelerator using a set of predefined standard classes. Objects created by the user are saved in the object-oriented database for later use. Also, the Domain Model Editor includes the possibility to create new user-defined classes. The domain model created with this tool comprises all of the data necessary for accelerator control.

At runtime it is important to display and update information concerning the current system state. For this purpose graphical representations of the states of the most important objects in the system are necessary. To manage such graphics and map graphical objects to model objects the Chart Editor tool is used.

While the Model Editor and Chart Editor are used in the model-creation phase, the Control Panel is a tool used at runtime. It implements a real control loop to check the conditions for executing system operations. Another part of the system is the Control View Editor, which creates GUI objects to implement custom control tasks, such as the direct execution of operations, the control of values and the providing of detailed information about the system objects.

In our system we widely use the dependencies facility of Smalltalk [1,2], which allows us to divide model development from graphics and user-interface tools development. Usually an interface object is registered as being dependent on the model object, so that it can handle update requests.

## 2. Domain model

In our system the Domain model of the accelerator is based on a set of fundamental classes. Those classes implement the basic features that the accelerator model contains. By using inheritance a user can choose the exact implementation and system decomposition that best fits his purpose. The main classes allow the user to describe object aggregation, to assign operations acceptable for objects and to define the conditions for operation execution. The condition equations use special kinds of objects: *values* which can be connected to equipment for measurements. The results of object-oriented analysis were first reported in [4].

*ControlObject* is a root system object; any object in the domain model belongs to this class. *ControlObject* properties are:

*name* (name used to identify the object, such as "Vacuum system"), and

*state* (an object state which can be displayed on the states chart [5], such as “Normal” and “Alarm”).

Also *ControlObject* contains associations for supported *operations* and includes *conditions*, *values* and *component objects*.

Operation is an instance of the *ObjectOperation* class. A user can define operations for different object types. For example, there can be operations that are applicable for klystrons, vacuum pumps and so on. Any operations should be first defined in an object class. When an object is created, a class constructor associates the object with the correct operations.

The condition (instance of *Condition* class) defines the rules for the execution of operations. Every condition is created from some abstract condition specification, like  $a < x < b$  with the addition of a parameters list and association to an operation. The condition specification, together with parameters, forms a logical expression. Such an expression is evaluated at runtime while polling and if it is valid, the operation is executed. Another property of the *Condition* class is polling type. It specifies whether it is necessary to create a special polling process for this condition or to use a common one. Condition arguments can be either constants or *values*.

A value is a generic datum holder, an instance of the *ObjectValue* class. The value magnitude corresponds to the *value* property of this class and value type (string or number) to the *valueType* property. To retrieve and set the value, *setValue* and *getValue* messages are used. These messages redirect the request to low-level software connected to the equipment; for instance, it could be the input-output channel for a VME slot.

*Accelerator*, *Subsystem* and *SubsystemObject* classes are inherited from the *ControlObject* class. *Accelerator* is the main root object; it contains a set of subsystems and an object states chart description [5]. The *Subsystem* is an object container for creating an rf system, vacuum system and so on. It may contain a set of global subsystem values. A *SubsystemObject* is used to represent objects which are subsystem components, such as klystrons and pumps.

These classes form a basis for a control system model. If necessary, a user can refine the domain model by using an inheritance. The basic classes for system design are presented in Figure 1 in Rumbaugh notation [3].

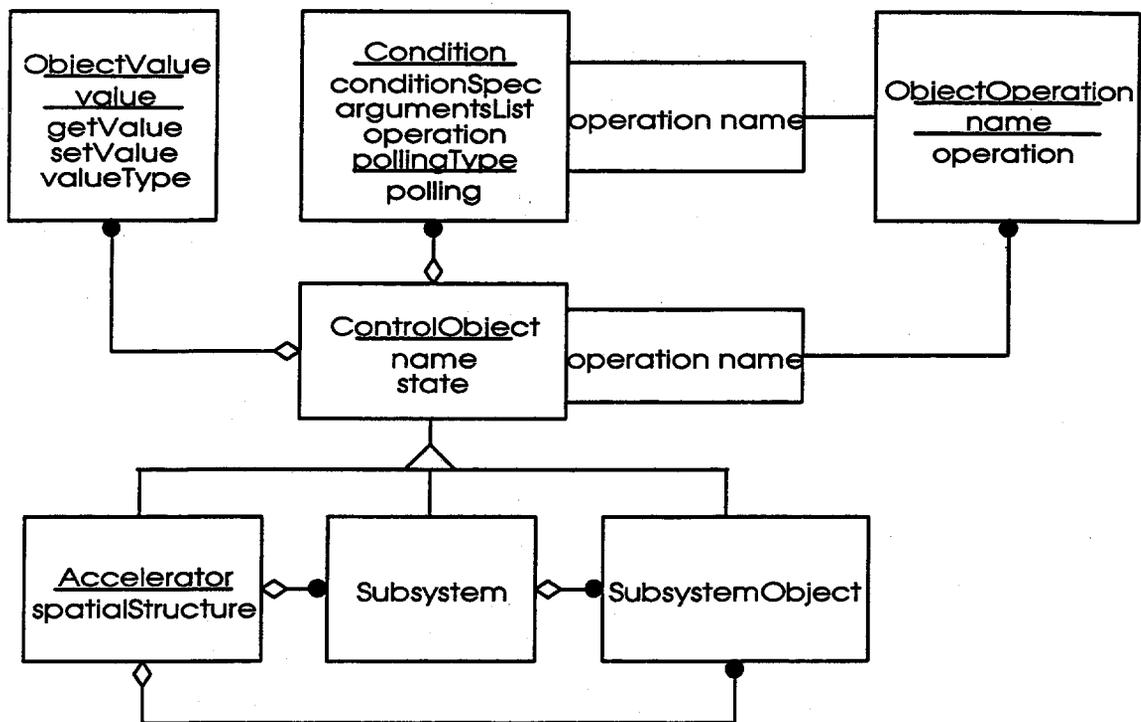


Figure 1: Basic system classes

### 3. Control loop

The control loop polls all of the model conditions. It is implemented by the Control Panel tool. The Control Panel uses an object database created by the Domain Model Editor. It sends message *polling* to all conditions from top-level *Accelerator* objects to low-level *SubsystemObjects*. The *polling* method checks the logical expressions for operation execution, and executes operations when necessary. Also, the Control Panel draws a states chart and establishes links between the domain model objects and chart elements [5].

The Control Panel creates the main polling process as well as auxiliary processes for conditions with corresponding polling types. Using additional polling processes allows one to increase the polling rate of the most important system parameters and to improve the system performance. Smalltalk language contains built-in process control possibilities which simplify the development of the Control Panel.

### 4. Tool relationships

The system core is an accelerator domain model which is stored in an object-oriented database. We use the VisualWorks BOSS file (Binary Objects Streaming Service) for this purpose [2]. However, before storing an object using BOSS it is necessary to remove any dependencies so as to exclude the possibility of an infinite recursion. The BOSS database is used by all system components as a control data repository.

The BOSS file is created by the Domain Model Editor. This program includes possibilities for creating objects as well as classes. It is possible to operate with values, operations and conditions on both the class and object level. An example of the objects' view is shown in the Figure 2 and the classes' view in the Figure 3. Classes play the role of object factories; any features implemented on a class level can be easily replicated. However since by using the Domain Model Editor there is a possibility to customize an object by adding extra values and operations without modifying the class definition, its use should be constrained, since it contradicts the conventional methods of Object-Oriented Design.

A part of the Domain Model Editor is the Chart Editor, which is used to create a states diagram and to connect model objects to chart elements. The Chart Editor just assigns object references to chart element variables. Real dependencies are established at runtime by the Control Panel. The Chart Editor tool is described elsewhere [5].

The Control Panel program is a runtime tool, which implements real control using data created by the Domain Model Editor. The Control Panel starts and stops the conditions scanning processes and also presents a chart with current object states. However, it is a self-contained program which doesn't support the initiation of operations by a control system operator. To build tools for operator control the Control View Editor is used.

The Control View Editor creates data for *control view*, a program used for operator control. Using the Control View Editor, a set of GUI objects can be created, such as: ControlObject value views, ControlObject value controls (sliders, combination boxes), action buttons, action lists and action menus.

The *control view* works together with the Control Panel and shares the same data using a BOSS database file. If the standard set of objects is not sufficient, a control view can be created using the VisualWorks environment. In this case the programmer is responsible for creating an interface between the control view and the domain model.

### 5. Distributed control

Large control systems can have a large number of values to control; they also can be territorially distributed, so it is necessary to have support for distributing control tasks over the computer network. The main goal is to give computers the possibility of using objects from the models of other computers, requiring value access and remote operation execution. For this purpose we use *remote objects*, which are present in the local computer model, but are really implemented on the remote computer (host computer) domain model. Every remote object includes the name of its host computer. When a remote object value or operation is required, the request is redirected to the host computer by a network dispatcher.

By using the remote objects, condition scanning work can be distributed over a set of computers. Each database representing a part of the system can be placed on a different network segment. For a local database

the BOSS file is sufficient. To take advantage of client-server technology, we can also use SQL servers. VisualWorks currently supports Smalltalk objects mapping to Sybase and Oracle databases.

## 6. Conclusions

The approach introduced in this paper shows that Object-Oriented Design offers a wide range of new possibilities for the development of accelerator control systems. A set of tools has been developed which allow any user to easily customize the system structure. The object-oriented programming style makes it possible to reduce the system maintenance and development costs. The Smalltalk language we used includes process control, a dependencies facility and visual programming tools that aid large-scale project development.

## 7. References

- [1] An Introduction to Object-Oriented Programming and Smalltalk; Lewis J. Pinson, Richard S. Wiener, University of Colorado at Colorado Springs, Addison-Wesley, c 1988.
- [2] VisualWorks User's Guide; ParcPlace Systems, c 1994
- [3] Object-oriented modeling and design; James Rumbaugh [et al.] Englewood Cliffs, N. J.; Prentice Hall , c 1991.
- [4] Application of an Object-Oriented Analysis for the PF Linac Control System Development; Mejuev I., Abe I. and Nakahara K.; Proceedings of the 20th Linear Accelerator Meeting in Japan, Osaka, 1995, p. 212.
- [5] Graphical Representation of Objects' State for the PF Linac Control System; Mejuev I., Abe I. and Nakahara K.; Proceedings of the 10th Symposium on Accelerator Science and Technology, October 25-27, 1995, Hitachinaka, Japan, p. 292

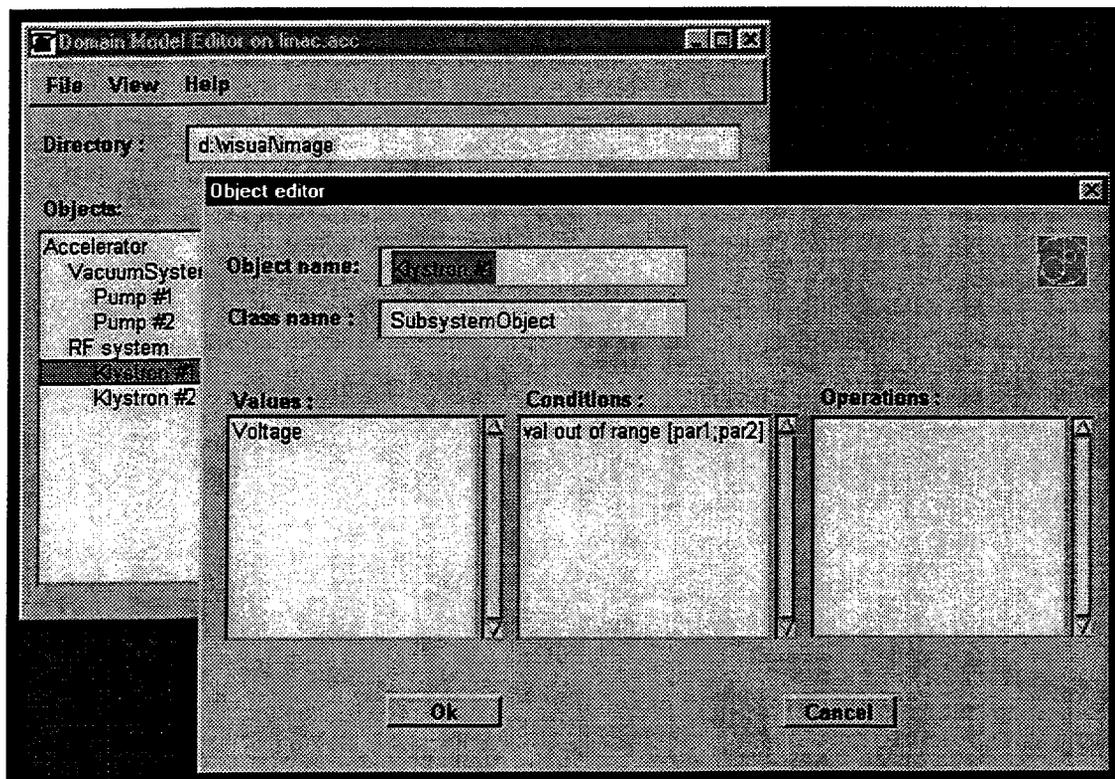


Figure 2: The Control Model Editor (objects view)

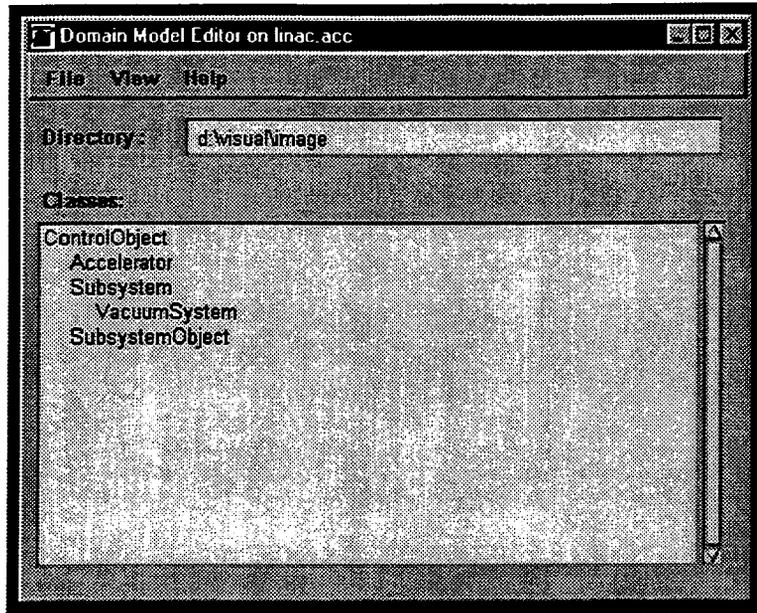


Figure 3: The Control Model Editor (classes view)

# CODE GENERATION OF RHIC ACCELERATOR DEVICE OBJECTS\*

R.H. Olsen, L. Hoff, T. Clifford, RHIC Project, Brookhaven National Laboratory, Upton, NY,  
11973-5000, USA

## Abstract

A RHIC Accelerator Device Object is an abstraction which provides a software view of a collection of collider control points known as parameters. A grammar has been defined which allows these parameters, along with code describing methods for acquiring and modifying them, to be specified efficiently in compact definition files. These definition files are processed to produce C++ source code. This source code is compiled to produce an object file which can be loaded into a front end computer. Each loaded object serves as an Accelerator Device Object class definition. The collider will be controlled by applications which set and get the parameters in instances of these classes using a suite of interface routines. Significant features of the grammar are described with details about the generated C++ code.

## I. INTRODUCTION

The accelerator controls architecture for the Relativistic Heavy Ion Collider (RHIC) being built at Brookhaven National Laboratory (BNL) uses the standard model with two conceptually distinct layers [1]: UNIX workstations provide the platform for the console-level computers (CLCs). These are networked with the geographically distributed VME systems running real-time operating systems that serve as front-end computers (FECs)[2]. The FEC supports an object-oriented paradigm using C++ object classes called accelerator device objects (ADOs). These ADOs provide a software abstraction of the underlying collider hardware.

Each ADO class comprises control points represented as parameters, and a standard set of operations with a consistent interface by which applications can examine or manipulate those parameters and the underlying hardware. The fact that the way the parameters are represented, and the interface to them, are highly consistent across ADOs allows us to gain considerable leverage through the use of code generation: a few lines of code specifying a parameter's characteristics can be used to generate the many lines of code used to implement that parameter and its interface within the larger context of a C++ class.

### A. *Adogen and .rad files*

The program used to generate the C++ source code is called *adogen*. It is itself built, in part, using two widely available code generation utilities. *Lex* and *yacc*[3] are used to generate the lexical analysis and parsing code and the remainder of the program is coded in C++. *Adogen* gets its input from files which, by convention, have a ".rad" (for **R**HIC **A**DO **D**efinition) filename extension. The .rad files are text-only files which are easily administered using any of the standard source code control packages available - we have been using RCS at RHIC.

### B. *ADO Hierarchies*

ADOs are conceptually divided into Concrete and Abstract types. A Concrete ADO can be accessed from the console level computers through libraries of routines which are used to examine or manipulate its constituent parameters. The Abstract ADOs are not directly accessible themselves, rather, they serve as a convenient way of packaging parameters. Different ADOs can share parameter collections by referencing the same abstract class as a parent. This allows for hierarchies of ADOs.

Hierarchies are implemented in abstract ADOs when a derived ADO class inherits parameters from its parent, possibly adding new parameters itself. Abstract ADOs define the data, and Concrete ADOs inherit this data from the Abstract parent(s) and can augment the methods that access the data in order to handle the specifics of the underlying hardware. Figure 1 shows an example ADO hierarchy. The base *ADO* class comprises parameters common to all ADOs, for example the *Name* and a *Description*. The *ProfileMonitor* is an abstract class which inherits those base parameters and adds *NHScanLines* and *NVScanLines* (Number of Horizontal/Vertical Scan Lines). The final abstract class *TwoDProfileMonitor* adds *Horizontal* and *Vertical* projections. In this example *Flag* is the concrete ADO. Applications can access any of the parameters which the *Flag* ADO inherits from its parents through an instance of the *Flag* ADO class, whose methods define what those accesses actually do.

## II. ADOGEN GRAMMAR

One of the primary goals of *adogen* is to allow the code for ADOs and ADO hierarchies to be created and modified quickly by specifying values for some minimum number of variables. The *adogen* grammar keeps the simple things simple. For instance,

\*Work performed under the auspices of the U.S. Department of Energy

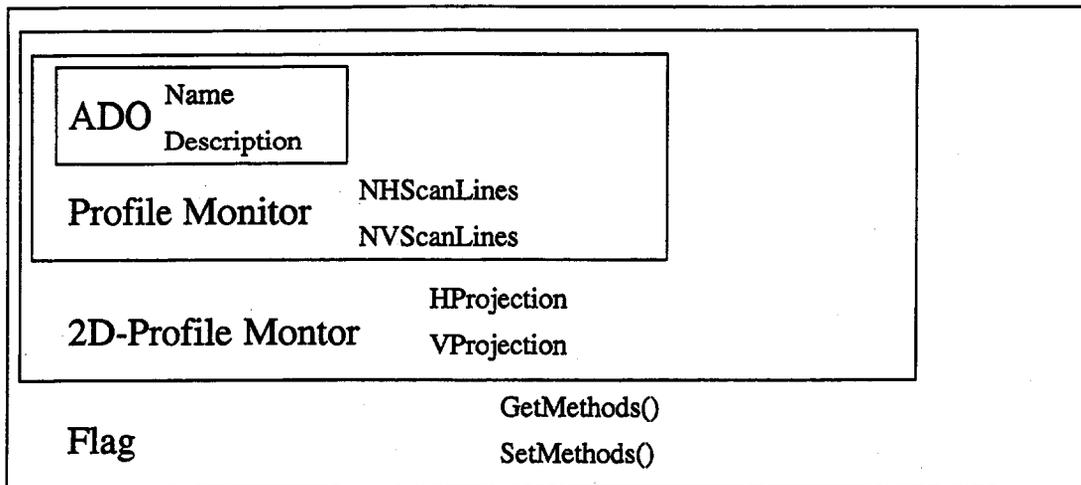


Figure. 1. Flag ADO hierarchy.

all of a parameter's important characteristics are specified in one place in the .rad file. A parameter's data type only has to be specified once and wherever the code that adogen generates is dependent on that type, consistency is guaranteed. Change that type specification and the generated code will be updated accordingly. Another goal of adogen is to eliminate the need to edit the generated code. (If there is no editing of the C++ code subsequent to its generation then the more compact rad files can become the source that is maintained in version control.) To this end the grammar for ADO definition is fairly flexible, and offers a rich set of keywords. Descriptions of some of these follow:

#### A. Keywords

- **CONCRETE** - the name of the concrete class.
- **ABSTRACT** - the name of the concrete class's abstract parent.
- **PARENT** - If an abstract class's parent is not the base ADO class, the parent name must be specified. (For hierarchies.)
- **PARAMETER** - the name of an individual parameter, and a **COUNT** if it is an array. In addition to this a complete parameter specification includes:

**CATEGORY** - Each parameter is assigned a category which determines a set of support properties for the parameter. All of the categories have support properties such as a description, and a format string. Continuous Settings add more information such as high and low limits. Discrete Settings add Legal Values, etc. The recognized categories are: **BASIC**, **CONT\_SETTING**, **CONT\_MEAS**, **DISC\_SETTING**, **DISC\_MEAS**, **CONFIG\_DATA**, **USAGE\_DATA** and **DIAG\_DATA**.

**TYPE** - a data type. The recognized types are: **CharType**, **UCharType**, **ShortType**, **UShortType**, **LongType**, **ULongType**, **FloatType**, **DoubleType**, **StringType**, **StructType**.

**READWRITE** - specifies whether the parameter is writable or just readable (R, W).

**DESC** - text describing the parameter is optional, but console level utilities know how to access this support property and it can be useful as "help" text.

**SETCODE** - any specialized set code for the parameter is specified using this keyword. While ADO infrastructure code provides default set and get codes for each parameter which accesses memory locations, any special driver or hardware level requirements for a particular parameter can be met by providing more set code.

**GETCODE** - any specialized code which may be necessary for retrieval of information from the hardware.

- **MEMBERCODE** - This code becomes member code in the ADO's C++ class. Different set or get codes can share the same member code.
- **INITCODE** - This code is executed when the ADO instance is first created.
- **THRUCODE** - This code is passed through "as-is".
- **EVENTCODE** - This code is made accessible to an Event Management subsystem. Machine events such as interrupts and timers can be "connected" to this code, which has access to the ADO's parameters and particulars about the underlying hardware, so that it is executed when the event occurs.

#### B. Syntax

The syntax for adogen resembles "C++" code. The ADO designers at RHIC are familiar with C++ and quickly became comfortable with the requirements of adogen. The same development environment can be used for either C++ or adogen grammar. Since line

number information from the .rad file is included in the generated code, compiler error messages can be processed and debuggers can reference code in the .rad file rather than in the generated code.

Here is an example rad file:

```
#!/- device.rad -----
// This example rad file shows some
adogen // input that can be used to
build ADOs.
ABSTRACT = two d profile;
CONCRETE = flag;
ARGS = int gain, char * string2;
VERSION = $Revision: 0.0 $;
DESCRIPTION = A simple, sample
ADO;
// Member data (ADO globals)
MEMBERDATA = long int _last;
PARAMETER gain {
CATEGORY = CONT_SETTING;
TYPE = ShortType;
READWRITE = W;
DESC = allows an electronic gain to be set;
SETCODE = {
// set the gain of our device
const int writeVal = write(_fd, (char *) setGain, strlen(setGain));
if ( writeVal != strlen(setGain) ) return ADO_HW_FAILURE; // hardware error
return OK;
} // ENDCODE
}
PARAMETER array size {
CATEGORY = BASIC;
TYPE = ShortType;
READWRITE = W;
SETCODE = {
// set the size of our array
variableLengthArray temp(p,
_arraySize); _arrayParameter =
temp;
} // ENDCODE
PARAMETER arrayParameter [ ] {
// Parameter uses a variable length
array CATEGORY =
CONT_MEAS;
TYPE = ShortType;
READWRITE = R;
DESC = variable length array values;
}
THRUCODE = {
extern "C" void updateNow(char * codes) {
// Tell the device to update
const int writeVal = write(_fd, (char *) codes, strlen(codes));
if ( writeVal != strlen(codes) ) return ADO_HW_FAILURE; // hardware error
return;
}
} // ENDCODE
EVENTCODE updateStuff= {
// update some stuff
updateNow('\0xff00');
} // ENDCODE
```

```
ENDDF;  
// end device.rad -----
```

The rather simple ADO described in the above .rad file has three parameters. It also has memberdata, thrucode, and some eventcode.

The *ARGS* keyword allows the ADO designer to specify arguments to the ADO constructor; values for these arguments can be listed in configuration files which will be used when a particular ADO instance is loaded. The *VERSION* and *DESCRIPTION* can be useful in CLC utility programs. The *arrayParameter* shows how a variable length array is specified by using empty square brackets; a fixed length array's length is specified by putting a number within those brackets. The *EVENTCODE* provides a routine called "updateStuff" where the Event subsystem can access the updateNow() *THRUCODE*.

### III. ADOGEN OUTPUT

When the above file is used as input, adogen generates files for both the Abstract and Concrete classes. These files contain all the source code, header information, and Makefile details necessary to compile an ADO class which can be loaded into a Front End Computer. Simply invoking the UNIX make utility with the concrete class's make file gets everything compiled and linked. Then instances of this class can be dynamically added and removed from the system, and the parameters can be set and obtained from applications running at the console level. The interface to an ADO and its parameters is well known, so even generic applications, which are not privy to the specifics of the hardware involved, can be used to view and adjust values.

### IV. CONCLUSIONS

Code generation allows the designer of an Accelerator Device Object class to leverage a concise description of the ADO's parameters and the particulars of the hardware interface(s) involved into a full-featured ADO with all of the attendant infrastructure elements and procedures fully implemented. This is much more efficient and less error-prone than either individually coding each class from scratch or using a "boiler-plate" method where one has to fill in the blanks with parameter information that must be kept consistent through several different representations. Among the future enhancements being considered is the possibility of using the .rad files as sources for database information about the ADOs in RHIC. In the same way that Adogen is used to generate C++ code from these files, another program can be used to generate SQL code for database access and updates.

One advantage of having the .rad files be the "authority" on what is the current version of an ADO is that they are text-only files. They can be manipulated and examined using standard UNIX tools. This allows, for example, using RCS for version control.

Adogen is currently being used at RHIC to define, develop and maintain more than twenty different ADO Classes. The median size of one of these rad files is about 200 lines; these 200 lines generate more than 1700 lines of C++ code and makefile information. RHIC benefits from the savings in programmer time and effort and from the improved concordance between different ADO class implementations.

### References

- [1] L.T. Hoff and J.F. Skelly, Accelerator Devices at Persistent Software Objects, Nucl. Instr. and Meth. in Phys. Res. A 352 (1994), 185-188
- [2] D.S. Barton, Controls for RHIC, a Progress Report, Nucl. Instr. and Meth. in Phys. Res. A 352 (1994), 6-12
- [3] J.R. Levine, T. Mason and D. Brown, lex & yacc, (O'Reilly & Associates, Sebastopol, CA 1992).

# USE OF AN EXPERT SYSTEM FOR BEAM DIAGNOSTICS

V.M. Rybin, G.V. Rybina  
Moscow State Engineering Physics Institute (Technical University)  
Kashirskoye shosse, 31, Moscow, Russia

## Abstract

This paper treats the use of artificial intelligence methods, in particular expert systems, for beam diagnostics. Such an expert system is capable of solving the problems of analysis, choice and design of beam monitors for an accelerator control system according to their purposes, technical demands, working conditions and criteria for choice. This expert system provides intellectual support at all stages of design and development. This paper presents basic results and the structural scheme of the integrated expert system.

## I. Introduction

Beam diagnostics involve a complex process of obtaining, transforming and representing various pieces of information about beam parameters using theoretical, technical and software tools. Successful decisions in these matters are necessary for effective management of accelerators; not only during the exploitation process, but also during construction and commissioning; for development of control systems, for increase of reliability, for improvement of measurement accuracy etc.

## II. Background

The following pieces of information on beam parameters must be available:

- I. Definitions of beam parameters and their common interpretations.
- II. A classification of monitors according to different criteria.
- III. A definition and standardization of measurement, exploitation-reliability and mechanical-technological characteristics of beam monitors, ranges of beam parameters, technical demands, working conditions and criteria for choice.

Analysis of modern trends in the development and the construction of monitors for beam diagnostics has revealed some problems. It is possible to use the methods and ideology of artificial intelligence for arriving at decisions concerning them. For instance, strong integration in the field of development, commissioning and exploitation of control systems for accelerators displayed evident contradictions with numerous existing commercial and industrial choices. These systems, having been created by different organizations and scientific centers, are incompatible with each other. Similarly the clients who wish to use and apply concrete expert systems for beam diagnostics may encounter some difficulties which are just part of this problem. Despite a large number of heterogeneous developments and a diversity of characteristics, parameters and demands, as well as a lack of competent independent experts, one can still address significant matters, especially when the results can lead to economic waste and unpredictable consequences.

Another problem is the exponential growth of information in specialized technical magazines, books, reports etc. This abundance of information makes it difficult for developers, managers and scientific workers who specialize in this field to keep up.

### III. Analysis

Taking into account all these elements, it is felt that there are gains to be had in creating an integrative expert system (ES) for overall analysis and for choosing monitors to study beams according to their purposes, technical demands and abilities, working conditions and the choice of data handling criteria during the design process of the accelerator control system.

It is known that the development of an ES is not always well founded. However in this case the principal factors which encouraged this development are as follows:

- 1) The problems are extremely specialized. They serve definite purposes and require large numbers of criteria in the choice of solutions.
- 2) The problems require a great deal of experience in the field of diagnostics of charged particle beams.
- 3) It is possible to formalize the facts and heuristics of the knowledge base.

The development of such an ES is well founded economically because similar systems can reduce demands on experts and consultant services and save labor. Also the characteristics of the problems under consideration do not allow them to be solved with methods of traditional programming, i.e. the application of non-algorithmic heuristics is essential because it is necessary to manipulate both symbolic information and numerical data. There is a great problem complexity and a multitude of problem variables and connections.

### IV. Integrated ES

This report considers a project of an integrated ES which examines the use of expert knowledge, as well as algorithms, procedures and models resulting from previous investigations regarding the nature of these problems, and a database of monitors and containing the information on measuring schemes [1].

The basic stages of choosing the monitors and the definition of parameters realized in the current ES are:

- 1) Evaluation of limitations of monitors according to their time characteristics and sensitivity;
- 2) Choice of those monitors which satisfy given technical demands and working conditions;
- 3) Choice of monitors according to their level of influence on beam particles, taking into account required measurement, exploitation, and reliability characteristics and any given constraints;
- 4) Comparison of selected types of monitors according to the information they examine, and choice of monitor type with appropriate mechanical and technological characteristics;
- 5) Calculation and definition of the parameters of a selected monitor type with the possibility of technical realization of a measurement scheme for charged particle beam parameters.

The structure of the ES scheme is shown in Figure 1. The integrated system GURU is selected as the most convenient and powerful tool of ES design automation and is run on an IBM-compatible PC/AT computer. GURU unites the ES design scheme, a relational database, spreadsheets, text editors, business graphics and the means of communication with other computers. The knowledge base volume of the current ES version consists of about 150 rules. The ES provides intelligent support of all design stages of a control system and its exploitation process as well as for diagnostics.

At present, the integrated expert system's knowledge base holds information about different types of beam parameter monitors (BPM), such as electromagnetic and electrostatic position monitors, current transformers, Faraday cups, secondary emission monitors etc. This information includes basic descriptions of the monitors, such as sensitivity, bandwidth, resolution, precision, dimensions etc. In

order to obtain this data over 400 papers, reports and lectures describing results of investigations of different BPMs were processed.

This knowledge base is being continually appended and extended as new information (conference proceedings, publications in different journals, ...) is received. The collection of related software is growing as well.

The possibilities for the integrated expert system are continually broadening, and the system is used in designing control and measuring systems not only when choosing a BPM and calculating its parameters but also to choose the structure and the modules of various subsystems (analog-to-digital converters, amplifiers, integrators, the means of processing, registering and displaying the information about beam parameters), concerning their working parameters, technical requirements and computation criteria.

## V. Example

An example of a rule in the expert system is the following:

```
RULE : R1
  IF : 0.1 <= sm <= 10 &
        1 <= pl <= 10 &
        50 <= pf <= 1000 &
        pc >= 1 &
        nopcf = "yes" &
        inf = "no",
  THEN : type = "current transformer" &
         run "ct.exe"
```

The following variables are used :

- sm - sensitivity of monitor (V/A);
- pl - pulse length (ms);
- pf - pulse frequency (Hz);
- pc - pulse current (mA);
- nopcf - necessity of obtaining the pulse current form;
- inf - influence on beam parameters;
- type - type of beam monitors;
- "ct.exe" - calculation program for a current transformer for satisfaction of technical demands.

## VI. Conclusion

It should be noted that the demonstration prototype of an integrated expert system illustrating the basic principles was developed by means of the system GURU for PC 386/486. At present the building of a of research real-time prototype has been started. This new project is based on G2 by Gensym for Alpha AXP workstations.

## Integrated Expert System

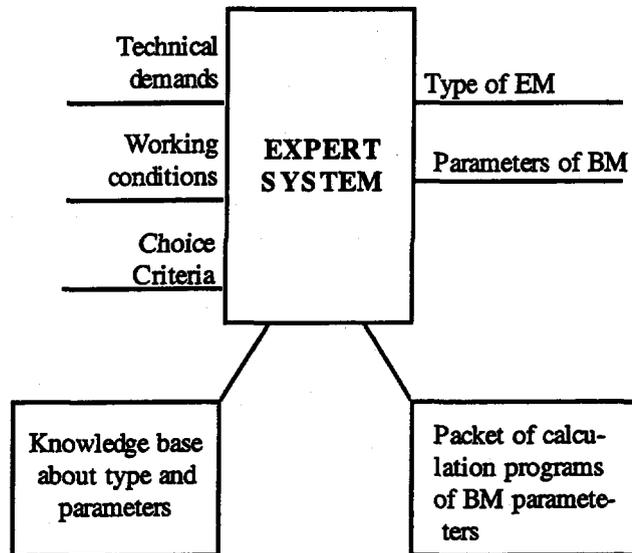


Figure 1: Structural scheme of the ES.

### VI. References

- [1] V.Rybin., G.Rybina., V.Safonenko. "An Expert System for Beam Diagnostics" 15th Intern. Conference on High Energy Accelerators, Hamburg, July 1992, pp. 237-238.

# Conditioning of the SPring-8 Linac RF System Using Fuzzy Logic

Hironao SAKAKI, Toshihiko HORI, Hiroshi YOSHIKAWA, Shinsuke SUZUKI,  
Tsutomu TANIUCHI, Atsushi KUBA, and Hideaki YOKOMIZO  
JAERI-RIKEN SPring-8 project team,  
Kamigori-cho, Ako-gun, Hyogo, 678-12 JAPAN

## Abstract

In present accelerators it is necessary to construct advanced remote control systems. This is because the facility becomes huge and high quality beams are always specified. Under such situations the accelerator must be operated by setting precisely quantitative parameters obtained from accelerator science (quantitative control). However excellent beam characterizations are obtained not only from the quantitative control but also from the hands-on tuning provided by accelerator experts or operators (manual control). Manual control is usually qualitative and is done with uncertainty. The computer is not good at such control, so that experts or operators must aid the computer with their knowledge. If the computer system were able to embed manual control, the required beam could be obtained with much easier adjustment of the system. Such would be an Artificial Intelligence (AI) control system. At the SPring-8 Linac we have undertaken the construction of an AI system, using fuzzy logic theory. In this theory manual control is expressed by a set of linguistic rules of the form IF-THEN, plus numerical membership functions. For our first attempt we apply fuzzy logic for conditioning of the RF system. This fuzzy logic system is based on VME computers.

## I. Introduction

The commissioning of the SPring-8 1 GeV injector Linac will start in August 1996. Before commissioning starts, all conditioning of Linac components will have to finish efficiently. We have designed and constructed the Linac control system [1] [2] and we are now trying to begin remote control of conditioning. Special mention should be made of the conditioning of the high voltage Radio Frequency (RF) system. Because we have chosen 80 MW klystrons, (Toshiba E3712, 2856 MHz) the RF system will have the largest average RF power in the world. The number of E3712s is 13, and we usually drive one at about ~60 MW, ~4 msec and ~60 pps. We must achieve near this power in a conditioning period from May to July 1996, a very short time. Thus, we plan that the RF conditioning will be under the AI control system, and hope to achieve a reduction in conditioning hours.

The chief problem in RF conditioning involves electric discharges, a difficult phenomenon, and best handled with the experience of experts. RF conditioning system is best directed by experience, just the situation for an AI control system. If an AI control system is adopted with fuzzy logic, then it might be able to simulate the behavior of the manual control. Thus, we shall try to control the conditioning of the RF system using the experience and sense embedded in fuzzy logic, in the hope of conditioning the RF system with high efficiency.

## II. High power conditioning test of the wave guide circuit

Before design of the conditioning of the RF system using fuzzy logic simulation, the high power conditioning test of the wave guide circuit (the RF conditioning test) was carried out [3]. The test unit of a wave guide system is composed of RF windows, 3dB directional couplers, vacuum pumps and phase shifters.

The test unit is shown in Figure 1, and the RF conditioning test is summarized in Figure 2. After 420 hours a stable operation with maximum RF power (80 MW - 4 msec - 60 pps) was realized. However, this test unit does not connect to accelerator tubes. When it is, we will have longer conditioning times (500- 550 hours per one unit). We shall try to realize conditioning times of under 500 hours using fuzzy logic control. This test was very helpful in making fuzzy production rules for our AI control system.

## III. Fuzzy logic system

### A. Outline of fuzzy logic and the fuzzy logic engine board

The characteristics of fuzzy logic are shown in Table I, in comparison with PID (Proportional, Integral, Derivative) control. What is seen is that fuzzy logic control is good at linear and non-linear multi-parameter systems using experiential rules.

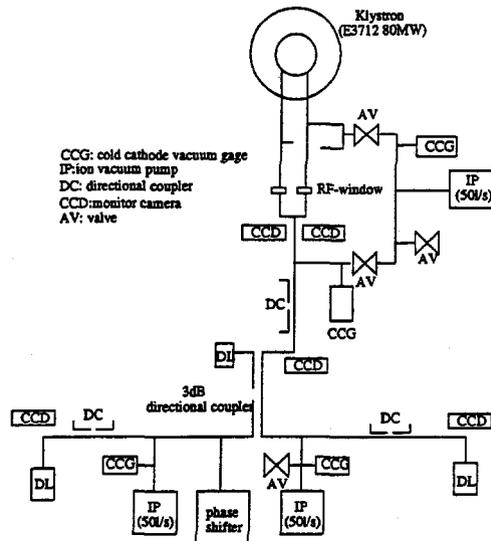


Figure 1. High Power test bench

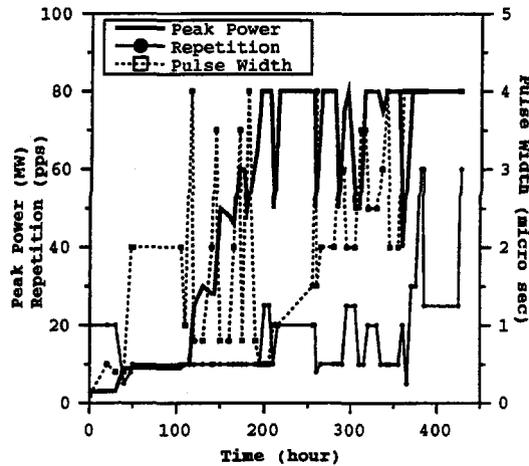


Figure 2. Summary of the RF conditioning test

As conditioning of the RF system is quite experiential, involving avoidance of electric discharge, it is difficult to make a PID model for its control. However, we think that it is suitable for fuzzy logic control. There are some fuzzy engine boards on the VMEbus market, able to calculate fuzzy logic at high speed. We are going to adopt one, EVME-FZY21(ELNIS JAPAN), for our VME computer control system, and have already written its device drivers in C. Under fuzzy logic, a system is controlled by qualitative rules, based on linguistic and unclear values, such as "TIME = little," where the rule parameters have no particular thresholds. The board has an easy rule editor, so that we have been able to model and set up the fuzzy production rules.

As to the fuzzy logic calculations, this board has two types of defuzzifier, a Center of Gravity (CG) method, and a Height method. Of these we have chosen the CG method for our engine. During conditioning, the system will be as shown in Figure 3.

### B. Modeling of the RF conditioning

We must model the conditioning of our RF system. All of an operator's judgments are influenced by experience in such work. First of all we state a new experiential guideline which we call "RF power safety (safety)".

Table I  
Comparison of fuzzy and PID control

characteristic	fuzzy	PID
the subject of control		
: model	unnecessity	necessity
: qualitative relation	experiential	unnecessity
operation		
: speed type	possible	possible
: positional type	possible	possible
: saturation	little	many
read a rule	good	bad
structure of control system	complex	simple
changeover control	sequence	nonsequence
knowledge control	fit	unfit
number of input, output	one or many	one set only

1. If it is zero, then the peak power agrees with the RF conditioning test value (experiential value).
2. If it is a negative large value (safety=bad), then the peak power does not agree with the experiential value, and it is above the "safety = zero" line. There is a significant possibility of electric discharge.
3. If it is a positive large value (safety=good), then the peak power does not agree with the experiential value and is under the "safety = zero" line. There is little possibility of electric discharge.

Safety is used for judgment of the RF peak power value. Figure 4 shows the judgment of RF power safety. Under this guideline, we have modeled that there is a two step judgment in the RF conditioning process.

*Step 1:* Safety is obtained from the results of RF conditioning tests. It requires two input variables.

- a. Time interval is small or large.
- b. Peak power is low or high.

*Step 2:* We judge the peak power steps according to the present safety. Three input variables are required.

- a. Safety is good or not
- b. RF pulse width is short or long
- c. Vacuum value is good or not

These variables are used to make the fuzzy production rules and membership functions. Figure 5 shows the block diagram of the RF conditioning system. The following subsection discusses in detail the construction of fuzzy production rules.

### C. Fuzzy production rules and membership functions

We have made 52 rules and 6 membership functions from models of conditioning.. Generally speaking, this control systems should be designed to be fail-safe. Our high power RF system, having such a large average power, has the potential to cause severe equipment damage in the case of an accident. Figures 6 and 7 show the fuzzy production rules and membership functions created.

Each membership function has the following definition.

#### A) Fuzzy inference Step 1

1. Definition of peak power and time on the RF conditioning test.

There are 5 levels for raising the peak power, 0–10 MW (0–100 hours), 30 MW (~110 hours), 50 MW (~120 hours), 60 MW (~150 hours), and 80 MW (~180 hours). These regions have greater possibility of electric discharge. Thus we must be careful in these regions not to hurry up input power increases. This is reflected in the definitions of the membership functions in these regions.

2. The definition of safety peak power has 5 regions, so safety also has 5 regions.

As the system must be kept fail-safe, the membership function of safety leans toward the negative.

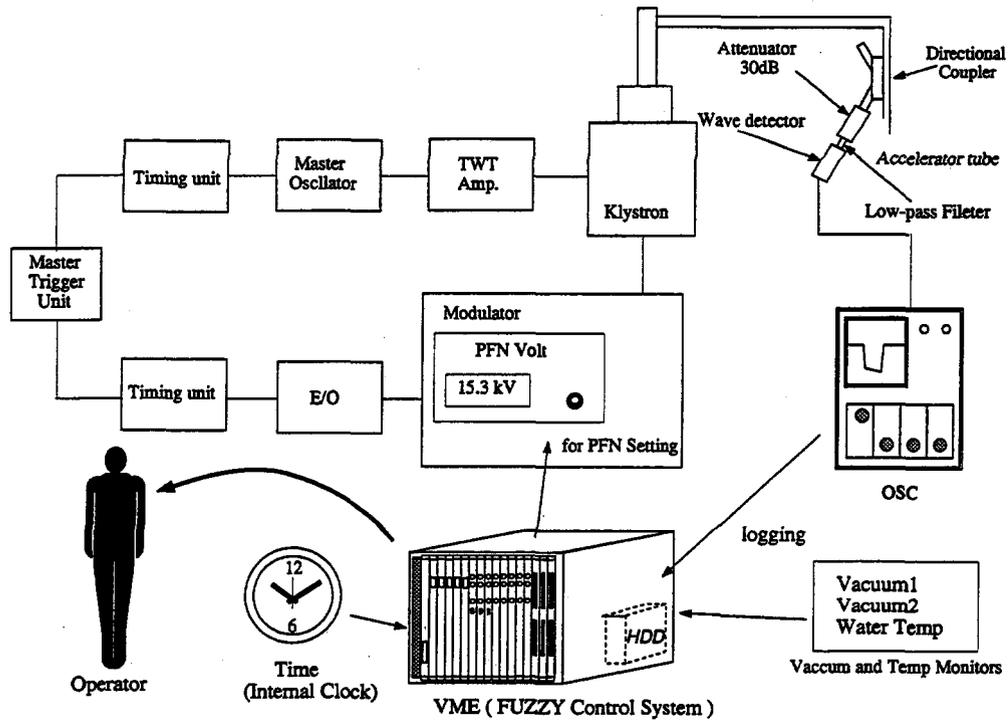


Figure. 3. Design of the conditioning system using fuzzy logic

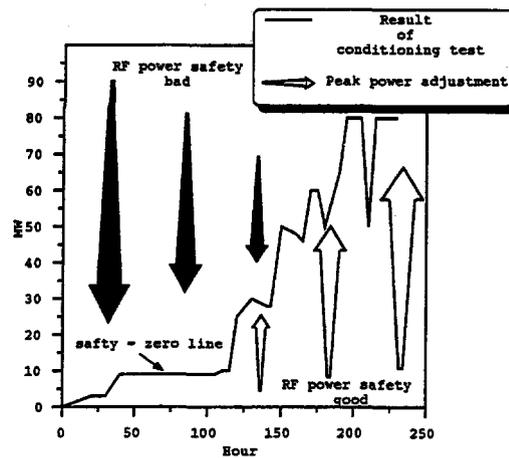


Figure. 4. Judgment of the Conditioning

### B) Fuzzy inference Step 2

#### 1. Definition of Vacuum

During RF conditioning we want to keep the vacuum value about  $1-3 \times 10^{-7}$  torr. The vacuum interlock set level should be  $\sim 5 \times 10^{-7}$  torr.

#### 2. Definition of Pulse Width

As pulse width increases electric discharges occur more frequently. We must be especially careful for pulses longer than 2 msec.

#### 3. Definition of Step

Step is the control parameter of RF power adjustment. If step is positive and large, then the RF peak power is getting large. For example, if the vacuum value is getting bad then the step must get negative quickly, as we must keep the system fail-safe.

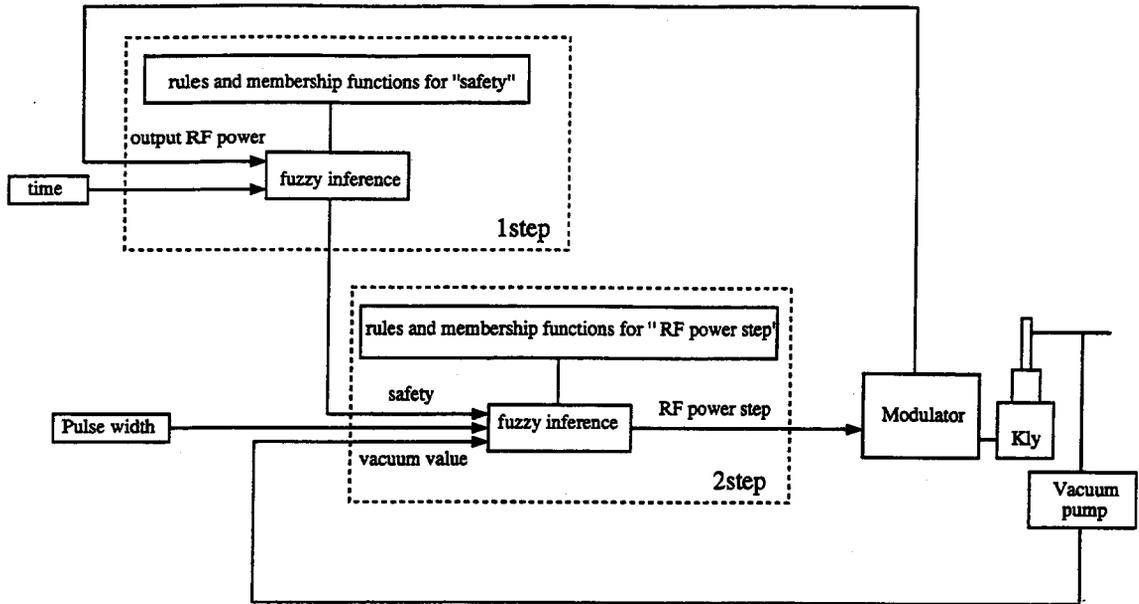


Figure. 5. Block diagram of the RF conditioning system

Table II  
Result of the step2 fuzzy inference

Peak power (MW)	10	40	40	40	40	40	60	80
Time (hour)	180	180	30	180	180	180	50	200
vacuum value ( $\times 10^{-10}$ torr)	1000	1000	1000	1000	100	4000	1000	1000
pulse width ( $\mu$ sec)	0.5	0.5	0.5	3	1	0.5	0.5	1
safety value	11	3	-35	3	3	3	-35	0
power step	7	5	-6	4	7	-26	-6	1

#### IV. Simulation

The RF conditioning was simulated using these fuzzy production rules and membership functions. Figure 8 shows the result of step 1 fuzzy inference. A simulation was performed relating passage of time and peak power. In this figure the open squares show "safety = zero" from the simulation. Using our fuzzy production rules and membership functions, the simulation is in good agreement with the RF conditioning curve. Table II shows the results of step 2 fuzzy inference. For example, if peak power is 10 MW, time is 180 hour, vacuum is  $10^{-7}$  torr, and pulse width is 0.5 msec while the passage time is very long, then peak power is not large, so that the power steps can be given a positive large value (=7). If vacuum is too bad,  $\sim 5 \times 10^{-7}$  torr, the power steps are large and negative (value = -26).

As we want to have a minimal discharge system, the peak power is lowered quickly. When peak power of 80 MW is achieved, the power step is near zero. In this case the system does not require more RF power.

#### V. Conclusion

We have constructed and simulated an RF conditioning system using fuzzy logic. In May 1996 we will use it in production, and hope to achieve conditioning times under 500 hours. First we will do the RF conditioning test (local operation) of only one RF unit (with accelerator tubes) for recheck of the RF system. Though there may be some accidents in this checkout, the fuzzy logic control system will reflect this. In the future, the membership functions will be learned by neural control, and thus the system will become more intelligent. Later we are going to gather many fuzzy production rules on Linac operation, for example, phase control, beam energy control, etc. These rules will be managed using a

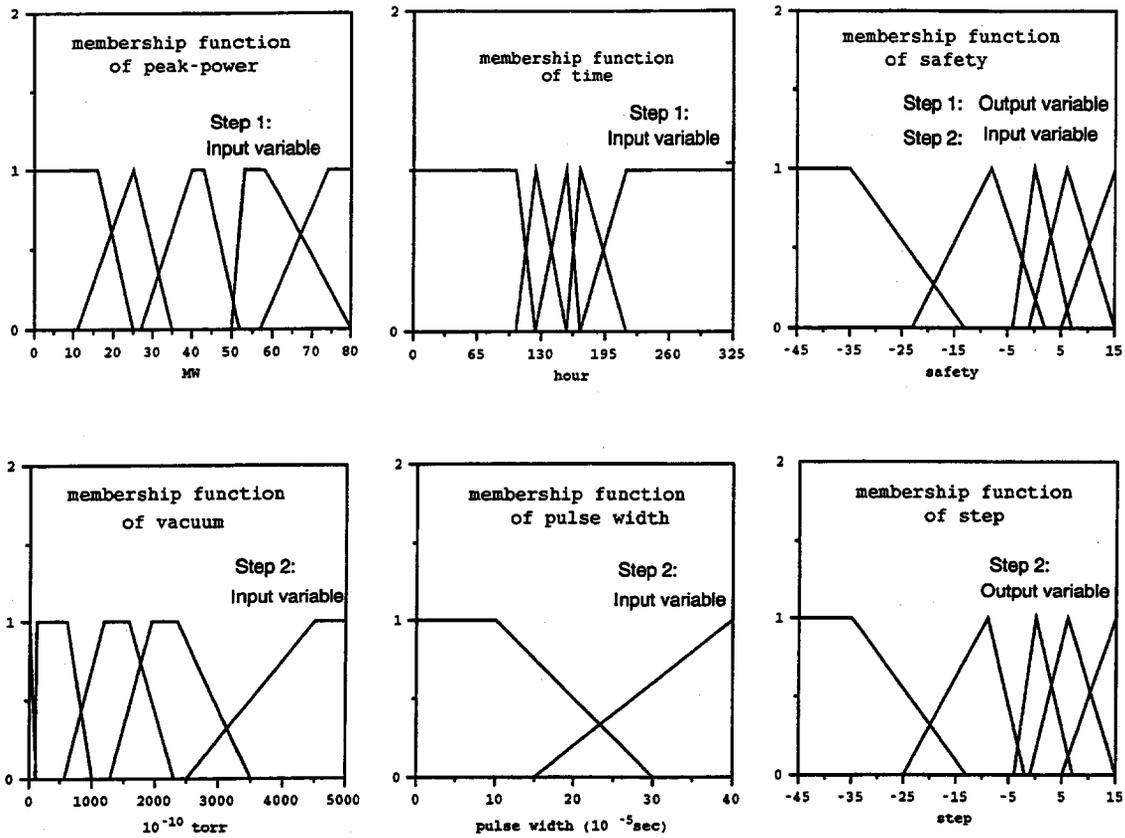


Figure 6. Membership functions of the RF conditioning

database and employed on the AI control system.

### References

- [1] H.Yoshikawa et al., Nucl. Instr. and Meth. A, 352,216-217 (1994)
- [2] H.Sakaki et al., Proc. of Particle Accelerator Conference and International Conference of High-energy Accelerators, Dallas, Texas USA (1995)
- [3] S.Suzuki et al., Proc. of 10th Accel. sci. and Tech., Hitachinaka, Japan, Oct. 25-27, 1995, to be published.

### Fuzzy inference Step 1: 25 rules

If PEAK = quite low	& TIME = little	then	STABILITY = good
If PEAK = quite low	& TIME = zero	then	STABILITY = positive small
If PEAK = quite low	& TIME = big	then	STABILITY = positive big
If PEAK = quite low	& TIME = quite big	then	STABILITY = positive big
If PEAK = quite low	& TIME = too big	then	STABILITY = positive big
If PEAK = low	& TIME = little	then	STABILITY = negative small
If PEAK = low	& TIME = zero	then	STABILITY = good
⋮			
If PEAK = quite high	& TIME = quite big	then	STABILITY = good
If PEAK = quite high	& TIME = too big	then	STABILITY = positive small

### Fuzzy inference Step 2 : 27 rules

If STABILITY = negative big	& VACUUM = too bad	then	STEP = negative big
If STABILITY = negative big	& VACUUM = quite bad	then	STEP = negative small
If STABILITY = negative big	& VACUUM = good	then	STEP = negative small
If STABILITY = negative big	& VACUUM = quite good	then	STEP = good
If STABILITY = negative big	& VACUUM = excellence	then	STEP = positive small
If STABILITY = negative small	& VACUUM = too bad	then	STEP = negative big
⋮			
If STABILITY = positive big	& VACUUM = quite good	then	STEP = positive big
If STABILITY = positive big	& VACUUM = excellence	then	STEP = positive big
⋮			
If PULSE_WIDTH = short	then	STEP = positive small	
If PULSE_WIDTH = wide	then	STEP = zero	

Figure. 7. Rules of the RF conditioning

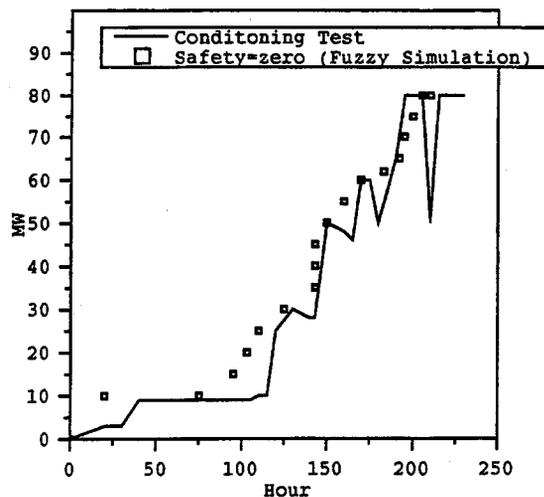


Figure. 8. Result of the step1 fuzzy inference

# XUIMS

## The X-Window User Interface Management System at CERN

M. Vanden Eynden (E-mail : veym@dxcern.cern.ch)  
CERN SL Controls Group  
European Laboratory for Particle Physics  
CH 1211 Geneva 23, Switzerland

### ABSTRACT

The CERN X-Window User Interface Management System (XUIMS) is a modular and highly configurable software development environment allowing the interactive design, prototyping, and production of OSF/Motif Human Computer Interfaces (HCI). Fully compliant with the X11R5 and OSF/Motif industry standards, XUIMS covers complex software areas like the development of schematics, the visualization and on-line interactions with 2D and 3D scientific data, the display of relational database data and the direct access to CERN SPS and LEP accelerator equipment.

The guarantee of consistency across the applications and the encapsulation of complex functionality in re-usable and user-friendly components has also been implemented through the development of home made graphical objects (widgets) and templates.

The XUIMS environment is built with commercial software products integrated in the CERN SPS and LEP controls infrastructure with a very limited in-house effort.

Productivity and quality have been improved through less coding and better HCI prototyping.

### INTRODUCTION

The X-Window and OSF/Motif industry standards have been used since 1992 to produce the Human Computer Interfaces (HCI) required for the control of the two CERN largest particle accelerators : SPS and LEP. The decision to base the HCI software activity on industry standards was made to avoid, as much as possible, vendor-specific developments and to guarantee thereby the software investment for a long time [1].

The desire to optimize the production of HCIs led to several market investigations and product evaluations. In 1994, a commercial User Interface Management System was selected to become the kernel of the CERN X-Windows User Interface Management System (XUIMS) [2].

### X-WINDOWS AND OSF/MOTIF

The X-Windows and OSF/Motif software market has rapidly evolved during last two years and today offers a large choice of HCI software development environments. One could still consider the development of HCIs using only the standard X-Window and OSF/Motif libraires, but if a higher degree of abstraction and reusability is required, or if special features (i.e. 2D and 3D scientific data visualization) are needed, then using a powerful software development environment becomes mandatory. The next four sections summarise the different strategies that can be considered for the development of OSF/Motif interfaces.

#### *X-WINDOWS AND OSF/MOTIF LIBRARIES*

The X-Windows and OSF/Motif libraries (Xlib, XtIntrinsics, and Xm) are intended to be called from a C application and contain low level functions for the creation and manipulation of HCI components (i.e. buttons, labels and lists of elements). Any modification in the HCI requires a change in the C program (i.e. editing, compilation, and debugging development cycle). The final HCI C code is not understandable by any Interface Development Tool (IDT) or User Interface Management System (UIMS) and has therefore to be maintained manually.

## OSF/MOTIF USER INTERFACE LANGUAGE

The OSF/Motif UIL language is a specification language for describing the initial state of a user interface for a Motif application. The specification describes the objects (i.e. menus, form boxes and buttons) used in the interface and specifies the functions to be called when the interface changes state as a result of user interaction. The main advantage of UIL is that it is understood by most IDTs and UIMSs. Nevertheless, UIL does not address the dynamic aspects of the interface (i.e. printing an information message when the user presses a button) and still requires the recompilation of the UIL file after each modification.

## INTERFACE DEVELOPMENT TOOLS

Standard Interface Development Tools (IDT) help the developer to design the layout of the interface through a *What You See Is What You Get* (WYSIWYG) interface and to generate the corresponding C code. The application code has then to be inserted into stubs. The main disadvantage of the IDT is that the HCI behavior (i.e. putting an alarm message on the screen) must still be handled by the application code, leading to non-reusable software (see chapter about HCI development principles).

## USER INTERFACE MANAGEMENT SYSTEMS

User Interface Management Systems (UIMS) add to the IDT functionality the possibility to design and prototype the dynamic behavior of the HCI. These tools can usually import UIL files and export UIL, C, or C++ HCI code. Their ability to address both the static and dynamic aspects of the HCI offers for the first time the possibility to put in place a real HCI software development lifecycle policy.

These considerations led us to buy a commercial UIMS and to enrich it with :

- new HCI components such as 2D and 3D graphics widgets,
- additional high level C functions (i.e. routines for direct SPS and LEP equipment access),
- templates for a common style and behaviour within the SPS and LEP operational environment.

## THE XUIMS ARCHITECTURE

As shown in Fig 1, XUIMS is a highly modular and configurable environment composed of the following building blocks :

- *The XFaceMaker™ UIMS*. XFaceMaker[3] has a WYSIWYG editor for the interactive creation of HCI layouts.

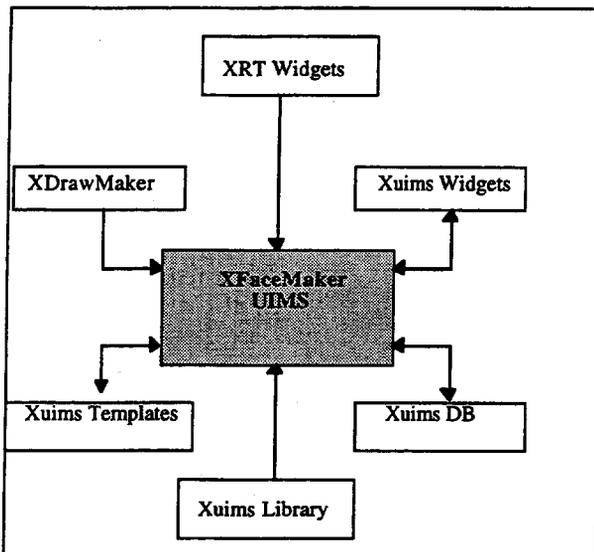


Fig 1 : The XUIMS Architecture

The HCI behavior is specified in the form of procedural scripts in a high-level, C-like programming language called FACE. The behavior can be tested within the tool using the built-in interpreter and debugger for the FACE scripting language. C, C++, and UIL code are generated by the tool. Additional C libraries can be linked into the UIMS kernel (i.e. libraries for SPS and LEP accelerator equipment access) in order to extend the basic functionalities of XFaceMaker. Working HCIs produced with XFaceMaker can run on any workstation without run-time fees (this is also the case for the other commercial products used in XUIMS). XFaceMaker also allows the programmer to define templates (composite objects) and to add new widgets to the standard OSF/Motif widgets set.

- *The XDrawMaker™ drawing package.* XDrawMaker [4] is a drawing editor enhanced by the ability to animate objects. When used with XFaceMaker, every object becomes a widget that can be included and manipulated in the HCI. Attributes of objects in complex schematics (see Fig 2) can be handled as any standard OSF/Motif widget. Almost any imaginable task can be handled with XDrawMaker.

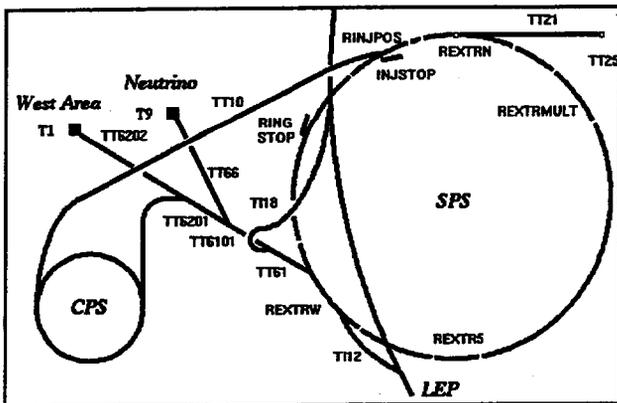


Fig 2 : Schematics of CERN's intricate particle beam network built with XDrawMaker.  
Courtesy of the CERN SL Transfer Line project team.

- *The XRT™ 2D, 3D and Table widgets.* The XRT/2D widget can display virtually any type of two-dimensional graph, including barcharts, X-Y plots, pie charts, area graphs and logarithmic scientific charts. The XRT/3D widget can contour and zone 3D data. Users can dynamically rotate, zoom and scale views. The XRT/table widget allows developers to display and manipulate tabular data. The XRT widgets were selected for their excellent real-time performance. Fig 3 shows Xplore, the CERN off-line scientific data visualization package built with XUIMS and the XRT/2D widget.
- *The XUIMS library.* This C library contains additional functions used for direct access to the SPS and LEP accelerators equipment, for direct access to the UNIX file system, for providing on-line hypertext help on the WWW and for the handling of HCI components (i.e. flashing widgets, information messages, and cursor manipulations).
- *The XUIMS widgets.* Extending the OSF/Motif widgets set with XFaceMaker is a trivial exercise. New widgets and their associated code can be developed interactively with the WYSIWYG editor and added to the standard palette afterwards. This important feature has been used to create additional widgets for formatted numeric entries, for file printing and screen dumping and for providing help.
- *The XUIMS templates.* As shown in Fig 3, a template is a model, defined by a pre-designed object, that can be applied to other objects, called the instances of the template, which inherit the attributes defined in the template. A template can also be a composite object (i.e. a main window with predefined buttons and actions). Templates are very useful to encapsulate high-level functionality in a reusable way and to enforce consistency across the applications.

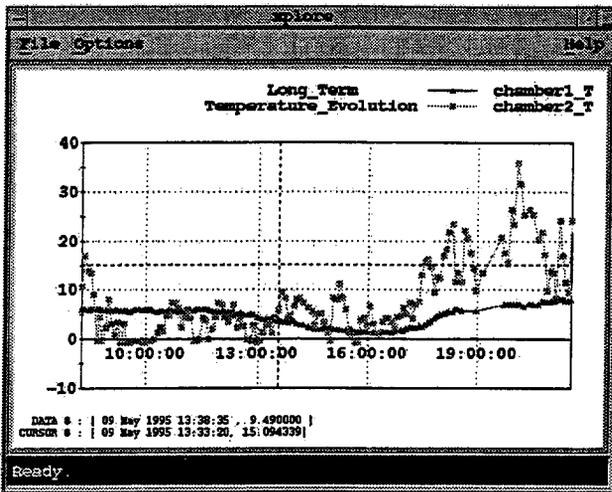


Fig 3 : The CERN Xplore off-line scientific data visualization Package built with the XUIMS 2D template.

- *The XUIMS resources database.* The main purpose of this database is to guarantee a standard behavior for all X-Window applications running in the SPS and LEP control room. Consistency is important both between applications and within a single application. Consistency helps the user to transfer familiar skills to new situations. The user can apply the knowledge learned from one application to another application, reducing the amount of learning and subsequent calls (i.e. an operator will easily recognize a push-button in an application if all pushbuttons have the same background and foreground colors). A primary XUIMS resources database [5] contains low priority definitions of resource values (i.e. fonts and colors) affecting all X-Window HCIs. If needed, a secondary resource database containing higher priority resource values can be provided by the developer for one given HCI.

## THE HCI DESIGN WITH XUIMS

### THE INTERFACE AND THE APPLICATION

The fundamental principle of HCI software development is the separation between the application and the interface [6]. The application can be seen as the set of routines and algorithms specific to the process or equipment to be controlled (i.e. routine for switching a vacuum pump ON and OFF). The interface can be seen as a set of windows through which the user will interact with the application.

In terms of reusability, one application code may be invoked by several interfaces based on different technologies (i.e. a C library used to control an equipment could be invoked either from an OSF/Motif operator interface, or from a C program running on a VT100 terminal).

A lack of separation between the application and interface software layers leads, in most cases to non-reusable software (i.e. equipment access code containing lowlevel OSF/Motif library calls). A functional specification of the data flows between the interface and the application layers is a mandatory step towards better software reusability and a better development lifecycle. The HCI development lifecycle described in the next section is a first step in this direction.

### THE HCI DEVELOPMENT LIFECYCLE

As shown in Figure 4, a typical HCI development lifecycle with XUIMS [6][7] can be decomposed as follows :

- I. *The application concept.* This phase, also called the user requirements phase, is aimed at gathering quantifiable HCI requirements, identifying the future users of the system and drawing the HCI metaphor.
- II. *The functional specification.* This phase consists of the translation of an application concept (developed by a user) into a functional software specification. It should be independent of any particular windowing or graphical system. The

functionality of the application and interface layers should be carefully analysed as well as the data flowing through these two layers. Experience has shown that this phase is very often omitted by developers. This situation leads in most cases to the development of monolithic, windowing-dependent, non-reusable code.

III. *Prototyping the interface.* This exercise is aimed at building a first prototype of the HCI layout with a UIMS. The benefits of this phase are the improvement in the definition of the user requirements and the detection of inconsistent or contradictory requirements.

At this stage, no connection between the interface and the application is implemented.

IV. *Designing the application.* This phase consists in the implementation of the application routines (i.e. database access, algorithms, interlocks, ...) as defined in phase I.

V. *Generating the working application.* This phase consists in linking together the interface code generated by the UIMS and the application code in order to produce the working application.

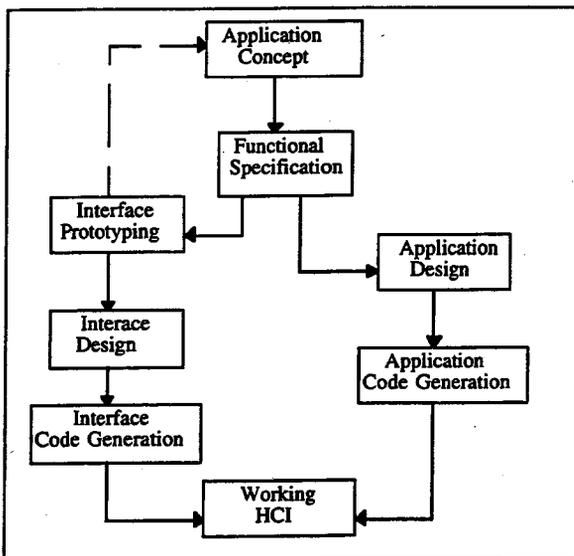


Fig 4 : The HCI development lifecycle with XUIMS

Very often the HCI lifecycle has to be iterative and prototyping should never be ignored. This XUIMS HCI development lifecycle allows the developer to draw the visual metaphor through which the user views and manipulates the system before starting any other coding effort.

## CONCLUSIONS

XUIMS is operational since September 1994 and is used by an ever-growing community of software engineers. Major rejuvenation projects such as the SPS transfer line project and several graphical interfaces for the control of SPS beam instrumentation and beam transfer have been developed with XUIMS. Off-line software projects like the interface to the SPS and LEP Oracle logging system have also been developed using the XUIMS graphics capabilities. The robustness of the end-user applications as well as the openness of the environment encourages us to reduce as much as possible home-made developments and to collaborate with industry. CERN engineers now spend more time on the analysis and less time on coding. Unfortunately, providing good HCIs remains a difficult task involving detailed technical knowledge (i.e. layering of the X-Window system) and concepts of ergonomics. Building a functional specification from user requirements is a crucial phase very often omitted by software developers. By using XUIMS, turning a functional specification into a working application is now a quicker process.

## REFERENCES

- [1] The CERN PS and SL Controls Consolidation Project  
CERN PS and SL Controls Groups  
CERN Note PS/91-09 or SL/91-12
  
  - [2] The CERN SL and ST User Interface Management System - XUIMS  
Software User Manual.  
P.Ninin, M.Vanden Eynden  
CERN SL Note 94-53 (CO)
  
  - [3] XFaceMaker User's Guide V3.0.  
Non Standard Logics
  
  - [4] XDrawMaker - The Animated Drawing Design Tool for the X Window System.  
Non Standard Logics
  
  - [5] The SL X-Window Migration Project  
Resources Management Policy.  
M.Tarrant, M.Vanden Eynden  
CERN SL Note 93-77 (CO)
  
  - [6] Software Engineering The PSS-05 Way.  
Quality Systems & Software
  
  - [7] Software Engineering Standards.  
C.Mazza, J.Fairclough, B.Melton, D. De Pablo, A.Scheffer, R.Stevens - Prentice Hall
- ™ XFaceMaker is a trademark of Non Standard Logics, France.
- ™ XDrawMaker is a trademark of Non Standard Logics, France.
- ™ XRT/2D, XRT/3D, and XRT/Table are trademarks of KL Group, Inc.

# An Experiment with Electronic Logs

Alex M. Waller  
Fermilab  
Accelerator Division Controls Department  
MS 347  
P.O. Box 500  
Batavia, IL. 60510 USA

## THE BRIDGE TO FUTURE GUIs

The Accelerator Division control system was proprietary since its inception in the late 1960s and early 1970s. In the area of graphics, in-house procedures were created and maintained to communicate through CAMAC to commercial graphics hardware and to proprietary console TV hardware. In the mid to late 1980s the underlying hardware and associated code for the console TV and auxiliary graphics was replaced with calls to X11 procedures as we moved our controls environment to DECWindows on VAX workstations[1]. It should be noted that the procedure calls were basically the same and the code was done in-house to now interface to X11. Our in-house graphical user interface support continued to grow with more value added procedures[2].

Moving to the X11 paradigm provided the bridge to the use of applications running on other platforms as more and more platforms and applications adopted the X11 protocol for graphics. One such utility was MetaCard[3], a Mac-like HyperCard[4] application that ran on many UNIX platforms. A more detailed account of this utility is given in the sections to follow.

An opportunity was waiting to try an experiment not only with integrating an alternate graphical interface with our Accelerator Division controls environment, but also with computerizing a long standing traditional way of recording log information. A relatively inexpensive license for MetaCard already existed. MetaCard behaved much as another popular utility that was available on the Mac, and thus several people were familiar with the HyperCard-like functionality and scripting. A request was pending from one of the Accelerator Division departments to venture into the field of computerized logs. The electronic log application was an excellent opportunity to gain experience and to introduce operations personnel to an alternate graphical user interface. It was also an excellent opportunity to explore the area of rapid prototyping.

## THE ENVIRONMENT FOR CO-EXISTENCE

The VMS DECWindows Motif style of GUI provided a consistent windowing paradigm[5]. The underlying X11 protocol, already adopted for controls system graphics, co-existed very nicely with the entire console environment[6]. The entire MetaCard application was provided as a pull-down menu item on the console Session Manager menu bar. DECWindows allowed one to customize this menu bar and to interface menu items through the VAX DCL command language[7]. In this particular case the command is an RShell one. The RShell command provided the vital link to a host computer[8]. The host computer has a list of trusted users and nodes to allow remote login. Since the console environment on each node runs under its own account, a large list of user names was avoided. The menu item command also allowed for passing the display address for graphical output from X11. The user never sees the details of this command call, only the item on the menu list.

MetaCard itself runs within a window. A "home stack" (application) contains the icons of applications within a window. When MetaCard is started, it appears as just another window on the Accelerator Division controls consoles. Thus the operation of evoking MetaCard and working with MetaCard does not intrude on the current console environment. The MetaCard utility graphical user interface, however, is quite different from that of traditional console application programs[9].

## WHAT IS METACARD?

*A Motif-like graphical user interface using the X11 protocol*

Three dimensional widgets such as push buttons, sliders and scroll bars are available and customizable in color and font just as in Motif[10]. Pull down and scrollable menus are available and customizable. Text areas called fields are also possible. More complex widgets, such as Dialog boxes, that are composites of text areas and push buttons are possible and customizable. Resizable overlapping windows with ornamentation are realized as individual "stacks" or applications in MetaCard.

*Multimedia capable*

Hypertext capability is possible when used in conjunction with the scripting language called MetaTalk. The script language is very similar to that used by HyperCard on the Macintosh computer and is used to control the flow of MetaCard operations much as in a conventional programming language such as C. One can also execute shell commands and fork other processes such as graphics browsers. One can include simple sounds within his/her stacks just as in HyperCard. Several formats of audio are also supported. One can use the AIFF, the Sun/Next (mulaw), and HyperCard 8-bit linear formats. Image support includes the popular UNIX formats pbm, xwd, xbm for pictures as well as encapsulated Postscript. For movie animation the Macintosh QuickTime format as well as AVI and FLC/FLI ones are supported.

*Builds applications by rapid prototyping*

Existing widgets (buttons, sliders, scroll bars) can be cloned by copy and paste from one application to another. This includes all attributes including colors, fonts and even any associated scripts. This can greatly aid in building up a new application from a previous one within a very short time. Tools are included to create new widgets with complete control of colors, fonts, labels, sizes and shapes. More complex widgets (such as dialog boxes) can be designed by grouping together more primitive ones.

A script can be attached to a widget so that the widget can react to various events such as mouse or keyboard input. As mentioned above a script is written in the MetaCard programming language called MetaTalk. There are control structures such as if-then-else and repeat-until/while/with. Functions can be written in C and are callable from MetaTalk so that the scripting language can be extended. The scripts are executed by events or direct calls from within other scripts. Mouse up/down, drag and in are the primary events that activate scripts. Other events such as a "stack" (application) open or close can also elicit an action.

Event simulation can be accomplished by sending an event to an object from another script. Widgets can also react to messages (events) which are sent up the object hierarchy. For example if a script to handle an event does not exist in one widget, the event message continues traveling up the widget hierarchy in the "stack" until a widget with an appropriate event handler will consume (act on the event with a script) the message. A script within a widget can even continue to pass the message on up the hierarchy so that other widgets may process the event message.

Finally a drawing editor is provided so that users can create their own graphics and icons.

## THE ELECTRONIC LOG BOOK

*User criteria*

To be successful an electronic log book had to emulate a hand-written log closely. Not only that but there were some criteria imposed on electronic media that did not exist with a hand-written log. One of these was that once was made a log entry could never be erased or modified. This led naturally to the criterion that each log entry be able to have an addendum that, again, once entered could not be modified. Each log entry could be time-stamped automatically rather than have the author enter the date and time as in a hand-written log. To make the electronic version of a log more useful it could be given a table of contents so that an entry could be found much more quickly than in a conventional log book. In conjunction with the table of contents, word and phrase searches could be possible so that pieces of information, which might not appear in the table of contents, could be located quickly. Finally, the electronic log process must be able to capture and insert figures within the log book.

### *Design criteria*

An electronic tablet form was chosen (see Figure 1). There is a large writable area. No scrollable text is permitted which might hide text. The entire log entry is visible. All header information such as author, date and title of entry appear at the top of the tablet. All controls appear at the bottom of the tablet. The log was designed for a large screen display so that a large enough font could be used for legible text. One can actually take a screen snapshot of the log tablet, and this entire entry will fit within the bounds of an 8.5 by 11 inch sheet of paper when printed thus appearing to be almost the size of an actual hand-written log.

Since the size of a captured image could vary, and even be quite large, a link to the image is made with a button that contains a figure number (incremented automatically by the electronic log application) and a caption that is entered by the user. Any number of links can be inserted into the text field of a log entry (see Figure 2).

Also on-line help exists for the electronic log book. This help was done in a hypertext fashion so that the user can go to other useful information without having to search through the help text for more.

### *Operational experience*

As the electronic log book was increasingly used operational experience was gained. Some important modifications were made to the electronic log. One of these that was interesting was due to the initial design specification. The requirement to have log entries be unmodifiable, although mandatory for acceptance of electronic media for logging, proved to be humanly restrictive. In actual practice log entries do get modified. Usually space is kept for additional information or additional information is side-barred. A modification that seems to have worked was to lock all entries only after the electronic log book was closed. As long as the log book remained open (the application running) any entries made during the current opening of the log could be modified. To ensure that the log book would eventually be closed, a count-down timer was put into the application to close out the log book after several hours of inactivity. Since MetaCard is event driven it was very easy to use keystrokes and mouse events as activity measures.

It is interesting to chronicle the effects in making a modification. After the seemingly innocent minor change to allow log entries to remain modifiable, as in a physical lever a larger operational problem was created. Over long editing sessions, especially since the log book could now be left open over a long time so that log entries would not be write locked, it became necessary to manually save the log entries on demand. Several occasions with the momentary loss of either the server node or host node made this modification necessary.

The table of contents window was pressed into service as a more sophisticated electronic log book navigator. The sequential navigation to previous or next pages within the log book was restrictive. Double-clicking on an entry in the table of contents allows one to go directly to the selected entry (see Figure 3).

The ability to display a reduced image of any given log entry and its addendum was added. This aided in allowing one to reference one log entry while making another (see Figure 4).

Since several electronic log windows could be opened when one wished to take a snapshot it was sometimes difficult to isolate the window that was a target for image capture. MetaCard allows windows to be hidden and to be made visible. This feature proved useful in helping with snapshot taking. All open electronic log windows, including the log book, were temporarily hidden while the snapshot was in progress. After the snapshot was taken, all log book windows that were made invisible were again made visible.

Another interesting issue related to taking snapshots had to do with the design of the Accelerator Division controls console hardware. Each console could consist of up to four physical display screens. Fortunately there was a network naming scheme for these. The existence of such displays could be discovered, and a special dialog box was created requesting the user to select one of the possible four quadrants in the dialog box (corresponding to the four physical displays) before the snapshot could be taken. The information returned from the dialog box was used to insert the proper display address into the snapshot procedure.

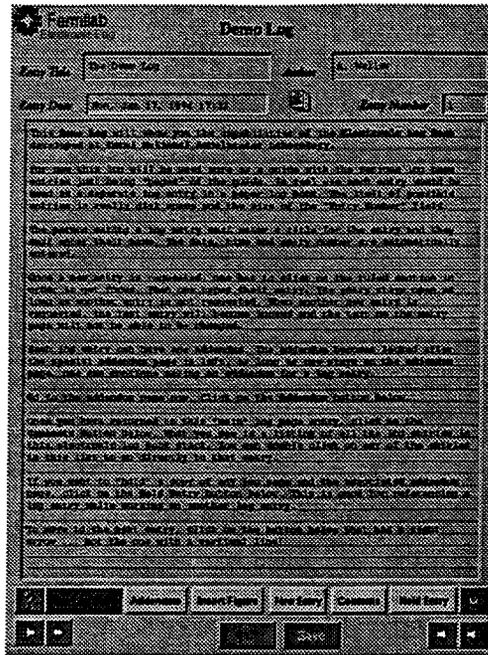


Figure 1  
The tablet format of the Electronic Log Book

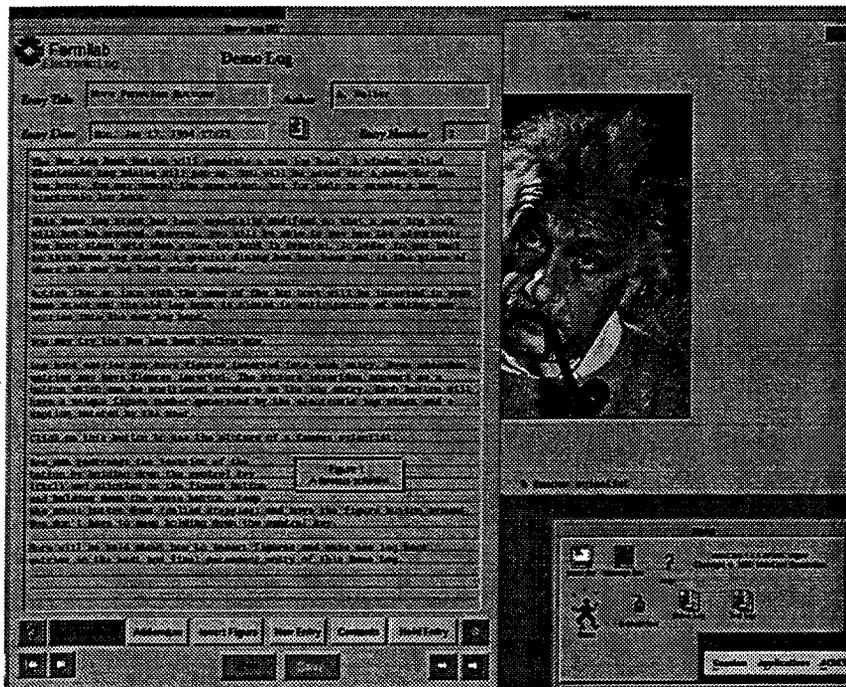


Figure 2  
Button (lower right section in tablet) with figure number and caption serve as link for the real image

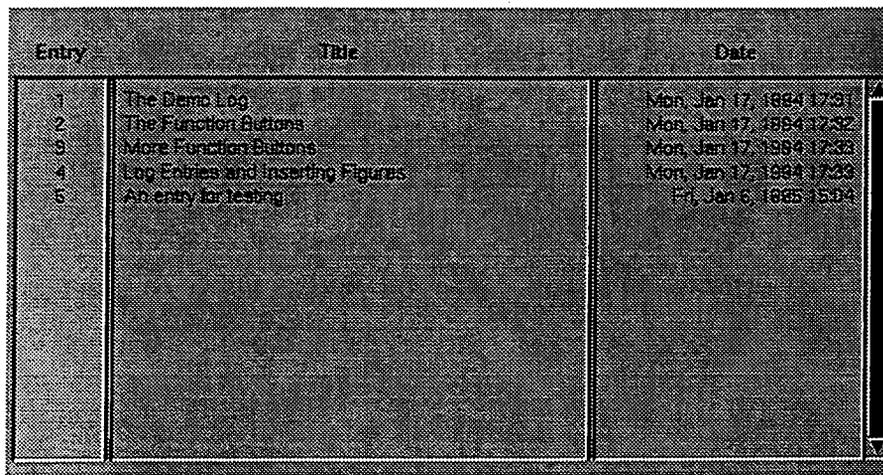
## METACARD EXPERIENCES GAINED THROUGH THE ELECTRONIC LOG BOOK

All text, graphics and MetaCard "code" called scripts reside together as a single entity called a stack. The power of MetaCard lies in utilizing this paradigm. This approach enables fast text searches through an entire electronic log with the embedded MetaCard search engine. No special code needs to be written for reading and searching text records within a file. This makes text searching an integral part of the MetaCard functionality rather than a separate part that either must be programmed each time or provided for by an auxiliary utility. Multiple users of a log book must however share the entire application not just the text (data) associated with it. The electronic log book must use a semaphore to synchronize the writer for a given stack (log book).

As new text is continually added the stack continues to grow. If pictures are stored in a log book this can make for an extremely large stack, and the author of MetaCard suggested storing only one image per stack. Images are stored immediately after the snapshot whereas the text entered into a log book may be cached before being written out. Storing the images separately has proved to be a good idea. They are not embedded within the log book and thus are available separately for perhaps reproduction for a report. Also, on the few occasions where the host or server computer has momentarily been down, the pictures have been recoverable and links able to be re-established to the necessary log book entries.

One of the useful features of MetaCard is the ability to clone all objects, including the entire stack (or application). Cloning an application is not advisable unless the application is really going to be used in a different context. For the electronic log book, each new or different log is cloned from an original copy. If one modifies a script that is a part of the log book functionality, perhaps because of a bug, this modification must be made to all existing stacks of electronic log books.

Initially the MetaCard snapshot utility captured pictures quickly and stored the image efficiently. However this utility could not handle multiple displays as other UNIX image capture utilities could. Others were tried but proved to be slow and/or not very efficient in storing the picture information. Eventually the author of MetaCard extended importing snapshots from other displays.



Entry	Title	Date
1	The Demo Log	Mon, Jan 17, 1994 17:31
2	The Function Buttons	Mon, Jan 17, 1994 17:32
3	More Function Buttons	Mon, Jan 17, 1994 17:33
4	Log Entries and Inserting Figures	Mon, Jan 17, 1994 17:33
5	An entry for testing	Fri, Jan 6, 1995 15:04

Figure 3  
Double-clicking on a log entry in the table of contents will take one to the entry

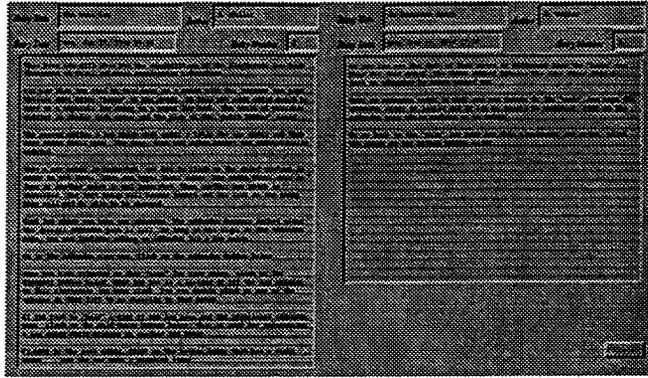


Figure 4  
A reduced window of a log entry and its addendum can be displayed for reference

## CONCLUSIONS

Rapid prototyping of the electronic log book allowed the visual layout to be done within hours. Writing scripts and learning more about the use of MetaCard took at least a few weeks. Most of the time was however spent on the learning curve for MetaCard. A knowledgeable MetaCard user would be able to write the necessary scripts for the electronic log within a few days. The rapid prototyping approach therefore does appear to be attractive for developing applications; especially in a situation where personnel and time are limited.

The image capture is a very valuable feature of the electronic log book. As it turns out, a log book may contain several hundred log entries but usually contains several thousand pictures. The author of MetaCard is to be thanked for his suggestion of keeping images in separate stacks detached from the main log book entry.

There are several concentrated users of the electronic log book for the Booster, Antiproton and Tevatron operations of the accelerator. While the log book is available to all operations, its acceptance has been mixed. This mixed acceptance has been partly because it was well known that this project was an experiment and as such carried an experimental stigma about it. Operations is apprehensive to use an experimental system, and it is constantly thought that this project, though now finished, is still under construction. Also the graphical user interface is quite different from the day-to-day interface operators must deal with. Insofar as this difference exists, the application itself is treated differently and is not warmly accepted as a part of the operational control system. However, the author of this paper believes that this will change in time as more applications with a Motif look and feel become available and a vital part of the Accelerator Division controls console environment.

We are pleased with the seamless integration of the electronic log book. Many people are surprised to discover that the code is actually running on a remote host. Many have thought that the electronic log book was running on the VAX VMS workstations that are used to support Accelerator Division controls consoles.

## ACKNOWLEDGMENT

I would like to thank Scott Raney of the MetaCard Corporation for his many suggestions during the development of the electronic log book. Scott also provided technical answers to my MetaCard questions and provided some bug fixes and technical solutions to accommodate and help make the electronic log book a success.

## REFERENCES

- [1] K. Cahill and J. Smedinghoff, Converting the Fermilab Accelerator Control Consoles to X-Windows Workstations, Nuclear Instruments and Methods in Physics Research, Volume A293 (1990), pp. 442-445.
- [2] B. Hendricks and R. Joshel, Overview of the Next Generation of Fermilab Collider Software, National Laboratory for High Energy Physics, KEK Report 1992, pp. 243-245.
- [3] S. Raney, Interactive GUI Development Environments; A Comparison of Tcl/Tk, the Desktop Kornshell and MetaCard, O'Reilly and Associates, The X Resource, Issue 11, Summer 1994.
- [4] HyperCard User's Guide, Apple Computer, Inc., Number 030-3918-A, 1989.
- [5] Using DECwindows Motif for OpenVMS, Digital Equipment Corporation, 1993.
- [6] K. Cahill and J. Smedinghoff, Exploiting the X-Windows Environment to Expand the Number, Reach, and Usefulness of Fermilab Accelerator Control Consoles, National Laboratory for High Energy Physics, KEK Report 1992, pp. 464-467.
- [7] OpenVMS DCL Dictionary, Digital Equipment Corporation, 1994.
- [8] MultiNet User's Guide, Digital Equipment Corporation, 1994.
- [9] J. Smedinghoff, The Fermilab Accelerator Control System Parameter Program and Plotting Facility, Nuclear Instruments and Methods in Physics Research, Volume A247 (1986), pp. 172-175.
- [10] OSF/Motif Programmer's Reference, Prentice-Hall, ISBN 0-13-640517-7, 1990.

# An Architecture for Intelligent Control of Particle Accelerators

William B. Klein, Robert T. Westervelt  
Vista Control Systems Inc., Los Alamos, New Mexico 87544  
and  
George F. Luger  
University of New Mexico, Albuquerque, New Mexico 87131

## ABSTRACT

In this paper, we discuss results of combining various methodologies from the field of artificial intelligence into the design of a control system for accelerator tuning. Our architecture brings together state space search and rule-based reasoning with adaptive/learning algorithms such as fuzzy logic, neural networks and genetic algorithms. We discuss current efforts extending the system to include a general purpose hierarchical control paradigm, parallel distributed reasoning, an object-oriented reasoning structure and additional heuristic control methods.

## 1.0 PROBLEM OVERVIEW

The goal of this project is to develop a flexible intelligent controller that can reduce the tuning time and the need for human intervention in the control of a particle accelerator. We also wish to produce better and more stable tunes than those that are now achieved by human operators. Various approaches have been taken to automate accelerator control [2], [8], [12], with varied degrees of success. Most effort has been directed toward solving specific problems for a particular facility and little effort has been directed toward developing more general solutions applicable to the diverse specifications and tasks of a number of different accelerators. This paper reports the status of our continuing research including efforts toward building a general purpose control system.

To build an environment for testing our control algorithms we interfaced TRANSPORT [1], a standard accelerator modeling program, to Vsystem [3]. Vsystem is a commercial software product for developing control systems. Vsystem provides a distributed database and tools for accessing real-time data, as well as a graphical environment for display and control of database channels. We began by developing a computer model to simulate steering and focusing elements used in beam transport. Noise and error effects including initial beam jitter, electronic offset and drift and random gaussian noise were added to data signals to approximate actual tuning conditions. Time dependent device behavior was also included in the simulation. Figure 1 depicts the simulated beamline.

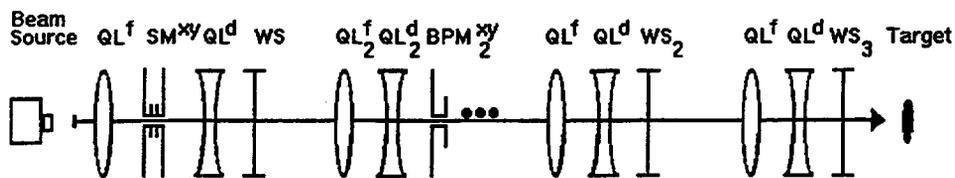


Figure 1. Typical Beam Transport Line. Beam behavior is simulated using transport modeling code. Measurement and control occurs through Vsystem control channels.

## 2.0 CONTROL DESIGN ISSUES

We considered many control design issues during the development of the control system. These include: 1) adaptive vs. non-adaptive control, 2) optimal vs. "good enough" solutions, 3) scalability, 4) determination of failure conditions, 5) online and offline learning, 6) stability in a heuristic control environment. Our

design includes a hierarchical decision maker which determines appropriate control techniques according to the issues listed above. The controller is able to choose from a variety of control techniques and substitute different methods according to the state of the system.

Our design uses an expert system at the top level for reasoning and control. An expert system is a computer program that uses an explicit knowledge base, often directly taken from human experts, along with logical reasoning to solve complex, real-world problems [7]. Placing the expert system at the highest level provides a controller capable of making decisions about the control problem in a global context, without considering detailed issues; context specific subproblems are handled by lower-level control modules. With direct access to the Vsystem control database, the expert system applies all pertinent information to build a model for solving the system and to reason about specific components and more general tuning issues.

We used CLIPS, a forward chaining object-oriented expert system shell [4] for building knowledge structures that represent expert knowledge in both the problem and solution domains. We separated the beamline components into groups by function and control characteristics and then developed partitions that implied certain types of solutions. That is, we developed control structures which included facts and rules about beamline interactions as well as control techniques for solving classes of problems.

We constructed CLIPS objects for beamline components which represented both physical entities and control characteristics. These objects included static information about beamline placement and orientation, as well as methods for data collection and control during operation. Creating an object representation of the system within CLIPS enabled us to place knowledge about a specific component within its representation while maintaining a separate knowledge base representing facts and rules describing the entire system. An object reasoning model allows appropriate encapsulation of knowledge with system objects, modularity of reasoning and the possibility of distributed control.

### 3.0 A CONTROL METHODOLOGY

In this section we outline a number of heuristic methods for control of the partitioned submodules of the accelerator. These control methods include neural (or connectionist) networks, fuzzy logic and genetic algorithms, as well as more traditional analytic methods. In using these heuristic methods we make certain basic assumptions about the control problem based on suggestions from Ross [9]:

- 1) Beamline behavior is observable and controllable. The control techniques used here rely on measurable state input and output variables. Human agents, through constant monitoring can control the system.
- 2) There exists a method for encapsulating knowledge about beamline control within the heuristic methods. This may come from neural network learning algorithms, a priori rule-based knowledge, or inherent knowledge encoded in the genetic algorithm population.
- 3) One or more solutions exist. The set of control variables is sufficient to produce correct beamline behavior.
- 4) A "good enough" solution is acceptable. We will identify a small error range within which all solutions are valid.
- 5) Optimality and stability may be shown through data flow analysis and empirical methods, rather than through formal proofs. Because many of our control heuristics use inexact methods, formal proofs are inappropriate if not impossible.

Keeping these assumptions in mind, we adapted the following techniques for use in accelerator control:

We experimented with multilayer perceptron networks to attempt to learn the relationships between control and feedback components in the beamline. Unlike the traditional use of neural networks in control [8] we did not model beamline behavior, but instead used the network to discover causal relationships between beamline components. Although preliminary efforts were unsuccessful, work is continuing using connectionist systems for direct beamline control [12].

Analytic techniques for control rely on beamline behavior consistent with a simple linear model. A straightforward analytic method makes no attempt to filter noise or eliminate component errors. In general, this method provides an accurate solution given large signal-to-noise ratio and properly functioning beamline components. We cannot expect a purely analytic solution to adequately tune a beamline in most cases, especially during initial startup. Fuzzy logic is used in the beamline controller for reasoning about real-valued data in the presence of noise and in situations where crisp analytic methods have failed. Fuzzy logic attempts to categorize real data sets with ambiguous boundaries. We may consider the data we receive from beamline measuring devices as ambiguous or imprecise, because data measurement involves errors of unknown distribution and sometimes from unknown sources. We can capture a human operator's reasoning about this imprecise data by specifying linguistic variables comparable to the fuzzy sets which match the operator's (implicit) fuzzy categories. We implemented fuzzy logic versions of analytic solutions successfully for noisy steering simulations [12].

Not only do fuzzy rules allow expert systems to reason about real-valued data without crisp data boundaries, they also allow reasoning about how data will be measured and evaluated. An expert system could refine the meaning of a fuzzy control variable during different stages of the solution. This refinement relates to the context dependent nature of fuzzy membership functions and the ability to reuse fuzzy rules in both coarse and fine grain solutions. For example, a rule governing steering behavior may state that a *small* adjustment should be made when the error is *small*. The term *small* may have a different meaning for error and adjustment. Furthermore, as subsequent adjustments continue to decrease the error, we may need to adjust membership functions representing *small* to ensure convergence. The expert system can change the membership function associated with a variable depending on the specific problem being solved, the accuracy required and the current state of the system.

The genetic algorithm offers an appropriate heuristic for focusing control because it can search large solution spaces in non-linear domains. The genetic algorithm is particularly useful when the controller must function using incomplete information or when the system behaves abnormally due to component failure or other unpredictable situations. We implemented a genetic algorithm for beam focusing using genetic operators which modified magnet strengths according to fuzzy patterns. Fuzzy patterns eliminate the need for a priori determination of magnet adjustment strengths. Since typical solution patterns can be determined for focusing, we used a special genetic operator to search the solution population for unwanted solution patterns (as determined by the expert system) and replace them with solutions fitting good patterns. We found that the fuzzy pattern matching solution focused the simulated periodic line in fewer than 100 trials and to a greater than expected accuracy [12].

#### 4.0 A SYNTHESIS OF CONTROL MODULES

By incorporating the solution methods of Section 3 into a single system, we are developing a powerful integrated problem solver that addresses many beamline tuning problems. Modifying existing solution algorithms and adding new solution strategies can enhance the quality and speed of the system within the current framework and generalize our solutions for use on other accelerators.

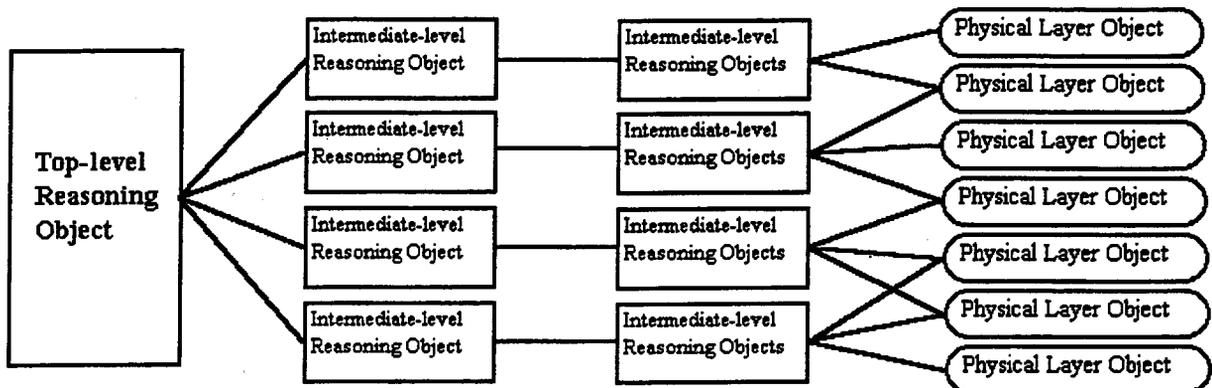


Figure 2. A typical object reasoning hierarchy.

By placing the expert system as the highest-level decision maker for the controller, we use expert knowledge to break the control problem into solvable units and then determine an appropriate solution strategy for various beamline problems. Keeping decision making isolated at a single level, however, causes problems with rule complexity and problem decomposition. Our current system makes use of a structure which represents a control hierarchy at higher levels integrated with a physical component hierarchy at lower levels. Figure 2 illustrates a multilevel reasoning hierarchy.

This modular decomposition of complex problems into multiple interacting subcomponents is central to our approach. The object oriented methodology provides data structures that "wrap" the submodules in a module hierarchy, where each module contains knowledge (coded as facts, rules, and procedures) describing its functionality, as well as sets of methods for cooperating with other modules. Together the interacting modules make up the larger system. This approach to problem solving is called by the artificial intelligence research community a *solution strategy based on the interactions of autonomous intelligent agents* and has been used in a number of different application areas including electricity transport management [6] and building environment control [5].

In the most effective system, an expert system coordinates the activities of a set of independent processes controlling small subsystems of the accelerator. The expert system manages the tuning process by identifying and configuring subgoals based on an overall goal for the accelerator. These subgoals are then either subdivided further or assigned a suitable solution strategy based on the goal and the current operational state. We have found that an expert system equipped with a "toolbox" of control methods can overcome limitations of any one control method by substituting a specific control strategy based on a particular subsystem goal.

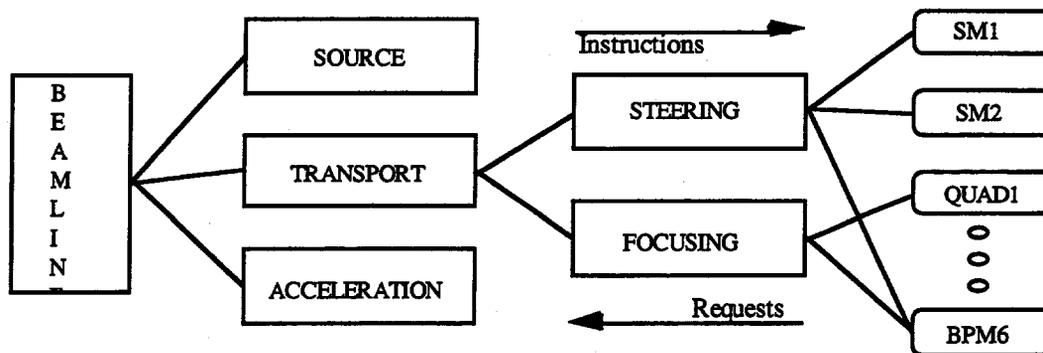


Figure 3. A hierarchy for a simple beamline with transport.

Figure 3 illustrates an example control hierarchy for a simple beamline consisting of a beam source, a transport section, and an accelerator. A top level control object representing the beamline contains high level information about each of its subcomponents (source, transport, acceleration) and their interactions. The beamline object also keeps track of global events and coordinates activity for responding to alarms or errors. The transport object contains knowledge about the transport section and information about interactions of its subcomponents, steering and focusing. Steering and focusing objects contain specific information regarding beamline components and problem solving methods. At each level, an object's parent serves as an intermediary for communication between control components. For instance, the transport object can send a message to the beamline object requesting that the input beam be more accurately centered. The beamline object may then send instructions to the beam source object to request the change or deny the request and send new instructions back to the transport object. Such communication and object interaction may occur at any level in the hierarchy.

## 5.0 SUMMARY AND CONCLUSION

The concept of object models for accelerator systems is a methodology gaining a large following in the accelerator control community. Work has been done at numerous sites to develop an object framework for describing accelerator control applications [11].

An expert system with general knowledge of the control domain exists at the top level for coordination and control of beamline subcontrollers; smaller domain-specific rule sets exist throughout the object oriented component module hierarchy. Distributing knowledge throughout the system has a number of advantages: 1) Rule sets are typically smaller; large rule sets indicate the need for breaking down the problem into smaller components, 2) Knowledge resides at the appropriate level in the system, so control objects can make domain-specific decisions without relying on a higher level control object, 3) Reasoning is faster; the conflict set for any one rule base is smaller and independent activations may be fired simultaneously in the distributed environment.

Further discussion of our research may be found in [12], [13].

## 6.0 ACKNOWLEDGEMENTS

This work was supported by a grant from the Department of Energy under the SBIR program (Grant #DE FG05-94ER81897). We would like to thank Andy Jason of AOT-1 at Los Alamos National Laboratory for his invaluable assistance in developing control scenarios and for his careful explanations of the underlying physics concepts involved.

## 7.0 REFERENCES

- [1] Brown, K., Rothacker, F., Carey, D. C., and Iselin, Ch. (1977). "TRANSPORT: A Computer Program for Designing Charge Particle Beam Transport Systems," SLAC-PUB-91 Rev 2.
- [2] Clearwater, S. and W. Cleland (1989). "A Real-Time Expert System for Trigger Logic-Logic Monitoring," *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems*, Vancouver, B.C.
- [3] Clout, P. (1993). "The status of Vsystem," *Proceedings of the Third International Conference on Accelerator and Large Experimental Physics Control Systems*, Germany.
- [4] Giarratano, Joseph and Riley, Gary (1989). *Expert Systems: Principles and Programming*, Ch. 7, Boston, MA, PWS-KENT Publishing Company.
- [5] Huberman, Bernardo and Clearwater, Scott H. (1995). "A Multi-Agent System for Controlling Building Environments," *Proceedings of the First International Conference on Multi-Agent Systems*, San Francisco, CA., MIT Press.
- [6] Jennings, N. R., Corera, J. M., and Laresgoiti, I. (1995). "Developing Industrial Multi-Agent Systems," *Proceedings of the First International Conference on Multi-Agent Systems*, San Francisco, CA., MIT Press.
- [7] Luger, G.F. & W.A. Stubblefield (1993). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Reading, Ma: Addison Wesley.
- [8] Nguyen, D., M. Lee, R. Sass, and H. Shoae (1991). "Accelerator and Feedback Control Simulation Using Neural Networks," SLAC-PUB-5503.
- [9] Ross, T. (1995). *Fuzzy Logic with Engineering Applications*, New York, NY: McGraw-Hill, Inc.
- [10] Schultz, D. (1977). "The Development of an Expert System to Tune a Beam Line," *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems*, Vancouver, B.C.
- [11] Skelly, J.F. (1991). "Object-Oriented Programming Techniques for the AGS Booster," *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems*, Tsukuba, Japan.
- [12] Westervelt, R.T., Klein, W. B., and Luger, G. F. (1995). "Framework for a General Purpose Control System for Particle Accelerators.," *Proceedings of the IEEE Particle Accelerator Conference*, Dallas, Tx.

# New Man-machine Interface at the BEPC

J. Zhao, C. Wang, B. Wang, X. Geng, J. Xu, H. Luo

Institute of High Energy Physics, Chinese Academy of Sciences

P.O.Box 918-10, Beijing 100039, China

Internet: Zhaojj@bepc3.ihep.ac.cn

The BEPC (Beijing Electron Positron Collider) control system was upgraded in October 1994. Except for low level CAMAC hardware equipment, all high level control computers and their adapters have been updated. The old console with touch panels, color monitors and knobs has been replaced by workstations. This paper describes the methods of developing the man-machine interface under X-windows in a network environment without having to change existing programs greatly.

## 1. INTRODUCTION

The BEPC control system was built by the end of 1987 and has been upgraded in October 1994.

The old control system used a VAX-11/750 computer and the VAX-CAMAC-Channel to connect to the low level CAMAC data acquisition system. The old console has color and B/W monitors for displaying the status of the controlled equipment. A touch panel and knobs were used as the man-machine interface to operate the BEPC machine. A Grinnell controller interfaced the color and B/W monitors to the VAX computer. This controller is no longer manufactured and the lack of spare parts made the central console unreliable. So workstations are now employed as the new man-machine interfaces of the control system.

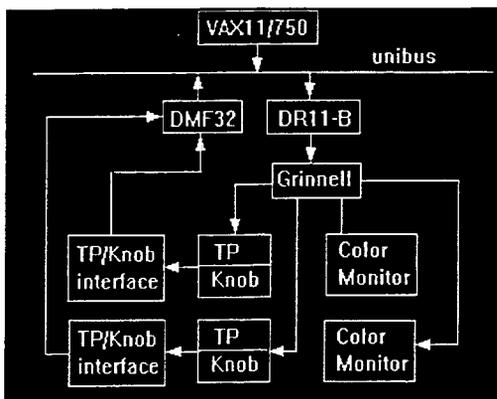


Figure 1 Former BEPC console structure

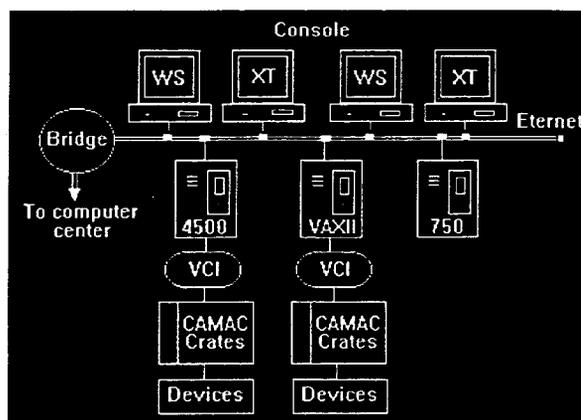


Figure 2 Upgraded BEPC control system

The upgraded control system, as shown in figure 2, adopts a distributed architecture based on Ethernet [1]. Both the VAX4500 and Micro VAXII can independently control all BEPC equipment or control the different devices separately. Two VAX workstations are used as the new console to replace the old one. A mouse is utilized as the operating tool. Ethernet connects all the computers serves for data communication. Our strategy was to avoid having to change low level programs greatly in the development of the new console.

## 2. CONSOLE MANAGER.

The new console consists of two VAX4090 workstations and two X-terminals. The console manager developed under X-Windows and OSF/Motif has a friendly man-machine interface which retains the former style of the control panel to reduce

the training time for operators [2]. The dedicated Grinnell adapter was eliminated; thus the new system is easy to maintain. A new knob control program has been developed, using the client-server mode to control individual devices.

### *2.1. Environment of the console development.*

Jointly developed by MIT and DEC, X-Windows is being widely adopted as a standard by nearly every workstation manufacturer for its extremely convenient user interface, powerful programming ability, high quality code and distributed processing ability, and it has been used in many large-scale control systems.

X-Toolkit is a software package established on the basis of X-windows for the convenience of users to develop interfaces. It contains a large widget set including Motif, which provides many artistic widgets for interface design and makes it possible for the user to do further development as required, and OPENLOOK, which adopts the concept of OOP.

### *2.2. Console database.*

A hierarchical database was installed on the console computer with a DBMS system, DBA, which can read write, print, and backup the data record on a terminal; console manager programs can access the database by a set of subroutine calls. At console startup whole data records, the information on the accelerator devices and console messages, are brought into physical memory from the disk. The console manager fetches the useful information from the database.

### *2.3. Program Realization.*

The hardware console was replaced by a software console, which provides both controls and displays. SLAC has developed a console manager program, the principle of which is as follows: Create a window on workstation and designate four areas (tile) simulating the original left and right touch panels, two color displays and one black and white monitor. When the information with Grinnell codes is sent to the workstation from server program Newavtx running on the VAX4500, it will be encoded into X-Window format and displayed on the workstation by the console manager. The control information from an operator at the workstation is sent to through Ethernet Newavtx, where it can be interpreted and switched to the corresponding application process.

Though this programming ingeniously avoids a great amount of work on improving the lower lever programs of the former control system, it still has some drawbacks: (a) The weakness of XUI: it can not make a perfect interface. (b) Serious shortcomings in operation: the size of "tile" cannot be changed and thus some tiles are too small, the text is overlapped and the operator can not observe the contents of both tiles simultaneously. (c) Inconsistency with developmental projects for the BEPC control system.

For these reasons, the program cannot be used unmodified for practical control of BEPC. Thus we did considerable work to analyze and regenerate it and now we use the Motif widget set. User Interface Language (UIL) is adopted to reduce program crashes caused by mistakes in interface design. Instead of one window divided into four areas, six independent windows represent console devices and the windows can be reduced or enlarged independently to make full use of the limited screen and to make it possible for the operator to get information on several devices at the same time. The windows adopt the Application Shell as their top level window so that they can be turned into icons and stored in the icon box when not in use. Each of the windows can be independently chosen to be shown on a workstation or X-terminal. Thus the inconvenience of overlapping windows is avoided. The improved program retains the advantages of the original and provides the operator with a friendly user interface and convenience of operation.

In addition, the operator can acquire functions by using menus. For example, using the FILE menu an operator can print and quit, the CONTROL menu can control the window state, the OPTION menu can change the foreground or background color, the HELP menu supplies details of operation to the operator.

### *2.4. Knob manager.*

There are four knobs on the old console to trim the currents of magnets individually; these are connected to the VAX-11/750 by an RS-232 port. In the upgraded system, the hardware knobs have been eliminated and a new knob manager program developed, consisting of a knob client program running on the workstation and a knob server program on the VAX4500 computer. A network link is created at start-up between the knob client and the server program in order to exchange data and commands.

The knob client program developed with Motif widgets allows the operator to tune one to four power supplies and to select the up or down speed and size of the steps. The relative values of the current can be shown in the knob windows. The

accuracy of the trim is a single bit. If the current setpoint value overflows, an alarm message will be shown in an information window and the operation will be stopped. The knob server program in the VAX4500 computer sets current values for specified devices. It waits for an event flag sent by an operator through the network. Once the event is set up, the server program packs the I/O commands and data into CAMAC control words, resulting in the setting values being exported to the devices.

### 3. NETWORK COMMUNICATION

DECnet for task-to-task non-transparent communication has been adopted to develop the network manager programs because it can provide data exchange between programs running on different operating systems and provides many network functions. Among such functions are rejection of the network link request, synchronous or asynchronous termination of the net link, management of net link multi-requests, etc. [3]

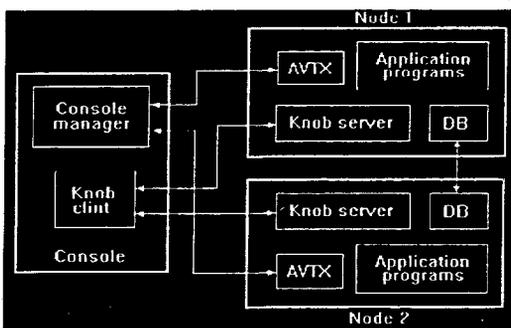


Figure 3 The network links

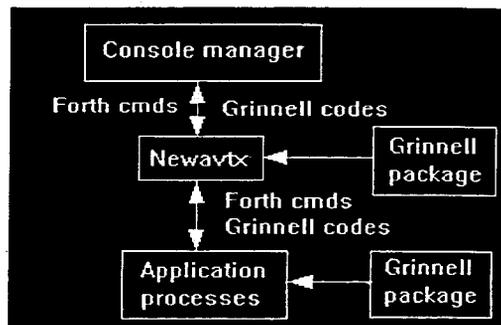


Figure 4 Relationship between the console and the application processes

The network link in our control system is shown in Figure 3. When the control system starts up, three kinds of network links are created by the network manager. One is a network link between the console manager and the multi-task scheduler on the front end computer, which sends commands and data from the console to the FEC and passes the FEC messages to the console manager. The second is between local databases on FEC nodes, to carry out data exchange among these databases. There is a third network link between the knob client program on the console and the knob server program on the FEC. All of above network links are kept open when BEPC is operating.

### 4. CONSOLE SEVER AND SCHEDULER.

There is a program Newavtx, a set of panel files written in FORTH and twenty application processes in the VAX4500 computer. Newavtx is used for scheduling and managing those processes. It may activate or stop any application process on operator request [4] and supervises many processes running simultaneously. Under the control of the program any equipment of the collider may be monitored and adjusted to ensure that the collider operates optimally.

The application programs are divided into two levels. The zero level contains programs for data input/output, on/off and ramping of the magnet power supply and monitoring of vacuum, radio frequency, beam position etc. The first level software contains the programs Orbit, Current Monitor, Lattice and so on. The program Newavtx is responsible for scheduling those processes. Because Newavtx provides the network communication by making a logical link with the console manager netlink, it may receive the data from the console through the network. Thus it is also a sever program for the console. Its other duties are: (1) receiving network commands, (2) interpreting and executing FORTH commands, (3) scheduling application programs, (4) diagnostics.

Here, for simplicity we only discuss some key problems.

#### *4.1. Data communication between the console and application processes.*

By creating a logical link with the console manager using task-to-task nontransparent communication, the program Newavtx may receive or reject the netlink request from the console manager, or synchronously or asynchronously terminate the netlink. The maximum number of remote nodes is six. In addition, by declaring itself to be a "network object", Newavtx not only receives the data with FORTH command format from the console, but also sends the data, mostly Grinnell codes, to the console. Thus it connects the console with the application processes. A mailbox created by Newavtx is used for the exchanger of data between the console and the application processes .

#### *4.2. Interpreting commands and scheduling*

Newavtx interprets the FORTH command from the console to active appropriate application processes and sends back the result to the console. So it connects the console with the application processes. Two kinds of data are distinguished by an event flag. One kind is for FORTH commands and the other is for Grinnell codes. The FORTH commands interpreted by Newavtx come from the two sources: the console panels and some application programs. The Grinnell codes were previously encoded by the Grinnell hardware adapter. To avoid great changes in the application programs, they still use the Grinnell codes and send these codes to Newavtx. The console manager converts the Grinnell codes to the X-window graphic ones.

Newavtx waits for an event flag. Once the event flag N-efn is set up by a network transmission, Newavtx fetches the console commands and sends them to the interpreter subroutine. The appropriate process is activated after these commands have been translated and executed. When an application process wants to send some graphic information to the console, it assembles Grinnell codes by using the Grinnell graphic library and sends them to the Newavtx mailbox with the event flag set up. As soon as Newavtx receives the event flag , it fetches the data and sends them to the console manager. The related graphic will be displayed in the console window after the codes are translated into X-Windows graphic codes by the console manager. If some application processes need to send text messages about the current operation status or error to the console panel, they must assemble the data in FORTH command format and send them to the Newavtx mailbox with the event flag set up. When Newavtx receives the event flag, it fetches the data and sends them to the interpreter subroutine. After these FORTH commands are translated and executed, they are changed into Grinnell codes. Finally the Grinnell codes are sent to the console manager by the network. The text messages which will be shown in the console panel for these codes are converted by the console manager.(figure 4 )

#### *4.3. Display console panel.*

There are six windows on the console. One is used to display panel files, the other five are used to display the status and messages from the controlled equipment. There are two kinds of buttons on the console panel, the panel selecting button and the control buttons. With the control buttons, the commands and data can be issued for controlling the collider. There are about two hundred panel files written in FORTH which are resident in the VAX4500 computer. At control system startup, Newavtx sends an index panel to the console. When the panel selecting button is pushed, the console manager sends the Forth commands to Newavtx through the network. Newavtx receives the commands and sends the selected panel to the console manager for display.

In the upgraded system, one VAX4090 workstation communicates with two nodes, VAX4500 and VAX-II. So network links are created between these nodes. A console manager program is located on the VAX4090 and two Newavtx of same version are running on the VAX4500 and and VAX-II. When debugging, we found the console panel "flashed" since the panel files from the two different computers appeared simultaneously on the screen. In order to solve the problem, we modified the program Newavtx in the VAX-II by setting a switch to prevent the VAX-II from producing panel files.

## 5. CONCLUSION

The upgraded BEPC control system runs safely and reliably. The system has been successful. The new workstation console with a friendly man-machine interface has replaced the former console. The operators can operate the machine in the same manner as before, the only difference being the use of a workstation and mouse instead of a touch panel.

## ACKNOWLEDGMENT

The author wishes to take this opportunity to thank Dr. Sam Howry, who gave us much help. We would like to thank all the members of our control group. The author gratefully acknowledges the support of the K. C. Wong Education Foundation, Hong Kong.

## REFERENCES

- [1] J. Zhao et al., Nucl. Instr. and Method in Phys. Res., (1994)A352.
- [2] J. Zhao, B. Wang, Int. Conf. on Elec.and Infor. Tech. (1994) 337
- [3] J. Xu, Internal Report.
- [4] C. Wang et al., Int. Conf. on Elec. and Infor. Tech. (1994)112

# Simulation Using Xorbit with EPICS\*

Kenneth Evans, Jr.  
Accelerator Systems Division  
Argonne National Laboratory, Argonne IL 60439

## ABSTRACT

The accelerator code Xorbit has an interface to the Experimental Physics and Industrial Control System (EPICS). This means that machine data such as magnet settings can be sent to Xorbit via EPICS and the resulting orbit parameters such as beta functions, etc., can be calculated. In addition, Xorbit can be made to simulate the real machine, whether the latter is running or not. To accomplish this for the APS, there is a database of process variables in an IOC corresponding to each APS ring and beamline. These process variables are very similar to the real process variables that read and set power supplies and read monitors, except that when a setting is changed, Xorbit is notified via a callback, calculates a new orbit, and outputs the appropriate readbacks to the database. By attaching the string "Xorbit:" to a control name the control system will respond to the simulation rather than the real system. This allows the testing of control algorithms, orbit diagnostics and many other components of the control system (as well as EPICS itself). It is fast enough to be visually similar to accessing the real system.

## I. INTRODUCTION

There are a number of reasons to have an accelerator code that is directly linked to a control system. Physics application programs, such as orbit control, may be developed and tested even before the machine is operational. After the machine is operational, these programs may be tested without using valuable machine time and without the possibility of damaging the machine. In addition, when the machine is running, magnet settings can be readily transferred to the code for analysis and the actual orbit compared to the calculated orbit. The effect of possible changes in magnet settings can be easily tested and evaluated. Theoretical quantities such as beta functions, dispersion, phases and tune, which often may not be readily available from the instrumentation (especially during the early stages of commissioning) become available, either in lieu of the real values or for comparison with them when they do exist.

It is an additional benefit when the orbit code has a graphical interface so that these data are displayed as soon as they are calculated. A history mechanism is useful for storing results and comparing the time evolution of important parameters, such as the tunes. These are capabilities that are not found in most of the standard physics lattice codes, such as MAD [1], however.

The most difficult problem in connecting an accelerator code to the control system lies in the fact that in the code, or for almost any orbit calculation for that matter, the orbit is calculated given the magnetic fields in the magnets, while the control system does not know these magnetic fields directly. The control system only sets and reads the currents in the power supplies that control the magnets. Consequently, it is necessary to know the somewhat complicated relationship between the currents and the magnetic fields to make the conversion. This problem is further compounded by the fact that this relationship has hysteresis effects, depending on the past history of the magnets.

This paper describes Xorbit, a graphical accelerator code that has been developed at the Advanced Photon Source (APS) and how it is linked to the EPICS control system in use there. Familiarity with EPICS is assumed. Section II describes Xorbit and Section III describes how it is connected to EPICS. Section IV describes the database that holds the Xorbit process variables (PVs). Section V discusses the problem of the

---

\* Work supported by U.S. Department of Energy, Office of Basic Energy Sciences under Contract No. W-31-109-ENG-38.

conversion between current and field, and Section VI describes a special EPICS record type, the IK record, that performs the conversion.

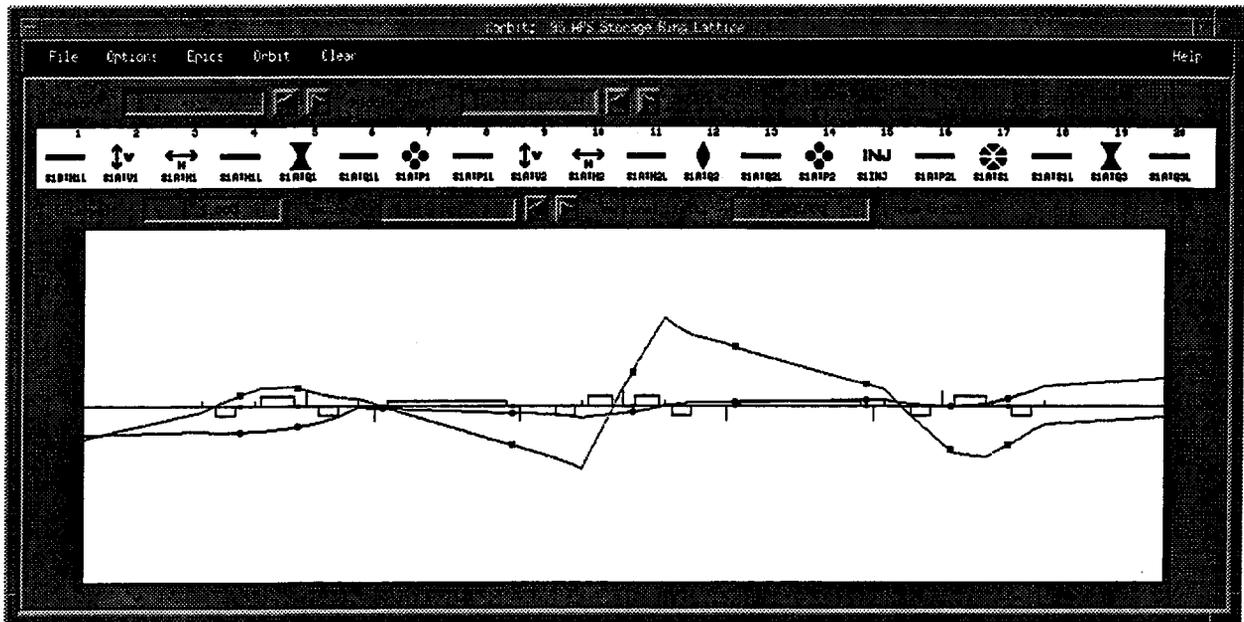


Fig.1 The Xorbit main window. Twiss parameters and a tune diagram are displayed separately.

## II. XORBIT

Xorbit [2] is a UNIX computer code designed to help in the design and operation of circular accelerators, storage rings and beamlines. It is a lattice design code that calculates the orbit primarily by using  $2 \times 2$  transport matrices. Nonlinear elements are simulated with kicks and random element displacements are supported. Dispersion is obtained from the  $2 \times 2$  matrices. It has been designed to be fast and easy to use and to provide immediate feedback when changes are made in the lattice. There are many options and features that will not be described here.

Xorbit has a GUI interface based on Motif and is designed for ease of use and immediate feedback. The main window is shown in Fig. 1. After entering the lattice via an input file, almost all of the interaction with Xorbit is via the mouse and keyboard. All of the results including the orbit and Twiss parameters (beta functions, tunes, *etc.*) are readily displayed in graphic form along with previous values of these quantities if desired.

Several of the principal APS physics application codes, such as Orbitscreen [3] and ADT [4] (which both display data such as monitor readings) and Corbit [5] (which does orbit correction), have an option to use Xorbit data rather than data from the real machine. This means, for example, that orbit correction can be done with the same code whether simulated or real and the resulting monitor readings during correction can be viewed the same way in both cases. There is very little conceptual difference in using the simulation compared to working on the real machine.

There is also a means to connect programs, such as Mathematica [6], or spreadsheets like Wingz [7] or Excel [8] to the orbit calculations in Xorbit. This capability, which is discussed in Reference 2, is actually an imitation of EPICS that uses UNIX interprocess communication (IPC) instead of channel access. It has the same channel access function calls and can be used somewhat interchangeably with EPICS tools [9-10] that connect to these same programs. (Re-linking is necessary.) These connections can be made at the same time as the EPICS connections are in effect and extend the capability to analyze data from EPICS.

### III. CONNECTION TO EPICS

In what follows we will use the mnemonic BPM for beam position monitors (hardware diagnostics that measure the position of the beam in the longitudinal directions, x and y) and just monitors for EPICS monitors (software which determines when a PV has changed).

The basic operation of Xorbit with EPICS is as follows: The user of Xorbit controls the EPICS connection via the EPICS menu after Xorbit is up and running with the lattice defined in the input file. When the EPICS connection is started by the user, Xorbit starts channel access and searches for the necessary PVs. It then calculates an orbit (the closed orbit for a ring or the trajectory corresponding to specified initial conditions for a beamline) and puts the orbit coordinates into the BPM readbacks and the magnet settings into both the magnet readbacks and the setpoints (all of which are PVs). It also establishes monitors on the magnet setpoints. Thereafter, when a channel access client changes the setpoints, Xorbit is notified via the monitors, calculates a new orbit and changes the BPM and magnet readbacks accordingly. Notice that Xorbit sets readbacks. It is simulating the hardware.

The orbit calculation is nearly instantaneous for the smaller APS rings and beamlines, but takes a few seconds for the storage ring on a SPARCstation 10.

The user of Xorbit can also change magnet and other parameters himself inside Xorbit, or they can be changed through connections to Mathematica, etc. All of the Xorbit features are available. As the orbit and magnet settings change, the BPM and also magnet readbacks are reset. In most cases the setpoints are also reset so they will be consistent. The user can explicitly reset all the setpoints. The mechanism for resetting the setpoints is described in the next section.

For those interested in more technical programming information, these simultaneous channel access and X-windows events are handled as follows: A monitor event triggered by channel access calls an Xorbit routine that changes the appropriate variable inside Xorbit and sets a switch that something has changed. A work procedure (one that runs only when Xorbit is not busy) does the resetting of the readbacks if there has been a change. A routine is also established that is called when activity occurs on a UNIX socket attached to channel access. This routine gives channel access a chance to calculate. In addition, connection monitors and associated Xorbit callback routines are set up in a similar fashion to handle lost connections and other channel access problems. All of these are handled by the X event loop, along with the user mouse and keyboard activity, in the order received.

Because of the need to convert between current and magnetic field, the setting of setpoints and reading of readbacks is more complicated than has been indicated. The details are discussed in Sections V and VI.

Since all of the Xorbit PVs and all of the real machine PVs are simultaneously available from the control system, it is straightforward to write a program that reads the real values and puts them into the Xorbit setpoints, causing the orbit in Xorbit to be that predicted by the current machine settings. One of the EPICS menu items in Xorbit runs programs like this. The result is that the current machine values are easily "dumped" into Xorbit. One could go the other way, but this has not been implemented because of its potential for accidental misuse. The values that are read in this procedure are the magnet setpoints, rather than the readbacks, because this is a more stable and reproducible procedure.

### IV. XORBIT DATABASE

The Xorbit database has a PV corresponding to each BPM readback and each magnet readback and setpoint in each of the APS rings and beamlines (with the current exception of the BTS and BTX beamlines). These process variables have the same names as the ones in the real machines, except that the prefix "Xorbit:" is attached to each one. For example, S1A:H1:CurrentAO is the name of a real magnet setpoint and Xorbit:S1A:H1:CurrentAO is the name of the corresponding Xorbit setpoint. Thus, the applications programs such as ADT that have an Xorbit option only have to add this prefix to their PVs to deal with Xorbit rather than the real world.

In addition, there are two process variables to handle the current-to-field-strength conversion for each magnet, one for going from Xorbit and one for going to Xorbit. Xorbit and similar codes actually use the

parameter  $KnL$  rather than the field strength directly. Consequently, the conversion is made between  $KnL$  and the power supply current  $I$ .  $KnL$  is defined to be  $BnL/B\rho$ , where  $B$  is the main vertical dipole field in the device,  $BnL$  is its  $n$ th partial derivative with respect to  $x$  ( $n$  is 0 for a dipole, 1 for a quadrupole, 2 for a sextupole, etc.) integrated along the longitudinal axis, and  $\rho$  is the gyroradius. The quantity  $B\rho$  is related to the energy by  $B\rho[T\cdot m] = 3.34E[GeV]$ . The  $KnL$  and hence the calculations in Xorbit do not depend on the energy. There needs to be one more PV for each ring and beamline to hold the value of  $B\rho$ . In addition, there are also two PVs for each ring that hold the readback values for the tunes.

The total number of process variables necessary to simulate all of the APS rings and beamlines is about 7500. These comprise the Xorbit database and are stored in one input/output controller (IOC) with a 16MB, 25MHz Motorola 68040 board.

The BPM PVs use a standard analog input (AI) record. The magnet PVs are more complicated because of the conversion: The original implementation, which is still used for the booster, is to use a standard calculation (CALC) record for the conversion as shown in Fig. 2. The input for the CALC record is described in the next section. In order to get greater accuracy the CALC record has been replaced by a special current/ $KnL$  conversion (IK) record, described below, in the other cases. Xorbit writes to the VAL fields in the xxx:KAI and xxx:KAO records (where xxx stands for any particular magnet) and monitors xxx:KAO. The conversions to the xxx:CurrentAI and from the xxx:CurrentAO happen automatically because of the forward links (FLNKs).

For either type of record, the setpoints are set equal to the readbacks by setting the OMSL field in the AO record from supervisory to closed\_loop, processing the record, then changing the OMSL field back. This causes the record to get its value from the DOL input, which is attached to the readback.

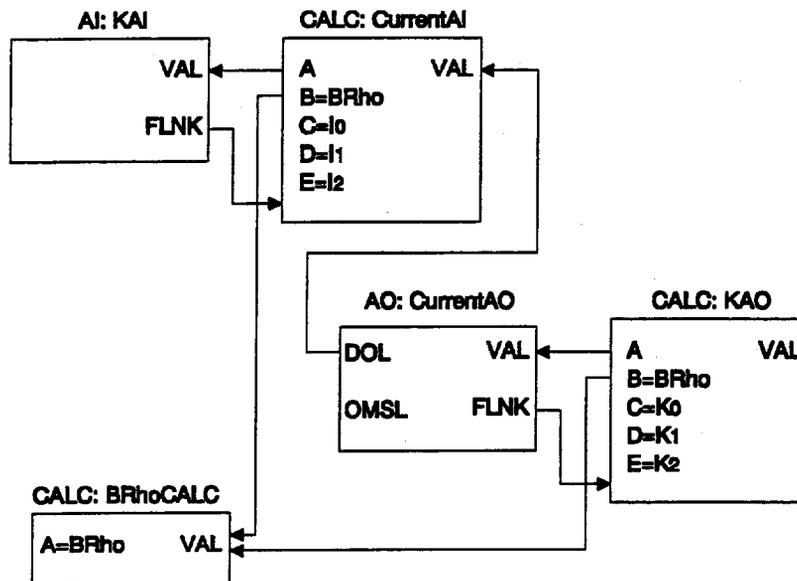


Fig. 2. The record structure for a magnet in the Xorbit database.

## V. CURRENT VS. FIELD STRENGTH

The relationship between the current and the resulting field strength for each magnet in the APS was measured [11] before it was installed. From these data, files were created [12], one for each magnet, with sets of values for  $BnL$  vs.  $I$ . These data are called excitation curves, though tables would be more appropriate. It is the excitation-curve data in these files that is used by Xorbit.

The earliest attempts to do the conversion just used a proportionality constant based on the single point corresponding to the design reference value for  $BnL$  and the corresponding current. When the magnets were set differently from the reference value (as they often were), it quickly became apparent this was not accurate enough.

The next approach was to make quadratic fits of the excitation-curve data. The fits were made to determine  $I$  as a function of  $BnL$  and the fits to  $BnL$  as a function of  $I$  were inverted from these so the inversion process is 1:1 and hence consistent. This gave a satisfactory conversion for machines up to the storage ring; that is, the orbit dumped from the machine was very close to the predicted orbit as were the beta functions and other parameters, including the tunes, which are sensitive to magnet, especially quadrupole, changes.

The excitation files were not made for the booster. The reason is that the booster values are time dependent and does not have or need the precision of the other machines. The fit coefficients used for the booster correspond to a linear fit passing through zero and the reference value.

Excitation Curve for S1A:Q1

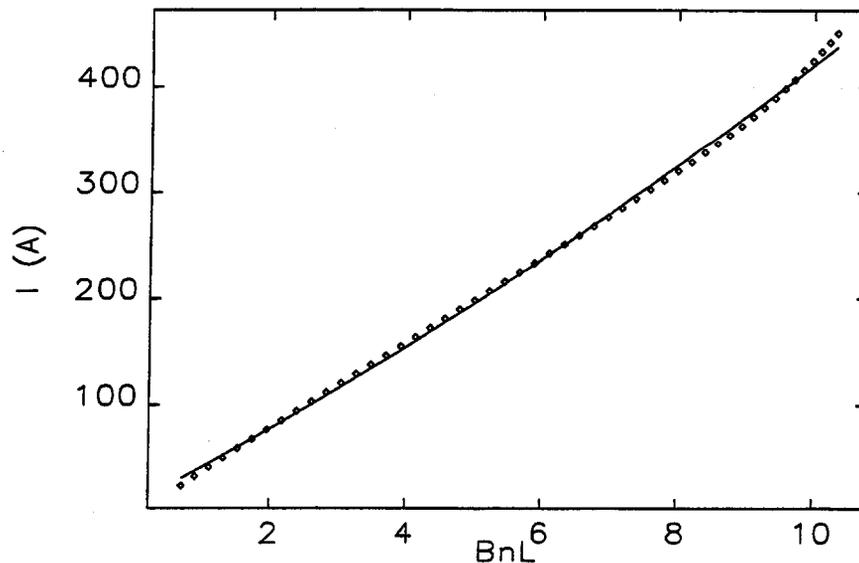


Fig. 3. Excitation curve points for a storage ring quadrupole and the quadratic fit to them.

For the storage ring these fits, though they are reasonably close as seen in Fig. 3, do not give accurate values for the tunes. In fact, the integer parts of the tunes are off by about two integers and the fractional parts are unrelated to the real values. (In practice the integer part of the tune is determined separately from the fractional part and the fractional part can be measured to three figures or more.) The reason for agreement elsewhere and not in the storage ring is that (1) there are many more magnets in the storage ring and (2) the quadrupoles (the main influence on the tune) are stronger. This is verified by the fact that the empirical values could be reproduced by small, systematic changes of the quadrupoles inside Xorbit. It seems apparent that to get sufficient accuracy in the conversion, it is necessary to do a table lookup on the values in the excitation curves. Since the excitation curves have on the order of up to 50 points, it is not possible to do this with a CALC record. Consequently, a new record, the IK record, was developed to do the conversion using table lookups.

## VI. IK RECORD

In order to do the conversions using a table lookup, it is necessary to have a record that includes fields for the  $I$  and  $BnL$  arrays, which may contain a variable number of points. The IK record which was developed is similar to the standard CALC record with the addition of the arrays. The record structure for the Xorbit database is shown in Fig. 4.

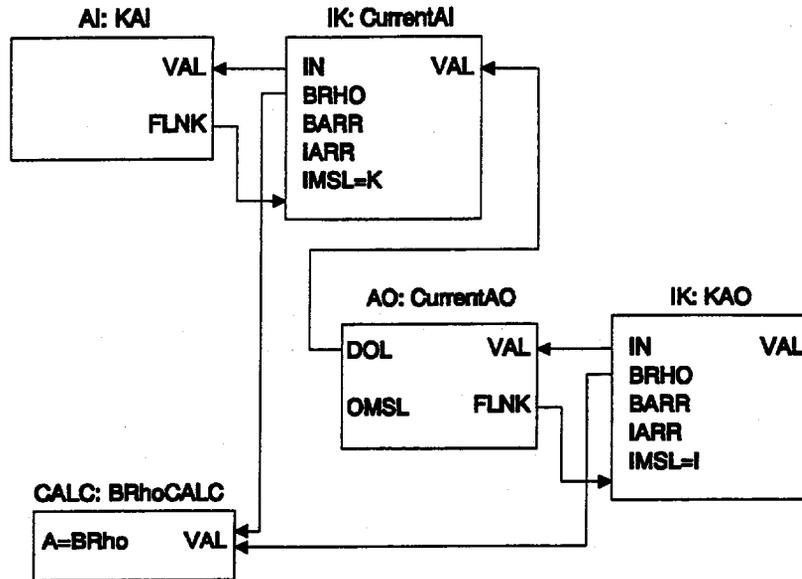


Fig. 4. The record structure using the IK record.

The input field is INP or IN (corresponding to INPA and A in a CALC record). This may be either a KnL value or an I value, depending on whether the IMSL (input mode select) field is I (convert I to KnL) or K (convert KnL to I). The BRHO field is an additional input. The arrays are held in BARR and IARR. There is currently no way in EPICS to specify initial values for array fields. Therefore, the values must be stored in the arrays during or after record initialization. For the IK record there are two options. There are two string fields, PATH and FILE, that allow you to specify a file from which to load the data. If FILE is blank, then the record looks for a routine called getIKArrayValues(type), where type is any string. Users of this record can use this function to load the arrays using their own method. There is also a string field, TYPE, in the record that can be used to hold the string.

For the Xorbit database the excitation files are processed into C programs where the data are hard coded in static arrays along with the names of the magnets. The C programs create hash tables for the data and getIKArrayValues gets the data from the hash tables using the magnet names, which were stored in the TYPE field.

## REFERENCES

1. H. Grote and F.C. Iselin, "The MAD Program," CERN Report No. CERN/SL/90-13(AP), 1991.
2. K. Evans, Jr., "Xorbit -- An X-Windows Accelerator Simulation," Proc. Computational Accelerator Physics Conf. 1993, Pleasanton, CA, February 22-26, 1993, p. 450.
3. K. Evans, Jr., "Orbitscreen Reference Manual," World Wide Web URL:  
<http://www.aps.anl.gov/asd/oag/oagSoftware.html>.
4. K. Evans, Jr., "Array Display Tool ADT Reference Manual," World Wide Web URL:  
<http://www.aps.anl.gov/asd/oag/oagSoftware.html>.
5. K. Evans, Jr., "Corbit Reference Manual," World Wide Web URL:  
<http://www.aps.anl.gov/asd/oag/oagSoftware.html>.
6. S. Wolfram et. al., *Mathematica* (Addison-Wesley, Redwood City, CA, 1991).
7. Computer code *Wingz* (Informix Software, Inc., Lenexa, KS).
8. Computer code *Excel* (Microsoft Corp., Redmond, WA).
9. B-C. Cha, "WingZ/CaWingz," World Wide Web URL:  
[http://www.aps.anl.gov/asd/controls/software\\_tools.html](http://www.aps.anl.gov/asd/controls/software_tools.html).
10. B-C. Cha and R. T. Daly, "CaMath User's Guide," World Wide Web URL:  
[http://www.aps.anl.gov/asd/controls/software\\_tools.html](http://www.aps.anl.gov/asd/controls/software_tools.html).
11. S. H. Kim, et al., "Statistical Analyses of the Magnet Data for the APS Storage Ring Magnets," Proc. 1995 Particle Accelerator Conf., Houston, TX, May 1-5, 1995 (to be published).
12. L. Emery, private communication.

# THE ACQUISITION AND DISPLAY OF BEAM PROFILES

L.David, P.Duneau, E. Lecorche, P. Lermine, J.Vila, C. Maugeais, M. Ulrich

GANIL BP 5027 14021 CAEN FRANCE

E-mail : David@ganexp.in2p3.fr

## 1. INTRODUCTION

The adjustment of the ion beam at GANIL requires the knowledge of its shape (profile). Until recently, this was provided by analog signals acquired through CAMAC interfaces and displayed on an oscilloscope. For weak signals, a very long integration time was necessary and as only one plane was acquired at each tick of the clock, there was a considerable delay when eight profiles were displayed together. No numerical values were available to be used by tuning programs for alignment or adaptation. The effect of adjustments and comparisons between two profiles could only be judged visually. The recording of the profiles could only be made by taking photographs.

The profile measurements are made by using multiwire grids [1], gas jets and microchannel plates [2]. The electronics group has developed a new electronic interface for the multiwire grids and the physics group interfaces for the other two types. Computations are performed on the data from these interfaces to get numerical values and profile shapes, which are then transmitted through the network to the main control room, to be displayed on workstations and used by different beam tuning programs.

## 2. CLIENT/SERVER PATTERN

### 2.1. *New VME processors*

In the past, all the thirty front-end controllers were identical, consisting of a CAMAC crate fitted with a number of general purpose modules, mainly controlling equipments such as motors, probes, power supplies etc. Each has a processor of type RTVAX 300, embedded in a Kinetics 3968 module. The operating system is VAXELN and the network protocol is Decnet.

We decided to use VME for the new multiwire profile system, using some special-purpose modules for the profile acquisition. To be able to use the same operating system and network protocol, we use the AEON VME300 board, which incorporates an RTVAX processor.

### 2.2 *Database*

Although the management of the profile data is carried out in a much more specific way than for the other front end controllers, the data follow the protocol used in the former drivers. These data, whatever their types, are integrated into the INGRES database from which, at boot time, they are extracted as files and loaded into the VME crate.

As this project is the first integration of VME into our control system, it has required the installation of the whole set of functions needed to manage the VME bus, added to the profiler acquisition program itself. Now these specific VME mechanisms are available for the next installation to come, such as new power supplies and ion source devices which will also be controlled through VME.

### 2.3 *Architecture*

This project was easily integrated in the client-server architecture that had been defined from the beginning of the new control system. The control workstations are always clients of the CAMAC crates in charge of reading or writing values in the equipments.

In this profile project, the acquisition and the management of numerical values are carried out by a VME crate used as a data server. This crate distributes the data through the network to each client program on a station in the control room or in the physics division. Therefore the same program can be used on different stations to get the profile data and several different programs (adaptation, alignment, etc.) can ask simultaneously for different data sets.

The network diagram is shown in Figure 1.

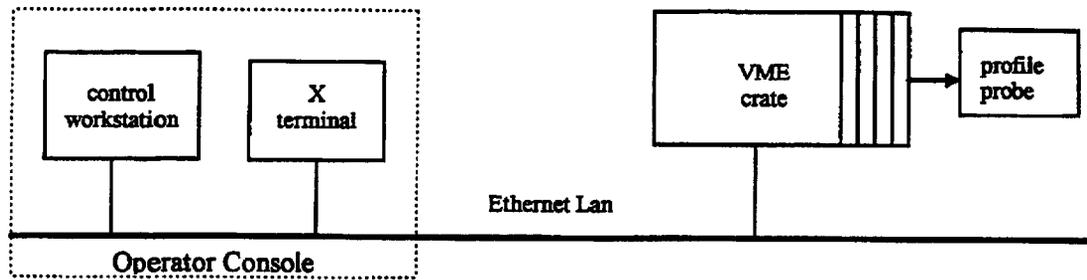


Figure 1. Network Diagram.

### 3. PROFILE PROBES

#### 3.1 *Multiwire profiles*

These consist of metallic grids with planes of 47 horizontal and 47 vertical wires. The grid is located perpendicular to the axis of the beam and each wire measures the intensity received over its entire length. These profilers stop a part of the ion beam (5 to 10%) and can't be used for high intensities.

#### 3.2 *Gas ionization profiles*

These consist of standard CHARPAK chambers filled with Ar-CO gas, with horizontal and vertical metallic grids inside with gaps of 1 mm between wires. Each wire grid is situated between two electrodes. The beam ionizes the gas and the positive ions are captured by the negative electrodes and the electrons by the wires. The ionization rate depends on the intensity of the beam and also on the high voltage applied to the polarization electrode (2500 V maximum).

These profilers are used with small beam intensities, between  $10^2$  particles/s to  $10^7$  particles/s. They stop almost the whole ion beam and thus must be moved in or out by pneumatic propellers.

#### 3.3 *Microchannel plate profiles*

These detectors use the beam ionization of the residual gas, and particles are recovered on microchannel plates for electron amplification. The gap is 1 mm; the high voltage (1500V maximum) and integration time are adjustable. These profiles have a great dynamic range, 1nA to several  $\mu$ A and they don't stop the beam, though they can easily be destroyed by too high a voltage.

#### 3.4 *Spiral scanner profiles*

For medium energy beams the multiwire or microchannel profiles cannot be used because of their fragility and neither can the gas ionization ones because the high voltage creates beam steering.

In cooperation with the Dubna Laboratory new profilers have been tested. They are composed of a helical shape wire in a cylinder that is angled at  $45^\circ$  to the axis of the beam. While turning once, the wire will have covered the entire vertical and horizontal apertures, while intercepting very little beam and without damage because of its size (1mm  $\varnothing$ ).

### 4. ELECTRONIC ACQUISITION

Three VME crates are dedicated to this acquisition which operates as a stand-alone remote process. One is used for the accelerator multiwire profilers, the second for those in the experiment area and the last for the special (microchannel, gas ionization, spiral scanner) ones.

#### 4.1 *VME Acquisition*

The two crates used for multiwire profilers include VME controllers based on RTVAX processors and memory interface and data acquisition boards. The physical profilers are connected to the acquisition boards, each of which can manage 8 planes (4 profilers). The crate contains 20 boards in order to access the 80 profilers of the beam transport lines. The integration periods are externally triggered and the A/D conversion is to 12 bit accuracy. The acquisition is carried out in approximately 150 ms for 160 profiler planes. At each acquisition cycle data are transferred to a shared memory board, and an interrupt is sent to signal that a new set is available for the processor.

## 4.2 Serial Acquisition

The VME crate used for the ionization gas and microchannel plate profiles contains the same VME controller and a board for serial-bus access to the G64 crates involved in the acquisition, Figure 2. Due to the two levels of computing, it takes 200 ms to dispatch data up to the G64 crate, and another 200 ms to send them to the VME crate (speed 19200 baud). As the serial JBUS line uses a master/slave protocol, these times have to be multiplied by the number of profilers that are going to be read.

The four serial ports of the board are used in order to acquire several profiles at the same time.

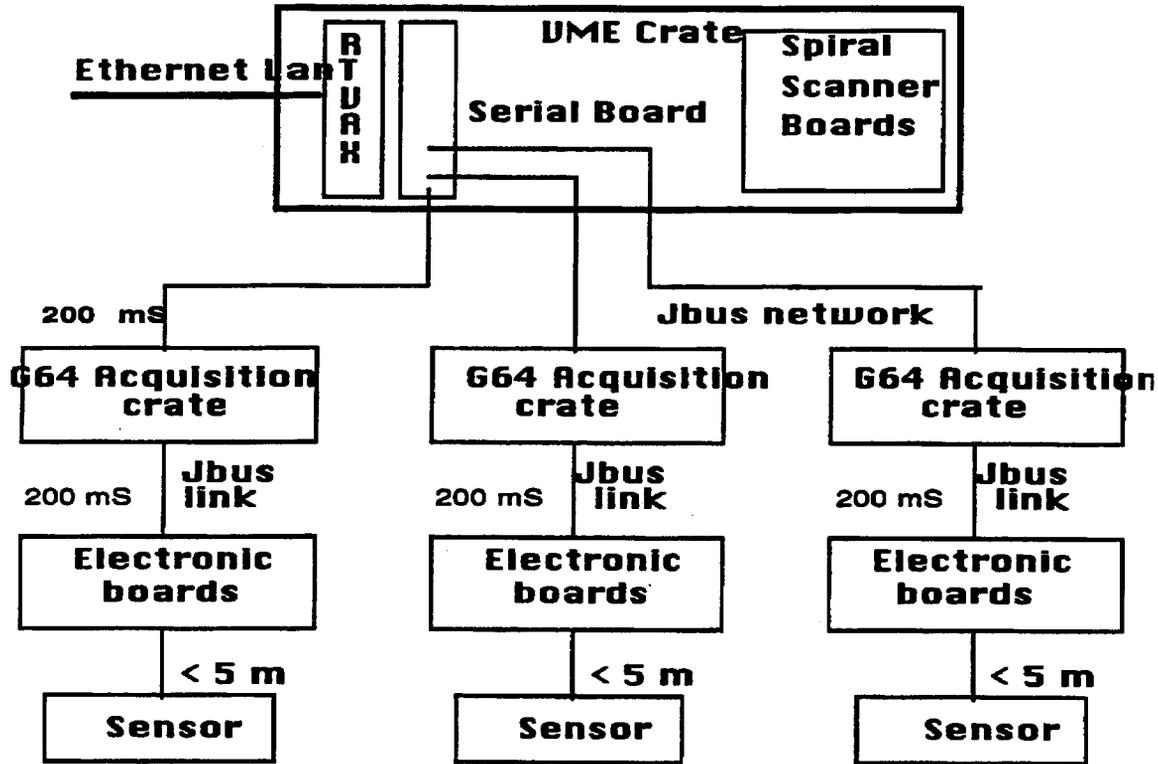


Figure 2. Schematic for serial data acquisition.

## 5. ACQUISITION SOFTWARE

### 5.1 Data Acquisition

In the control system, ADA was chosen as the common language, both for the workstation and the real-time crates, where the multitasking is commonly used.

The main package is built with two tasks, together with generic procedures and the starting entry for the other tasks. The first task is a loop for the acquisition of the data from 160 planes. The other is the main part of the application. It reads the names of the chosen profiles from a mailbox filled by the network, and returns values by sorting the profiles and calling appropriate tasks for the different functions available.

For the gas ionization and microchannel plate profiles, a systematic reading of the requested profiles is carried out by a polling program, and inserted into a DATA COLLECTOR. When a workstation asks for a profile, the DATA COLLECTOR is scanned and data are sent immediately, however those transferred correspond to a measurement having taken place slightly earlier. A generic task was written to obtain or store profile values, whatever their types. By instantiation, it becomes one task to read the multiwires and an other for the microchannel plates, using different drivers.

These tasks can be used for:

Refresh reading	a full polling cycle is made before sending the data
Immediate reading	reading from the data collector
Offset reading	reading the offset value

Measure offset	a new measure of the offset average value
Clear offset	set the offset to zero
First reading	new list of profiles to refresh

## 5.2 Computing

After data acquisition the ELN process undertakes some computations so as to provide numerical values to client programs on the workstations:

- center of gravity
- width at half height
- width at the foot
- surface area

When a wire is broken, computations undertaken on measurements using it will be wrong. In this case it is useful to obtain a reconstituted profile by interpolating the curve between the previous and following wires. This approximation will be signaled on the display, because it is necessary to be sure that the wire is really lacking and that one isn't observing an actual weak value.

The electronic acquisition generates background noise that is added to the measured signal. It is possible, when a profiler has been pulled out of the beam, to measure only such noise. For normal measurements these offsets, that are stored, are automatically subtracted from the values for the corresponding profiles. The offset measurement has to be under operator control (it is necessary to pull the corresponding profiler out of the beam) because any change in profiler configuration can nullify the meanings of the values. Therefore a special function exists to set the offset to zero.

## 5.3 Access Lists

So as to decrease the network traffic, all the means of access to the profile crates have been made by list, to obtain data from several simultaneously. In the case of multiwire profiles, all are read every acquisition cycle but only the ones requested are sent back to the client. In the case of gas or microchannel plate profiles, only the requested ones are read by the data collector, in order to limit the JBUS line traffic. The ADA/VMS package contains about twenty procedures that are used to read the data, clear the offsets and control the integration time and high voltage. This package uses another one that carries out network communication with all the different crates. If the list contains profiles dispatched from several crates, the package generates automatically a sub-list for each crate.

# 6. PROPELLER CONTROL

## 6.1 Commands

As most of the profilers stop the beam, they have to be pushed in during the tuning, then withdrawn. Thus each profiler is moved by a pneumatic propeller. The control of the propellers is not carried out by the same VME crate but by other crates on the network. The display program on the stations can order them to push or pull the profilers. Depending on the station on which this program runs, the available propeller list is different. For instance it is not allowed to manipulate profilers on the beam lines from physics or experimental area workstations. Therefore propellers have been listed by clusters and the following ones are, or are not, displayed in one's menu: injector 1 or 2, beam line 1, 2 or 3, exp. rooms. These menus are constituted from a database mainly used to define the different beam optics and beam paths, and depending on the workstation name. Propellers are displayed in green if they are outside the beam, red if they are inside and yellow if their status is unknown. It is possible to select only one propeller in a list, or a lot of them, through the user interface.

## 6.2 Protection

The profile shape display doesn't automatically push the profiler into the beam. The human operator must do so, because it is quite difficult to know if the beam intensity will be too high for the equipment. If it is too high, an electronic system (chopper) can cut out part of the beam, an amount depending on the beam path and parameters.

# 7. DISPLAY SOFTWARE

## 7.1 Multi-threading

As for every ADA control program, this one is separated into three tasks (Figure 3):

- the graphic task is mainly composed of the MOTIF mainloop, with an added delay for other tasks to be able to run, and MOTIF callbacks (procedures activated by mouse clicks).
- the action task is activated by the command callbacks. As the callbacks are run during the main loop, they cannot be used to control equipment and wait for replies, because the main loop would be in a wait state and no operator's clicks could be received. So the callbacks only give synchronization points called «rendezvous» to the action task that will be in charge of the equipment commands. After testing for possible errors, this task uses MOTIF calls to display the data as numerical values.

- the refresh task is activated by only a single callback. This task performs polling on the required profiles and refreshes them on the screen. When the first profile is chosen, the corresponding callback activates this task. After that, when a new profile is required, the same callback adds its name to the polling list.

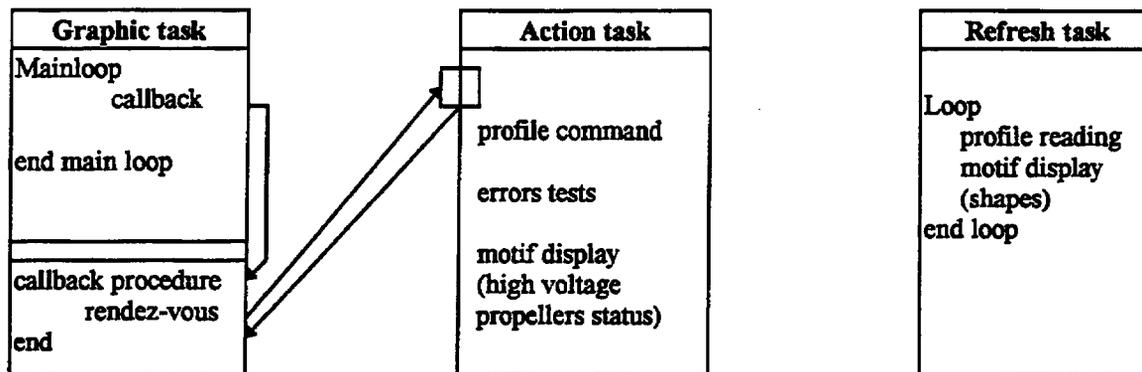


Figure 3. Console software schematic.

## 7.2 Menus

As the profilers are sometimes moved from one beam line to another, their names are not defined in the display program but in the database. The profilers are described in a tree-like structure, corresponding to the beam lines and read only at start time of the display program. The scrolled lists are built from these structures, in a recursive way and when an operator wants to watch a new profile it is available in memory without need for disk access. Other functions are also defined in these structures, such as clustering together all the profilers used for a beam line adaptation and displaying them always in the same place. Other specific profile characteristics are defined in the database, such as type of monitor, distance between the wires, gap between wires with reference to the axis of the beam, etc.

## 7.3 Commands

### 7.3.1 Scale

In case of saturation, the graph becomes orange. The operator can modify the scale by clicking on the + and - push buttons on each profile. He can also use the automatic scale option, in the top menu bar to fit the profile to the maximum of the window.

### 7.3.2 Offset

It is necessary to be able to eliminate the background noise during measurements. So, when there is no beam, the push button "offset measure" carries out a noise measurement. This measure will be subtracted from the real profiles later, and the label "offset" is displayed under the shape, to note its purpose. The button "clear offset" is useful when the beam intensity is very weak, not so different from the offset itself.

### 7.3.3 Integration time

For each profile, the current on each wire charges a capacitor, so the electronics is able to measure the voltage that will be displayed on the screens. If the current is too low, the voltage can be amplified by increasing the capacitor charge time, called the integration time. For the multiwire profiler the integration time is controlled by an external electronic system, but for the gas ionization and microchannel plates, it must be controlled separately for each profiler from the control room.

### 7.3.4 High voltage

For the gas ionization profiles, the amplification rate depends on the high voltage applied to the polarization electrode or between the microchannel plates. For these profiles a push button in the display window opens specific windows with sliders dedicated to high voltage and integration time control. This window is movable so as to see the result of the commands.

## 7.4 Graphs & Widgets

Previously oscilloscopes allowed us to see 8 profile planes, 4 horizontal and 4 vertical. The size of the workstation screen and its definition (1280x1024) are large enough to display 8 profiles, (e.g. 16 planes) and their

associated commands in eight windows. In each window a push button can allocate the profile chosen by the menu. If two planes of the same profiler have been selected they will be displayed in the same window.

A button allows to freeze in red the external shape of a profile, while its real-time representation is still refreshed. The influence of an adjustment in comparison to a reference previously defined by the operator can be seen in this way.

Each profile window is divided into two parts

- a label for the profiler's name and buttons for the commands
- a graph for the profile shape

The label and push buttons are standard MOTIF widgets but the graph is drawn with a commercial one called XRT, from the K.L. Group, a widget which is also used in other GANIL beam tuning programs.

### 7.5 Numerical data

With two button pushes (HO for horizontal plane data and VE for vertical), the graphic display can be replaced by an array of numerical values :

- center of gravity
- integration time
- high voltage
- width at the foot
- width to mid-height
- surface area

### 7.6 Archive storage

The numerical values and acquired data can be saved in the database, in order to be able to restore them later during another beam tuning session, for comparison by superposition with real-time measurements. In addition to the profiler data, other data relevant to the beam and optics are automatically saved. For restoration different sorting criteria, namely by name, date or beam identity, can be used.

### 7.7 Printing

The print button in the main window opens a new menu to print either all the eight graphs on the screen, or only one profiler (two planes) with its numerical data. The print function of the XRT widget writes in postscript format a file with the "print on close" attribute, to be directly sent to the printer at the end of this function. The numerical values are written on the same page as the XRT graph.

A home-made postscript library has been developed to be called from the ADA programs and to be compatible with the XRT functions. The initial procedure of this library tests a boolean notifying a completion of postscript by XRT and gets the name of the file created. The text is written by the postscript routines and the file is closed without print. Then an XRT function opens the same file in append mode (and not in creation mode reserved when XRT is alone), writes in the file and closes it, sending it to the printer. VMS logical names are used to dispatch the printout to different printers, depending on the workstation submitting the job.

## 8. OTHER UTILIZATIONS

### 8.1 ECR source control

A specific program has been developed for the control of ion sources. Besides control of all the source equipment, this program can display the shape from that profiler which sees the source output. The XRT widget is used exactly in the same way as in the common display program, so the operators can directly see the consequences of their actions on the ion source.

### 8.2 Tuning

An optimization program will be developed to tune the beam lines. The operator will select the adjustment point among all those possible and then the relevant three corresponding profiles will be displayed. The program will control the nearby quadrupole magnets to achieve the desired proportions between the three profiles.

## 9. REFERENCES

1. Electronique d' acquisition rapide des profils, *T. André, R. Baumgarten, E. Petit*, Ganil/575/93/ml
2. Installation des Profileurs à gaz & à galettes de microcanaux, *Groupe Aires d' expériences*, Ganil/20/94
3. Controlling the Ganil from an operator workstation, *L. David, E. Lécorché, C. Maugeais, M. Ulrich*, Conference EPAC 94 London
4. Beam profile monitors at Ganil, *E. Petit, C. Jamet*, Conference DIPAC 95 Hamburg

# INTEGRATING INDUSTRIAL CONTROL SYSTEMS INTO THE CONTROL ENVIRONMENT OF THE TECHNICAL INFRASTRUCTURE OF CERN

P. Sollander, D. Blanc and A. Swift  
CERN, Geneva, Switzerland

## Abstract

At CERN, more and more of the technical infrastructure is controlled by industrial systems, using Programmable Logic Controllers (PLCs) supplied by different manufacturers. The systems are also increasingly being installed and configured by our industrial partners. These diverse systems need to be integrated into the existing control environment, which itself is part of the accelerator control system, so that a standard environment for supervision can be provided to the Technical Control Room (TCR). Recently, several systems for electrical distribution, cooling water, safety and controlled access to accelerator radiation zones were equipped with new controls and required integration. This paper describes the solutions adopted for common interface standards, selected hardware systems and software packages and the collaboration with industrial partners. It also discusses the difficulties encountered.

## I. INTRODUCTION

The CERN Technical Control Room (TCR) is responsible for the overall surveillance and control of the technical infrastructure covering systems such as cooling, ventilation, electricity distribution and safety. The TCR is an alarm-driven control room in the sense that the arrival of an alarm will alert the operator and make him take appropriate actions. The operator acts upon the alarms primarily by consulting and interacting with mimic diagrams.

These human-computer interfaces, alarms and mimic diagrams have been structured systematically and they have been standardized to a large extent, in spite of the large diversity of installations being supervised. Any new installation must be integrated harmoniously into this environment.

Until recently, technical infrastructure equipment has been connected to the control network at the lowest level via specific CERN-developed front ends, called Equipment Control Assemblies (ECAs). For new projects, however, industrial controllers (PLCs) are being increasingly used. In the same way industrial supervision systems are being installed and must be integrated with the existing control system.

## II. CURRENT ENVIRONMENT

### A. *The Reference Database*

For a large part of the technical infrastructure, accessible equipment parameters that are known to or used by the TCR are described in a reference database (STRefDB). Every parameter or point is uniquely identified and describes the type of point (alarm, event, measurement), the geographical location of its controller and its logical location in the control system, i.e. which supervisor, which front-end computer, data block, word and bit holds the information.

This information is used to define the alarms for the central alarm server, to configure the supervisors and to create the mimic diagrams.

### B. *Equipment Front-End Computers*

Process equipment is connected to the control system via ECAs. The ECAs are CERN-developed microcomputers, usually VME-based, which are directly connected to the process equipment with digital or analog input/output or via serial interfaces such as RS 232. The ECA communicates with the level above in the control system, the Process Control Assemblies (PCAs) via MIL-1553 field-buses.

### C. *The Central Alarm Server*

The Central Alarm Server (CAS) [1] concentrates all the alarms from the equipment, constructs the alarm messages and dispatches them to the users. Every user can subscribe to the categories of alarms he is interested in. The CAS works

with one standard human-computer interface which is used in several control rooms at CERN. All new alarms must use the CAS and its user interface.

Alarms are fed to the CAS via alarm servers distributed throughout the control system. The CAS receives "V2-strings" that contain, amongst other information, the triplet FaultFamily-FaultMember-FaultCode which uniquely identifies an alarm.

#### *D. The Mimic Diagrams*

All mimic diagrams used in the TCR have been developed using a high-level graphics environment called the Uniform Man Machine Interface (UMMI) [2]. It uses the graphics package DVdraw and DVtools, on top of which is added a set of functions and languages which eases the work for the application programmer and gives a standard look-and-feel for all applications.

The applications are primarily made for the TCR operators, but they are also used by the equipment specialists who want to check their equipment remotely (e.g. from their offices).

### III. INDUSTRIAL CONTROL SYSTEMS

The control system of the technical infrastructure is a hybrid of in-house and industrial systems. This infrastructure very much resembles that of any industrial site with systems like cooling-water circuits, electric-power distribution, ventilation and safety. It is therefore possible to take advantage of products existing on the market for industrial controls to shorten development time and to ease the maintenance burden.

CERN has been using industrial PLCs for some of its technical infrastructure for several years. These PLCs are of different makes depending on the installations they control and are all connected to the CAS and the UMMIs using in-house-developed concentrating devices for their integration.

In recent years, industrial supervisory systems that include data acquisition, trending, graphical user interfaces and alarm displays have become popular at CERN. These systems provide a high-level control environment and are designed for the upper level of the control network. They are usually configuration tools rather than programming libraries. Such systems are best suited for installations of a specific type (of a limited size, a defined geographical area, etc.)

Amongst other systems, CERN has used Siemens PLC equipment and the industrial supervisor FactoryLink for some of its recent installations. It is the integration of these two products into the current environment which will be discussed in this paper.

### IV. INTEGRATION

#### *A. Reference Database*

The reference database STRefDB must now cater for the new data structures used with PLCs and FactoryLink. The procedure for entering new data into the reference database is as follows: The PLC programmer establishes a list of all equipment parameters and their location in the PLC; then this list is completed with alarm information and FactoryLink tag names and then imported into the database. We use spreadsheets as support for initial and completed data to ease data entry and to ensure correctness. The content of the spreadsheet is then converted into ASCII text and imported into the reference database using a standard SQL\*Load package.

Data must then be extracted from the database to provide the CAS with the information necessary for handling the alarms and the supervisory system with the parameters for the control of the systems. Extraction tools have been created to translate the reference database data into the appropriate formats (Fig. 1).

Data gathering and data definition are lengthy processes prone to errors. The implementation of specific procedures to rationalize this activity must be envisaged.

#### *B. Equipment Front-End Computers*

The integration at the front-end level using PLCs demands that a coherent approach towards the equipment be applied. Each type of equipment, be it fire-detection systems or cooling-water systems, has its own types of interface and data, i.e., the interface to the equipment will inevitably vary with the application. The interface from the PLCs upwards to the rest of the control network, as well as the organization of the data inside the PLC, should be dealt with in a single way for all types of equipment as far as possible. Having a number of PLCs for various installations, we have developed generic communication procedures and a standard organization for any PLC.

We use the Siemens implementation of the Profibus (DIN 19245) protocol called SINEC L2 for communication between PLCs and the Siemens ethernet (IEEE 802.3) protocol SINEC H1 for the interface between the concentrator PLC and the supervisor.

This ensures efficient network usage and allows the connection of different systems made by different firms on a single field bus communicating with one standard concentrator PLC. The procedures are integrated into PLCs programmed either in house or by an external firm.

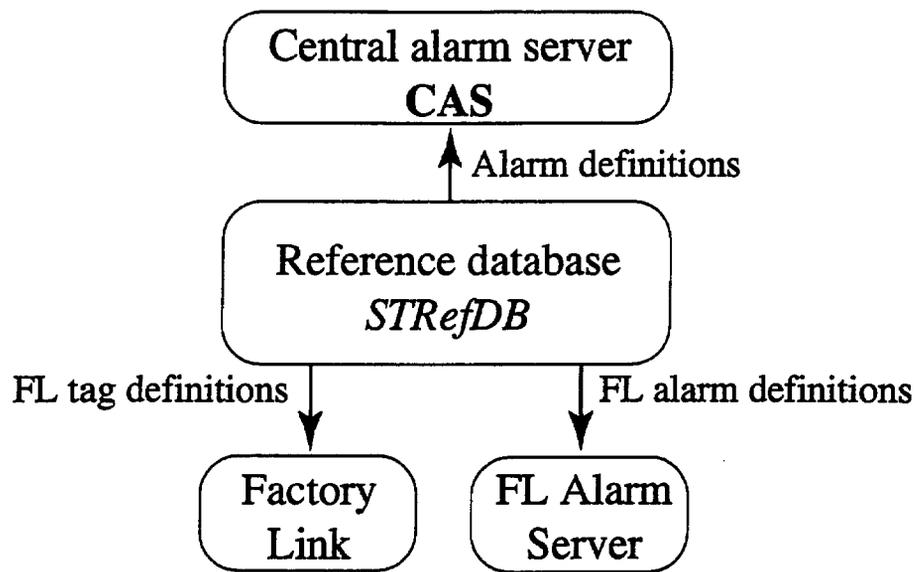


Figure. 1. Information extraction from the STRefDB

### C. Central Alarm Server

To allow alarms generated by FactoryLink to be transmitted to the CAS, an alarm module [3] was added to FactoryLink. This module is fed a list of strings associating a FactoryLink tag with a CAS V2-string where each tag corresponds to one specific alarm. The alarm module detects any change of state in one of the tags and generates the corresponding alarm by sending the V2-string to the CAS.

### D. Mimic Diagrams

The use of FactoryLink does not imply the abandonment of our UMMI environment. As the UMMI is our standard graphical user interface, it must be possible to use it with data coming from any source. On the other hand, for some specific applications we do use FactoryLink graphics which must be granted access to existing ECA equipment as well as to PLCs.

We have added two modules to FactoryLink which conform to the communication standard Equip [4] built at CERN and based on Remote Procedure Calls (RPC). It allows transparent data access to compliant equipment or computers by giving the equipment description. An Equip transaction is made between a client, generally in a console of the control room, and a server implemented on the equipment. Control room clients may now access an industrial process via FactoryLink as if it was home-made equipment using an Equip server [3]; FactoryLink may access home-made equipment via an Equip client.

### E. Architecture

Figures 2 and 3 show the hardware and software system architectures after integration.

## V. APPLICATIONS

Several systems have now been integrated into the supervisor. One supervisor covers the control for the entire demineralized water system for the Proton Synchrotron (PS) accelerator and another deals with the safety alarms for the Meyrin site and the demineralized water for the West Zone experimental area of the Super Proton Synchrotron (SPS). It is furthermore envisaged to extend the system to include the heating plants of the Meyrin site and a large part of the CERN ventilation systems.

### A. Safety Applications

The first application that was developed using the above architecture was the safety system for the Chorus [5] and Nomad [6] experiments. The system is controlling fire, gas detection and emergency-stop equipment in one experimental hall. The alarms generated are distributed to the TCR, the CERN fire brigade and to both experiments via the CAS and its user interface.

As this system controls personnel safety equipment, the alarm transmission to the fire brigade is made in parallel with a hardwired synoptic display located in the fire station.

As a follow up to this project we are currently installing new fire detection equipment on the Meyrin site in the same way. All new equipment is now ordered with a SINEC L2 interface for smooth and standard integration.

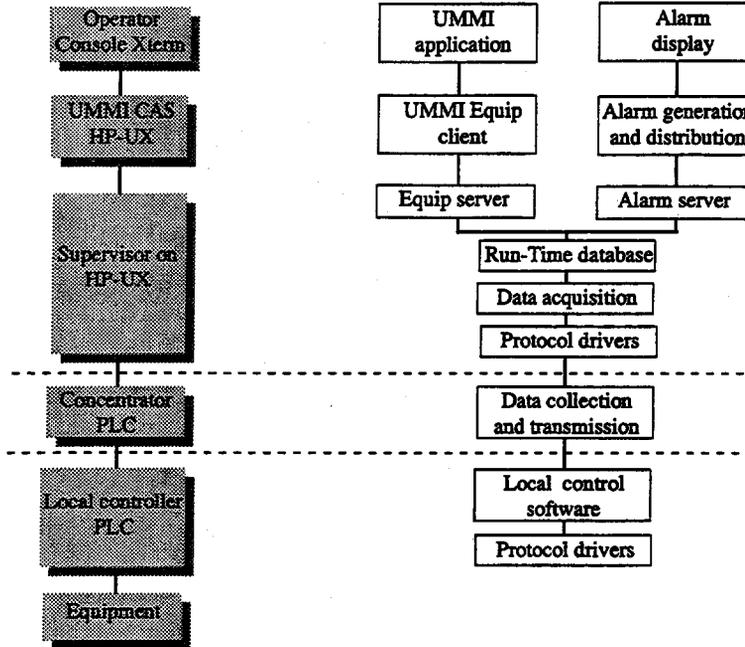


Figure. 2. Hardware architecture

Figure.3: Software architecture

### B. Cooling Water in the West Zone

The second system to be integrated in the new control architecture was a cooling station for the West Zone experimental areas. The process control was developed by an external firm using two PLCs. The field bus used for the previous Chorus and Nomad safety system was extended to include also these two PLCs. The communication functions for the PLCs were provided by CERN to ensure compatibility.

### C. Cooling Circuits for the Proton Synchrotron Accelerator Complex

The cooling water control systems for the Proton Synchrotron (PS) were entirely rebuilt in the period 1994–95. The project was given to an external firm who was asked to deliver a SINEC H1 connection to an HP-UX machine running FactoryLink.

The complex comprises 15 cooling stations controlled by PLCs connected to two separate field buses by SINEC L2 to a concentrator PLC which is the connection point with FactoryLink installed on the HP-UX machine.

Owing to the large number of equipment parameters controlled and the relative stability of the system, an event-driven data acquisition system was implemented. A change of state of any parameter of an installation will trigger the sending of the information upwards in the control chain. To ensure the integrity of the supervisor's database, status information is also sent regularly at 30-second intervals.

### D. Access Control

The PS and the SPS machines have recently been equipped with new access control systems. In both cases an external firm was asked to deliver a turnkey product using the CERN standards.

In these two applications, which are operated only from specific consoles, FactoryLink was used for both data acquisition and mimic diagrams, and since there is no alarm handling through the CAS there is no need to hold the data on the STRefDB.

### *E. Electricity Distribution*

For reasons of electrical protection and in order to comply with safety standards, the approach of the electrical group is somewhat different, since we have to buy specific equipment from agreed manufacturers. However, since this equipment broadly resembles the more general purpose PLCs used by other groups, in that it communicates via a serial protocol, the problems of integration of this equipment into the control network are identical. The message structures contained in this protocol are equipment-specific, which is why we have chosen to insert a data-concentrator/decoder (called a Bus-Manager) between the equipment and the control system, in order to transform all equipment data into a single standard format. At the supervisory level we are at present studying the possibility of integrating one of the many supervisors available on the market, dedicated to electrical supervision, into the proposed Technical Data Server (TDS) [7].

## VI. CONCLUSION AND OUTLOOK

The different projects that integrate lower level industrial control equipment and higher level industrial supervisor packages so far have shown us the major advantages and disadvantages of both the products and the policy of outsourcing development work.

For the industrial supervisor FactoryLink, it can be said that it is not generally the best solution to our problems due mainly to the fact that we cannot make use of all of its features. We are limiting ourselves to the data acquisition and the real-time database, not using its graphics, alarm handling or trending. On the other hand, for applications like the access control systems, FactoryLink seems to be adequate.

For the users, the new control architecture has meant shorter response times due to the fact that we now acquire data from a powerful machine situated high up in the control network instead of acquiring data directly from the equipment. The time it takes to update a mimic diagram has gone down from approximately 15 to less than one second. The fact that the information read is a few seconds old does not matter in the operation of this kind of equipment. This is, however, not true for the access control applications, in which the system is carrying data from a person wanting to access the accelerator to an operator and rapid transmission of information is very important.

The difficulty of integrating industrial equipment has been found not so much in the higher levels concerning the alarms and mimic diagrams as in the lower levels concerned with integration of the actual equipment to the industrial PLCs. In effect, systems like fire detection are not designed to be a part of a larger control system but are normally stand-alone systems with their dedicated user interfaces and alarm screens. A solution to this is to order the low-level integration directly with the system, specifying the communication protocol desired.

At present we are trying a different product for the integration of industrial controls. We plan to use the distributed control system RTworks for some systems. This product also has the advantage of directly integrating with our existing mimic diagrams made with DVdraw.

### References

- [1] M. Tyrrell, "The LEP Alarm System", in Proc. ICALEPCS'91, pp. 254-259.
- [2] P. Ninin and P. Sollander, "Utilisation d'un logiciel commercial graphique pour le contrôle des installations techniques du CERN", CERN/ST/MC/TCR/92, presented at EERIE, Nîmes, France, October 1992.
- [3] J. Brahy, "Une passerelle entre le monde des automates programmables industrielles et le contrôle des accélérateurs", AT 93-05 (IC) (private communication).
- [4] P. Charrue, "Accessing equipment in the SPS-LEP controls infrastructure: The SL-EQUIP package", SL/Note 93-86 (CO) (private communication).
- [5] <http://choruswww.cern.ch/welcome.html>
- [6] <http://nomadinfo.cern.ch/>
- [7] P. Ninin et al., "The Technical Data Server for the Control of 100 000 Points of the Technical Infrastructure at CERN", these Proceedings

# DATA LOGGING FOR TECHNICAL SERVICES AT CERN

R. Martini, H. Laeger, P. Ninin, E. Pfirsch and P. Sollander  
CERN, Geneva, Switzerland

## Abstract

A data-logging system for the technical infrastructure of CERN was developed and has been used for over two years by the Technical Control Room (TCR) and by equipment groups. Hardware and some software are common to the SPS/LEP accelerator control system, though some particular features had to be implemented to meet specific requirements of the TCR. One of these requirements is the need for data-logging throughout the year without any interruption. Currently, around one thousand measurements are logged at rates ranging from one to fifteen minutes; data are kept online for periods varying between one month and one year, after which they are automatically archived. This paper gives a description of the data-logging system, including the database structure, the interface tools for data management and data display, the data-acquisition mechanism, and the integration of the system in the existing control environment. An account is given of the experience gained and the problems encountered as well as the evolution of the system.

## I. INTRODUCTION

A complex infrastructure of basic technical services, comparable to the industrial services of a medium-sized town, is required to run the accelerators as well as all other activities at CERN. These services include electricity supply, water supply, heating and ventilation, and personal safety systems such as fire and gas detection. To ensure the smooth running of these services the Technical Control Room (TCR) monitors the state of this infrastructure 24 hours a day and 365 days a year, and takes corrective actions where needed.

For many years a number of measurements, mainly electrical and fluid data, had been recorded on paper. The recording devices were expensive to maintain, and could be non-operational due to trivial causes such as lack of paper or ink. Logged data was becoming increasingly necessary to analyse the behaviour of equipment prior to malfunction, to help in optimizing performance, or to reduce cost, e.g. energy consumption. In order to respond to a growing number of measurements that required logging, a computer-based system was envisaged which could be linked to the existing control system.

The start of this project coincided with a project in the SL control group to log a large number of measurements relevant to the performance of the LEP accelerator. A collaboration was undertaken to devise a solution that would share as many resources as possible [1].

## II. REQUIREMENTS, CONSTRAINTS, DESIGN CRITEREA

The TCR required the following basic functionality from the data-logging system:

- different measurements should be logged at different frequencies (1 to 30 minutes)
- data for different measurements should be kept online for different periods (days to months)
- data should be archived
- logging parameters should be changeable at any time by non-specialists
- adding new measurements should be possible by non-specialists
- logged data should be available for display in graphical or in raw form
- summary data (averages, maximum and minimum values within a period of time) should be kept if required
- data should be logged without interruption
- problems should be reported to TCR operators for remedial action.

### A. Design Constraints

The design of the product was heavily influenced by the LEP logging system [1], which is an ORACLE-based solution, implemented for storing key measurements for LEP operations. The table structure defined for this project was used in the TCR logging system.

The measurements to be logged are all on the control network [2], which consists of Equipment Control Assemblies (ECAs) connecting the equipment, linked to Process Control Assemblies (PCAs). Data is accessed differently depending on the type of equipment being treated. The standard communication modules, which allow users at a workstation to obtain data from the equipment, were used in the data logging application.

The TCR has a policy to hold all elements of the surveyed equipment under its responsibility in an ORACLE reference database (known as STRefDB), so that control system applications can be data-driven. The data logging application conforms to this policy.