THE AUTOMATOR : INTELLIGENT CONTROL SYSTEM MONITORING*

M. Bickley, D. A. Bryan#, K. S. White,
Thomas Jefferson National Accelerator Facility, Newport News, VA

Abstract

A large scale control system may contain several hundred thousand control points which must be monitored to ensure smooth operation. Knowledge of the current state of such a system is often implicit in the values of these points and operators must be cognizant of the state while making decisions. Repetitive operations requiring human intervention lead to fatigue, which can in turn lead to mistakes. The authors propose a tool called the Automator based on a middleware software server. This tool would provide a user configurable engine for monitoring control points. Based on the status of these control points, a specified action could be taken. The action could range from setting another control point, to triggering an alarm, to running an executable. Often the data presented by a system is meaningless without context information from other channels. Such a tool could be configured to present interpreted information based on values of other channels. Additionally, this tool could translate numerous values in a non friendly form (such as numbers, bits, or return codes) into meaningful strings of information. Multiple instances of this server could be run, allowing individuals or groups to configure their own Automators. The configuration of the tool will be file based. In the future these files could be generated by graphical design tools, allowing for rapid development of new configurations. In addition the server will be able to explicitly maintain information about the state of the control system. This state information can be used in decision making processes and shared with other applications. A conceptual frame work and software design for the tool are presented.

1 INTRODUCTION

Distributed and networked control systems have become very common for use in the control of large scale experimental systems, such as particle accelerators or telescopes, as well as for industrial control systems. As the scale of these control systems has increased, the number of parameters for controlling the system has increased as well. A large modern control system may have as many as a quarter million control parameters.

With such a large number of control parameters, determining the overall status of the control system or of large subsystems has become difficult. This high-level view can be thought of as a "meta-parameter", a value which implicitly contains information obtained from many individual signals.

Users often must view tens or even hundreds of signals and infer the state of the machine from these values. Archiving information about these "meta-parameters" is difficult since one needs to archive the many individual signals and later post process the data to obtain these meta-parameters. This increases the volume of information that must be archived. Clearly it would be better to be able to dynamically monitor the individual values and create this "meta-parameter" as a single control system variable.

2 PRACTICAL CONSIDERATIONS

There are many ways to implement these meta-parameters, but some considerations could make them more useful. At an existing facility, there is most likely a large collection of tools in place to deal with existing control system parameters. These tools would include viewers, archivers, dynamic data analysis packages, etc. The tools are usually well tested and the operations crew is familiar with their use. This situation makes it desirable for the new meta-parameters be available to the control system using the same protocol as the existing signals. This approach allows for maximum code reuse.

Most facilities control hardware with a collection of front-end machines which monitor the hardware and make information about the hardware available to the control system as signals. The tools mentioned above run on back-end machines, often UNIX or PC based.

One approach to solving this problem would be to simply create a new signal on the front end server that contains the meta-parameters. This could be implemented via communication between front-end machines. In practice, this presents a number of problems. First, the front-end machines are usually responsible for critical operation of the hardware. As such, increasing the workload of these machines or modifying tested software is often undesirable. In addition, the meta-parameters that one is interested in may change frequently, perhaps even while the system is running. If these variables are placed directly on the front-end machines, modifying them could interfere with the operation of the control system.

Similarly, placing the processing of these meta-parameters in each client has drawbacks. Each client must be modified in order to use the new variables, and

* This work was supported by U.S. D.O.E. contract #DE-AC05-84ER40150

Email: bryan@jlab.org

each must keep a separate copy of the logic needed to infer the meta-parameter from existing variables.

3 IMPLEMENTATION AS MIDDLEWARE

The authors have decided to implement the desired functionality using a middleware server [1]. Such a server is a piece of software which monitors signals from some source, in this case the front-end machines. These servers then produce new signals for the destination, such as the back-end machines. One can then treat these signals in the same way as the original signals. These new channels are often called virtual signals.

At the Thomas Jefferson National Accelerator Facility (JLab), there has been considerable positive experience with using such servers for various applications. Each server has been written individually as a separate program. The authors' proposal is for a general purpose, user configurable program for creation of these special meta-parameters.

JLab uses CDEV, or Common DEVice, to communicate with the underlying EPICS control system. CDEV provides a generic server framework for allowing users to write such middleware servers. The Automator will be based on this CDEV Generic Server. [2,3,4]

4 DESIGN AND INTERFACE

The program provides the frame work to define a server for creation of new signals or to react to these new values. Users will configure the Automator by generating a configuration file. The configuration file contains information on the name of the server instance to be created and information about the interrelation of modules. These modules can perform a number of tasks. They can either monitor a variable in the system or produce a new signal to be used by client programs. They can process incoming signals. Finally, they can take an action such as executing a script. These modules are connected together to define the structure of an instance of an Automator, as shown in figure 1.

The user creates modules to monitor the desired signals and connects these to gates. Initially, the gates available will be simple, logic-based gates such as noting if a monitored value is within certain limits, logical "AND" gates, logical "OR" gates, and similarly simple structures. The user then connects the outputs of these gates to either a new signal to be monitored, an object which modifies a value in the control system or something which executes an action. By allowing for an arbitrary script to be executed, the tool can be used to automatically respond to problems in the control system.

Initially, the user will define the structure of an Automator instance by directly editing the configuration file. Eventually this process will be automated to allow a schematic capture tool to be used to create these files graphically.

The Automator will be implemented using C++ in such a way that the gates can be easily defined by users. The gates will be implemented as C++ objects. The user can create new ones by inheriting from the base object and defining the operation of the new gate. This will allow for more complicated, "intelligent" gates to be developed to handle site specific concerns. It is hoped that as the Automator is used at more sites, developers will share these modules, promoting software sharing and reducing development time for developers.

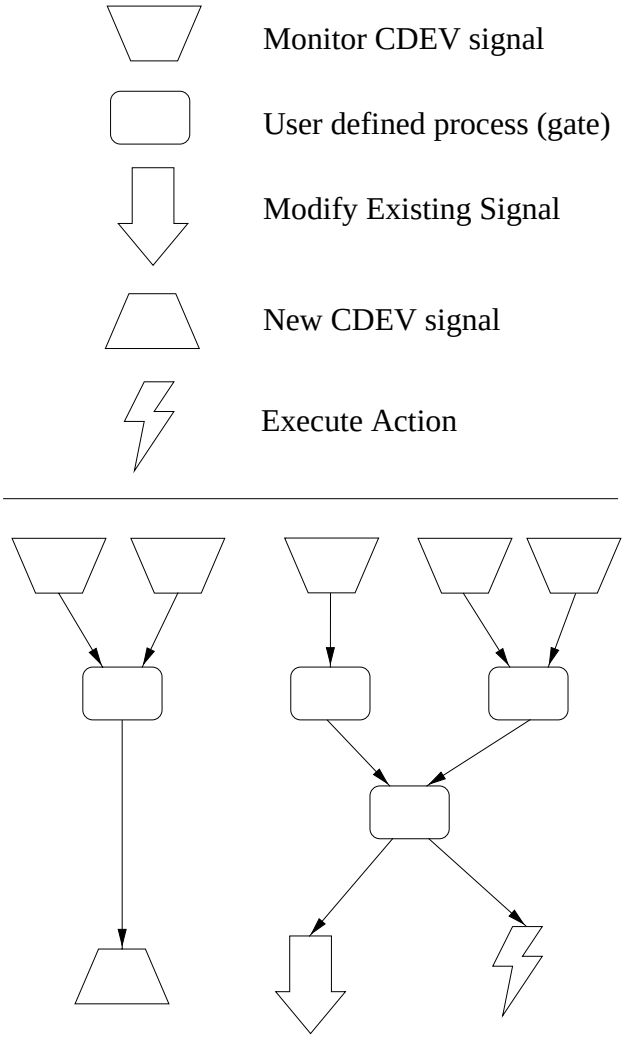

Figure 1 : Automator Configuration Layout

5 ADVANTAGES

5.1 Operational Advantages

At a large facility, there are typically many simple but repetitive tasks that operators must perform manually. The tasks never seem to be pressing enough to warrant the development of a software tool to respond automatically. The Automator will allow rapid development of such responders.

Similarly, short term procedure changes and temporary workarounds can present problems that interfere with operations. These special instructions must be given to all personnel operating the machine and they must know the new limits or procedures to follow for a particular device. JLab's main accelerator is run twenty-four hours a day by three crews working on alternating seven day shifts. By automating such processing, the chance of instructions getting lost or distorted during crew changes is reduced.

Finally, such a tool allows software developers to capture some of the knowledge of a trained operator or technical specialist. If the operator's response to a certain condition is quantifiable it can be translated into an instance of the Automator. This response is then available even when the skilled operator is not.

5.2 Development Advantages

In addition to advantages to operators and all of the development advantages discussed above, the Automator provides a few more advantages. By controlling when and under what conditions certain tools are invoked, the developer can build complex tools from collections of small, tested, known good applications. The developer can also use the Automator to prototype servers (if they can be easily described by a collection of gates), and to monitor the control system for transient behavior that may lead to problems.

This design has several advantages over other tools designed to provide similar functionality. SDDS is a collection of scripts designed to be used together by piping the output of one script to another, providing some of the processing abilities of the Automator [5]. Because these scripts post-process data stored in SDDS files, they cannot provide the processed data to users dynamically. The Automator is a compiled program rather than a sequence of scripts connected using UNIX I/O, which should allow faster execution time. While there is some overlap the Automator is generally designed to provide simple but dynamic processing and response, while SDDS is intended to provide more complex offline processing.

Some of the automated response functionality can be provided by EPICS sequencer programs [6]. The fundamental difference between a sequencer instance and an Automator instance is the level in the control system where the response is taken. By running the Automator as a UNIX process access to common scripting languages for invoking responses is simplified. Additionally, changing a running instance of the Automator would be straightforward and would not interfere with the operation of the control system as modifying a sequence on a front end machine would. If one wishes to modify the action taken by the Automator in response to certain conditions one can substitute a new script in place of the old. Modifying logic for running these scripts or producing new signals would require generating a new initialization script and restarting the Automator, but this could still be accomplished without interrupting the front end machine of the control system.

5.3 Diagnostic Advantages

Finally, by allowing for rapid development of tools to monitor the control system, individuals responsible for diagnosing problems with the system being controlled can easily develop one-off diagnostic tools. Since the tool can be configured to monitor signals constantly, noting and observing transient behavior is easier than with some other methods.

6 CONCLUSION

This tool is currently under development and a Beta version is expected in the next few months. The authors feel this will provide a powerful new tool to help in the operation of large scale control systems.

7 REFERENCES

[1] M. Bickley, B. A. Bowling, D. Bryan, J. van Zeijts, K. White, S. Witherspoon, "Using Servers to Enhance Control System Capability", these proceedings (1999)

[2] J. Chen, G. Heyes, W. Akers, D. Wu and W. Watson III, "CDEV: An Object-Oriented Class Library for Developing Device Control Applications", Proceedings of ICALEPCS 1995

[3] http://www.aps.anl.gov/asd/controls/epics/EpicsDocumentation/EpicsGeneral/epics_overview.html

[4] W. Akers, "An Object-Oriented Framework for Client/Server Applications", Proceedings of ICALEPCS 1997

[5] http://www.aps.anl.gov/asd/oag/oagSofware.html

[6] http://www.aps.anl.gov/asd/controls/epics/EpicsDocumentation/ExtensionsManuals/Sequencer/snl_seq.ps