A Table Driven Database and Applications Generator for Accelerator Control Systems

Anthony Carter , Coles Sibley and Tomás Russ
MIT - Bates Accelerator Center
21 Manning Rd., Middleton, MA. 01949

Abstract

Accelerators and related hardware are dynamic entities. Similarly, the control systems that are designed for them must be dynamic. This paper describes our attempt to define a fundamental part of the control system for a pulse stretcher ring that adapts easily to the changes inherent in a system of this complexity.

I. INTRODUCTION

The Bates Pulse Stretcher Ring [1] control system architecture is a distributed system of computers distinguished as either data processing or data acquisition/control [2]. The data processing computers serve as either display stations or control stations. The display stations are low-cost workstations or personal computers that provide continuous update of the status of pertinent Ring data. The control stations are high performance workstations that give the operator real time control of Ring elements. Also, there are various other computers that serve specific functions such as Ring modeling and orbit correction. The data acquisition/control computers acquire raw data from the Ring hardware at periodic intervals (roughly 4 Hz) and multicast the data on the Ring Ethernet. They also listen for requests from the data processing computer and perform the necessary control functions.

II. GOAL

The common aspect of all these applications is that they all need certain system-wide information in order to function properly. This information could be constants (which are not necessarily very constant), hardware address information, scaling information, or any information that is not application specific. Typically, this information is hardcoded into an application making it very efficient but very hard to maintain. Our goal was to develop a system whereby an application could be written to query a central database for this class of information (see figure 1).

Writing an application to incorporate system-wide constants is a relatively simple operation. Once the information has been retrieved, the application should proceed with no significant overhead. However, designing an application where scaling information is incorporated at run-time proved to be a much more serious task.

III. DESIGN

Our first attempt at designing a system where the scaling of raw data could be defined in a database was to have a third-order (or higher) polynomial describe the transformation. The database would contain the coefficients of the polynomial and the raw data would be the independent of the polynomial. The computing of the polynomial could be hard coded into an application preserving much of the efficiency. Unfortunately, this proved to be not flexible enough to describe the data in our system.

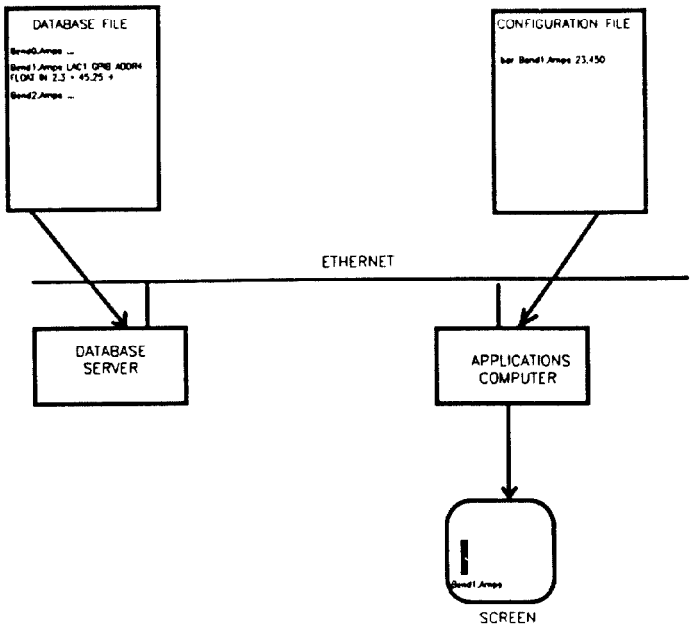

Fig. 1 - Server / application architecture.

0-7803-0135-8/91$01.00 ©IEEE

Table 1.

Operators	Description
+ - * / %	simple math
! && || > >= <= == !=	boolean
sin, cos, pow, log, ln, sqrt	complex math
sto	memory store
rcl	memory recall
in	input
out	output

Our next step was to throw simplicity out the window and look for a system that could describe virtually any transformation. We settled on a system where the transformation was described by a postfix equation (also known as Reverse Polish Notation or RPN). The supported operators and functions are taken from the C programming language where possible (see table I). A few functions were added to provide for special operations.

The IN function was added to describe how to extract raw device data from the multicast Ethernet packets. It takes four arguments: source computer, source port, offset into the packet and the data type of the raw value. The data type can be any of the following: char, unsigned char, short, unsigned short, long, unsigned long, float or double.

The OUT function defines how to send a request to a data acquisition/control computer. It takes the same four arguments as the IN function, adding a fifth for a command. Each data acquisition/control computer supports a set of commands, such as READ, WRITE, INCREMENT, RAMP, CYCLE, and so on.

In addition the STO and RCL functions provide the memory store and recall capabilities of a calculator. These are used in a special way. When you convert a raw signal to some engineering unit, the IN function effectively holds the place of that raw data in the equation. However, when you want to reverse the process, so that you can SET something for instance, the data you want to convert comes from within the application program. The way we handle this is as follows: the equations for setting things are written with RCL's holding the place of the data. The application programs are written to STO to memory the data to be output before computing the equation. As the equation is computed, the actual data will be inserted anywhere a RCL is found.

IV. IMPLEMENTATION

In our implementation of this system, we have one machine that acts as a central database server. Configuration information is broken into two distinct groups, application dependent and application independent. The former would be things like where on a particular screen a bar graph is placed, and the latter would be things like limits on a particular power supply. Only application independent information is kept in the central database. All programs are written to query the server for this data at initialization time.

For example, one display computer program reads a configuration file with a format like:

BAR CB1.ADC.GAUSS x1 y1 x2 y2

This tells it to put a bar graph from coordinates x1 y1 to x2 y2, computing the data with the algorithm known as CB1.ADC.GAUSS. The equation for converting amps to gauss for that power supply might be

$$B_0 + B_1X + B_3X^3 + B_7X^7 + B_{11}X^{11} \quad (1)$$

where B0, B1, B3, B7 and B11 are constants derived from magnet mapping and X is the signal in normalized amps. This equation would be entered into the central database. Figure 2 shows what would be a typical excerpt from the database, with some names altered for clarity.

At initialization time, the applications read their own configuration files, which specify which signals are needed. The server is queried for the definition of each signal and the equations are parsed and stored in an internal form for processing. How the equations are parsed and stored is application dependent, but a standard parse/compute engine has been written and should work well for most applications.

Since configuration information is only read at initialization time, the overhead of querying the server is insignificant. The overhead of repeatedly computing each of the equations is not. However, given the ever increasing computational power of today's workstations, and some intelligent software design, this should not be a problem.

V. CONCLUSIONS

This system of describing the Ring data should make it easy for the user to do high level application development by hiding details behind logical naming schemes. Also, it will guarantee system-wide configuration data integrity by keeping it defined in one place, in the central database. The decrease in dependence on skilled computer personnel to maintain this information is well worth the added cost to the computer hardware required to support it.

Signal Name	Definition
LAC8	8
IOPORT	1
USHORT	2
B0	2.83236E-2
B1	7.498663
B3	-2.3996E-2
B7	1.00004E-2
B11	9.9411E-3
CQ1.AMPS.MAX	600.00
CQ1.ADC.RAW	LAC8 IOPORT 0x10 USHORT IN
CQ1.ADC.AMPS	CQ1.ADC.RAW CQ1.AMPS.MAX * 65535 /
X	CQ1.ADC.AMPS 150.0 /
CQ1.ADC.GAUSS	X 11 ^ B11 * X 7 ^ B7 * X 3 ^ B3 * X B1 * B0 + + + +

Fig. 2 - Sample database excerpt

VI. REFERENCES

[1] J. Flanz et al. "The MIT- Bates South Hall Ring", Proc. of the 1989 Particle Accelerator Conf., Chicago, Mar. 1989, pp.34

[2] T. Russ et al. " The Bates Pulse Stretcher Ring Control System Design", Proc. of the 1989 Particle Accelerator Conf., Chicago, Mar. 1989, pp.85.