DESIGN OF THE COMMISSIONING SOFTWARE FOR THE AGS TO RHIC TRANSFER LINE*

C. G. Trahern, C. Saltmarsh†, T. Satogata, J. Kewisch,
S. Sathe, T. D' ottavio, S. Tepikian, D. Shea
Brookhaven National Laboratory, P.O. Box 5000
Upton, New York 11973

Abstract

RHIC accelerator physicists and engineers have collaboratively specified the control system software for the commissioning of the AGS to RHIC transfer line (ATR) to occur in the fall of 1995. This paper summarizes the design and progress to date. We discuss the basic physics/engineering device model that we use to understand process and data flows, and describe the architecture and tools we will use to build the application level software.

I. INTRODUCTION

The AGS to RHIC transfer line will be commissioned in the fall of 1995. In preparation for the commissioning of RHIC in 1999, prototypes for some parts of the RHIC control system are being prepared to commission the transfer line.

The basic assumptions in our design effort are 1) the use of object oriented (OO) coding throughout, both at the embedded systems/hardware level as well as in physics level applications, and 2) a sequencing and interprocess communication protocol implemented using a program known as *glish* [1]. Given these assumptions, the design of the control system has proceeded in two stages. In the first stage we took the list of requirements for the commissioning test and made a general analysis of the data and process management needed to meet these goals. In the second stage we have prioritized the requirements and organized the work of the accelerator physics and control groups to implement the relevant systems.

II. ACCELERATOR PARAMETER OBJECTS

High-level requirements for ATR commissioning range from the ability to control currents on individual power supplies to adjusting steering corrector elements in concert to produce the desired orbit deviation as measured by an assembly of beam position monitors (BPMs). In order to conceptualize the entire range of applications, a set of data models was created. For each accelerator parameter we define an object which contains the following data:

Parameters (e.g., tune, chromaticity, magnet current, etc.) are the set points that describe the desired state of the accelerator. Parameters are functions of time. The parameter value at any time is obtained by specifying the values at *step-stones* and using an appropriate interpolation method. In most cases the step-stones will coincide with the events on the accelerator event line. The control system can modify future step-stones, while present and past step-stones are not changeable.

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

All quantities that prescribe the state of the accelerator are parameters. The parameters are therefore redundant. A system of processes ensures that the parameters are consistent. This allows the operator to describe his goal directly. He can set any parameter and leave the task of finding the corresponding hardware settings to the control system.

Measurements are the actual readings from the machine hardware. (For most parameters a direct measurement is not available.) They are independent of the parameters with the same name. Although some hardware devices include the setting and measurement of an accelerator parameter in one module, this is the exception.

Trims are changes to the accelerator parameters that enforce desired behavior. Trims are only used in objects where a measurement is available. Ideally, trims make the value of the parameter and the measurement the same.

Method data are data that determine how to convert dependent parameters. Methods often change during operations. For example, given the desired orbit the method data describes which correction dipoles are used to move the orbit.

Accelerator knowledge is configuration information about the accelerator. This data changes only when the machine or the control system is reconfigured. Examples are lattice information, host names, power supply names and magnet data.

In order to describe the idea we resort to the methodology of data flow diagrams[2]. The general picture is shown in Figure 1 and a more complete discussion of this model can be found in [3]. In the figure processes are defined which perform the following data transformations:

1. Calculation of all dependent parameters, if the new parameter is changed.
2. Update of the new parameter if a parameter on which the new parameter depends changes.
3. Measurement of the parameter, if possible, and generation of an alarm if outside limits.
4. Prediction of trims.

A set-request from the operator or sequencer or a higher level object is first checked for semantics and for range in the "validate and sequence" process. Method data is used for this check.

If the request is valid, the parameter is set in the parameter data store. The "validate and sequence" process administers the step-stones in the parameter store. The parameter is passed on to the "calculate lower parameter" process. Method data is used to determine which lower level parameters are used to do the change. The process may also use other parameters and accelerator knowledge for the calculation.

The calculated parameters are sent to appropriate object(s) which internally have the same structure as this object: the de-

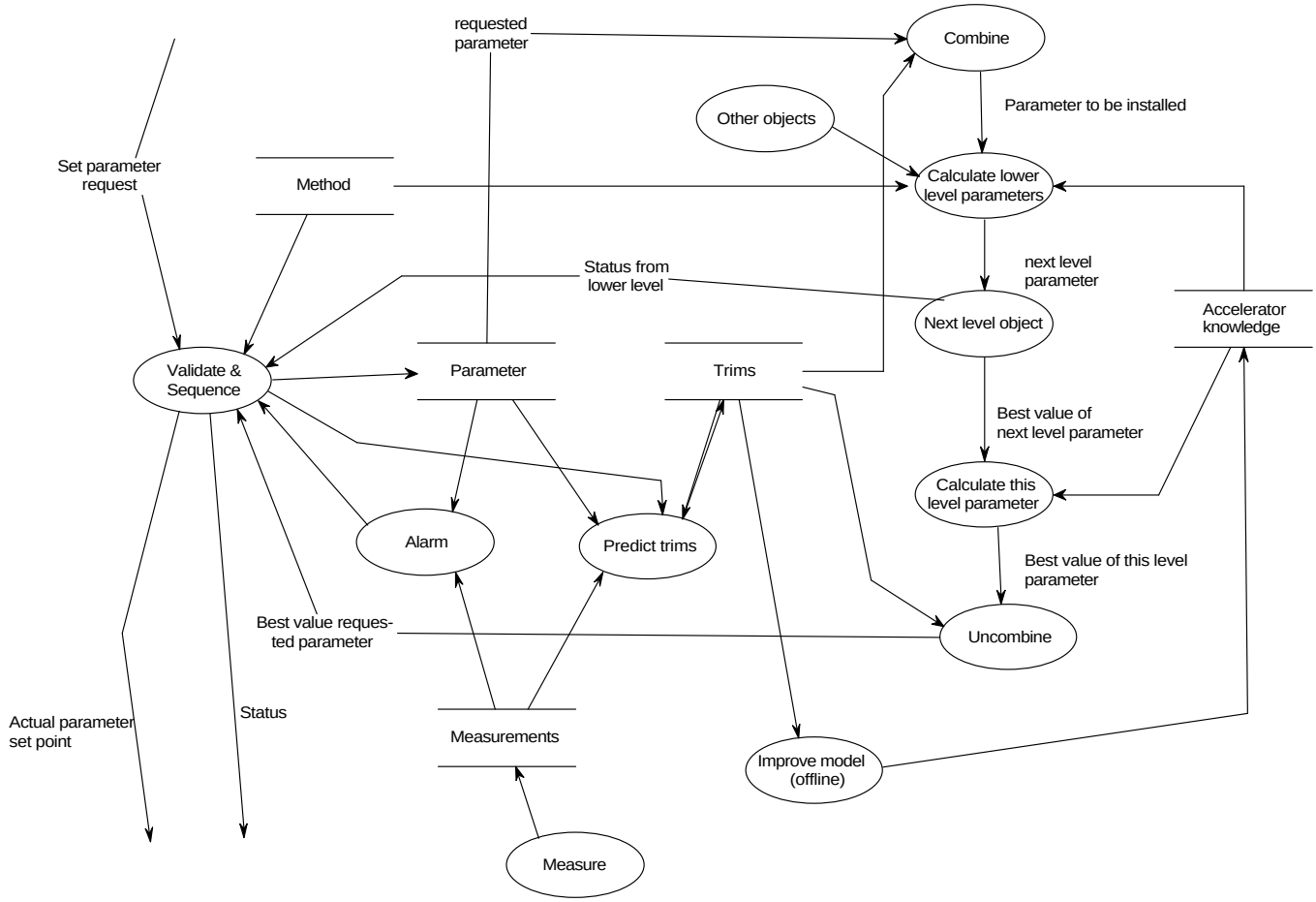

Figure 1. Data Flow Diagram for Parameter Object

sign is recursive. The lower level object returns the set parameter or, in the case of error, an error message and the best achievable parameter value which is passed to the validate and sequence process.

The validate and sequence process checks the success of the set operation and fields asynchronous errors. It updates the parameter value in the data store and returns the value or best value to the next higher level.

The "model" of the accelerator resides in the "calculate lower level parameter" and "calculate this level parameter" processes. It is important that these modeling processes occur on each level and are by design independent from each other. This allows implementation of new parameters and extensions of the model in a flexible way. In implementation the same program or process might be used at different levels. For example, an optics calculation process might be set up as a "persistent" service for different parameter objects.

The accelerator knowledge data store provides the basic information for the model. It contains lattice information, magnet data, etc. Although this data changes rarely, it is a part of the data flow.

The recursive design suggests that parameters can be organized as a tree, where each object is a node with all internally used objects as leaves. However, it turns out that there is no natural structure for this tree. Whatever parameter is set by the operator or sequencer is the top level. The tree structure must therefore be dynamically configured for each operator or sequencer command.

The system described so far allows the generation of a consistent set of redundant parameters. If the model used is correct, the parameters give a complete description of the machine. Unfortunately, real life is often different, and the measurements of parameters differ from their desired values. The parameter object therefore contains measure and trim processes.

In addition to the parameter store the parameter object contains a trim store. The goal of operations is to predict the best trim, so that the machine behaves as described by the parameter. The trim is predicted on the basis of past experience. A "Predict Trim" process calculates the trim for the future step-stone from the measurement of the past or present and the parameters and trims that lead to those measurements. The "combine" process combines the parameter value and the trim value into a "parameter to be installed" value. (In most cases the combination means just adding the two values, but more complicated combinations are possible.) Instead of the desired (requested) parameter, this "parameter to be installed" is passed to the "calculate lower parameter" process. The returned values from the dependent parameter objects need to be "un-combined" from the trims before they are sent to the validate and sequence process for interpretation.

III. IMPLEMENTATION

C++ will be used for the programs that control the front end electronics, global beam-level applications and monitoring systems. Consequently, complex data structures can be developed with all the useful inheritance features of the language. Configuration data for all of these structures will be organized using both relational database management systems and UNIX file systems[4]. The transport of these data structures from one independent process to another will be managed by a program context known as *glish*. Client processes interconnected in a *glish* environment need no knowledge of other processes or their data structures. Events, defined as a name-value pair, are directed to various clients by the *glish* interpreter. When these named events occur in some process, the interpreter recognizes the event and passes the value data onto appropriate clients. This environment enables significant modularization of code as well as the distribution of processes over an extended network.

The architecture for the control system implied by the parameter object design would allow a hierarchy of software systems communicating with each other without a "hard" line between the front end and global beam-level control systems. For example, in principle the orbit correction algorithm could address hardware by passing the appropriate data structure via the *glish* interpreter directly to the front end computers. However, this functionality of *glish* will not be used during the commissioning test. A protocol/class library known as the Accelerator Device Object Interface or ADOIF will mediate between the global control systems and the front end modules. The front end electronics is controlled by VME based computers using the VxWorks[5] operating system. The conceptual design for the front end systems has been described elsewhere[6]. A *glish* client built around ADOIF will receive a *glish* event and forward it to the appropriate front end. Information from the front ends will pass back to the global systems analogously. Depending on the need to cache the readback information from hardware for use by various processes, device managers, also *glish* clients, will sit between the ADOIF *glish* client and the beam-level applications. These manager processes would be responsible for absorbing the data from front end computers and making it available to generic *glish* clients.

All processes are designed to be independent of any graphical user interface as a data source. Whether the local data of a process is output to the screen or not, the process will be addressable via the *glish* event sequencer. Generic device input follows the same principle. There will be button/menu driven parameter pages available though a graphical interface, but the same parameters will also be adjustable via the event interpreter.

IV. CONCLUSIONS

The commissioning software for the AGS to RHIC transfer line will be a model for the RHIC control system. We will test the ideas and principles outlined here in September 1995. After the test a thorough review of both the systems and design will be carried out. We are committed to "throwing away" code as well as a re-conceptualization of the system design in order to avoid future problems. The construction of the control system requires the close collaboration of both the accelerator physics and controls groups. The outlook of the physics group tends to focus on the beam-level systems and that of the control group at the level of the front end computers. Making the transition between the two domains a smooth one is not a trivial task. We believe that the experience gained from the upcoming test will provide enough information to make whatever changes are needed for RHIC commissioning.

References

[1] Vern Paxson, "The *Glish* 2.4 User's Manual", RHIC AP note 30.

[2] T. DeMarco, *Structured analysis and system specification*, Englewood Cliffs, New Jersey, Prentice Hall, 1979.

[3] T. D'Ottavio, et. al., "ATR Commissioning Software Task Force Report", RHIC AP note 53, 1994.

[4] C. G. Trahern, et. al. "Relational Databases for RHIC Design and Control", EPAC proceedings, 1994.

[5] VxWorks, Wind River Systems, Inc., 1010 Atlantic Ave., Alameda, California 94501-1147.

[6] L. T. Hoff, J. F. Skelly, "Accelerator Devices at Persistent Software Objects", Nuclear Instruments and Methods in Physics Research A, 352 (1994) 185-188.

† Current address: Flat 4, 89 Belvedere Road, London SE19 2HX England, email: salty@crapeau.demon.co.uk