A GENERIC FINITE STATE MACHINE FRAMEWORK FOR THE ACNET CONTROL SYSTEM *

L. Carmichael#, A. Warner, FNAL, Batavia, IL 60543 U.S.A.*

Abstract

A significant level of automation and flexibility has been added to the ACNET control system through the development of a Java-based Finite State Machine (FSM) infrastructure. These FSMs are integrated into ACNET and allow users to easily build, test and execute scripts that have full access to ACNET's functionality. In this paper, a description will be given of the FSM design and its ties to the Java-based Data Acquisition Engine (DAE) framework. Each FSM is part of a client-server model with FSM display clients using Remote Method Invocation (RMI) to communicate with DAE servers heavily coupled to ACNET. A web-based monitoring system that allows users to utilize browsers to observe persistent FSMs will also be discussed. Finally, some key implementations such as the crash recovery FSM developed for the Electron Cooling machine protection system will be presented.

INTRODUCTION

Operation of an accelerator control system involves the creation of many transient and persistent tasks. These tasks generally access and integrate various components of the control system, such as accelerator devices, databases, timing systems, etc. A Finite State Machine (FSM) framework with the capacity to respond to state and input data and with hooks into different parts of the control system provides a mechanism to generate these tasks in a simple and reusable form.

A Java-based Data Acquisition Engine (DAE) infrastructure [1] has been used at Fermilab for many years to provide a Java layer to ACNET data acquisition and control. This infrastructure has been extended to provide a generic FSM framework embedded in a client-server model. FSM client applications start jobs on remote FSM servers which collect and process all input data and return state and requested data to the client. This framework is illustrated in Figure 1. These FSMs provide users with the ability to quickly build, test and deploy reusable tasks.

This paper will provide a description of the FSM structure and functionality, a detailed look at the client-server framework and an overview of the web tier that allows for the monitoring of persistent FSMs. Additionally, several key FSM implementations will be covered, including a detailed illustration of the crash recovery FSM that serves as the regulation component of the Pelletron's Machine Protection System (MPS) [2].

* FNAL is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

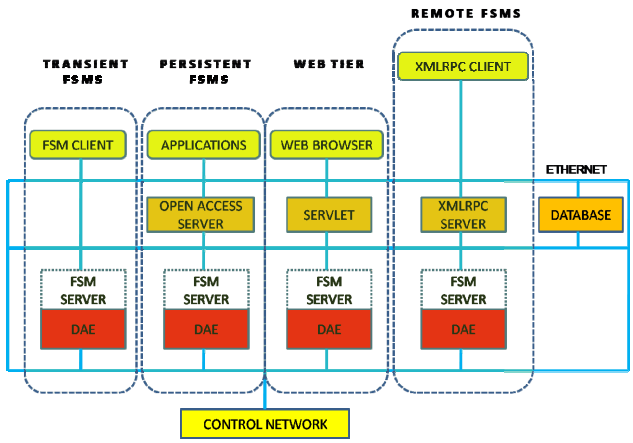

Figure 1: FSM framework.

FSM DESCRIPTION

The FSM design is based loosely upon StateCharts[3]. Each FSM is constructed with an entrance state, an exit state and a set of intermediate states. Within each state, there can be multiple transitions and actions defined. The actions can be classified by when they are evaluated; on entering, exiting or periodically within the state. Furthermore, each action is conditional and if fired can result in devices being set, database tables being inserted, etc. Figure 2 shows the FSM Builder application that is used to construct the FSMs and provides a graphical illustration of the FSM structure.

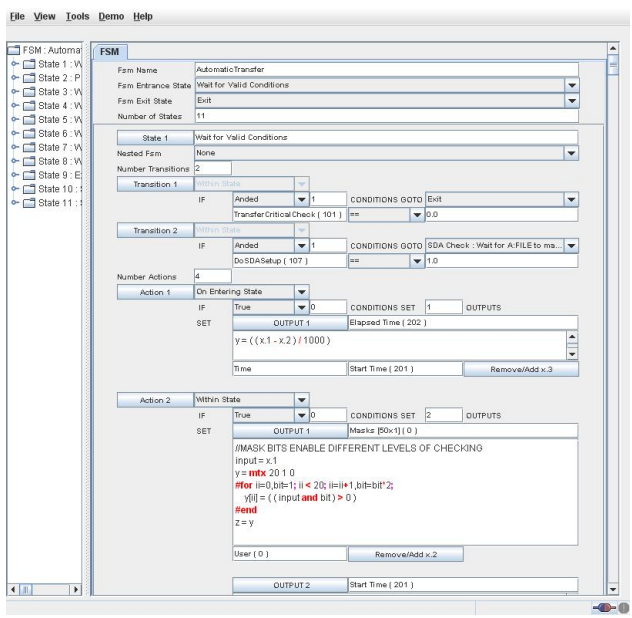

Figure 2: FSM Builder application.

FSM Functionality

The execution of a FSM occurs as a single threaded process that involves collecting and processing all state inputs before executing any of the fired actions or transitions. A list of the FSM attributes, including allowable inputs and outputs, is given as follows:

- FSM inputs fall into a wide range of categories including device readings, events, database queries, ACNET messages, XML-RPC requests, etc.
- FSM outputs can be classified as device settings, event triggers, database inserts, ACNET messages, internal variables, etc.
- An expression parser is the central component of the FSM. Collected inputs are first embedded in expression strings. The parser then employs a set of predefined functions and Java reflection in order to evaluate the output of these expressions. Once computed, the output type is used to dictate what is done with these calculated values.
- FSM timing is user specified and is driven by the return rate of the devices defined in a particular state. On entering a state, the FSM blocks until fresh readings are received. It then continues on with processing the state.

The FSM execution process is handled by the server component of the FSM framework.

CLIENT-SERVER FRAMEWORK

The FSM client is a Java application that uses Remote Method Invocation (RMI) to communicate with the FSM server. The client starts and stops jobs on the remote server and also receives state and any requested data from the server. Figure 3 illustrates the FSM Display application which is used to launch the FSM. As shown, the states are traversed and outputs are displayed with data returned from the server. Users may also use widgets on this application to send data to the server. FSMs launched with this application persist only as long as the application is open.

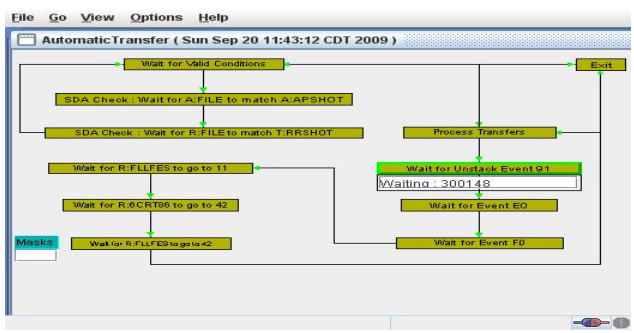

Figure 3: FSM display application.

Main Server

The central FSM server exists as a component of the DAE infrastructure. The FSM class is imbedded in the Java-based Data Acquisition Engine (DAE) infrastructure [1] that sits on top of the Fermilab accelerator control network (ACNET). The ACNET control system is a 3-tier system that uses a connectionless User Datagram Protocol (UDP) to connect different machines. The DAE infrastructure refers to a client server model where Java clients use RMI to communicate with the DAE servers that are tied directly to the ACNET control system. The DAE servers are on high bandwidth nodes which emit and listen to multicast messages, work with raw bytes and speak proprietary ACNET. The DAE framework introduces the concept of a job which consists of a 6-tuple; a disposition to send data to, a source to receive data from, an item to describe the data, an event that specifies the data collection rate, a user to provide security and job control. The FSM server can be viewed as an item in the DAE framework while the FSM client is the disposition that receives data. The process of launching a FSM can be regarded as starting a DAE job where the server (item) collects the data from the accelerator (source), processes it and returns it to the client (disposition).

Alternate Servers

The framework that has been discussed so far handles the execution of transitory FSMs. An extension to this framework, which utilizes the Open Access Client (OAC) architecture [1] developed at Fermilab, has been established to handle persistent FSMs. The OAC architecture exists at the ACNET middle tier and serves to emulate the front end tier. It provides access to all of the ACNET protocol available at the frontend without requiring front end programming. This protocol includes alarms, device downloads, setting uploads, etc. The FSM OAC links persistent FSM jobs to specific ACNET devices. The control values of these devices are used to start and stop FSM jobs. Device settings are sent to the FSM server and data returned from the server are placed in the device readings. FSM job persistence is guaranteed for the life of the OAC.

An alternative server is currently under development that utilizes XML-RPC [4] to allow remote access to the FSM framework. It is envisioned that not all FSM users will be within the firewall. So, the XML-RPC server would provide them with limited access to the FSM server. These users would only be able to start FSMs in safe mode with settings and any other control access to the control system disabled. This server has been developed and is illustrated in Figure 1. It is currently undergoing beta testing and is expected to be released shortly.

WEB TIER

A web interface has been developed for the FSM infrastructure that allows browsers to act as the client to the FSM server. The HTML pages that serve as the interface use the Asynchronous JavaScript and XML (AJAX) [5] protocol and allow users to view and monitor persistent FSMs. Users may also start FSMs, but only in safe mode. The HTML pages illustrate the concept of

FSM views that is provided by the FSM framework. This refers to the client-side processing that is undergone by data returned to the FSM client. This processing utilizes expression parsing and Java reflection to produce user panels and graphs from the data. These panels or graphs are then converted to Scalar Vector Graphics (SVG) and displayed on the web browser. Figure 4 illustrates some of these FSM views. In order to mitigate security concerns, any FSM launched from the browser is done in safe mode with all settings and control disabled.

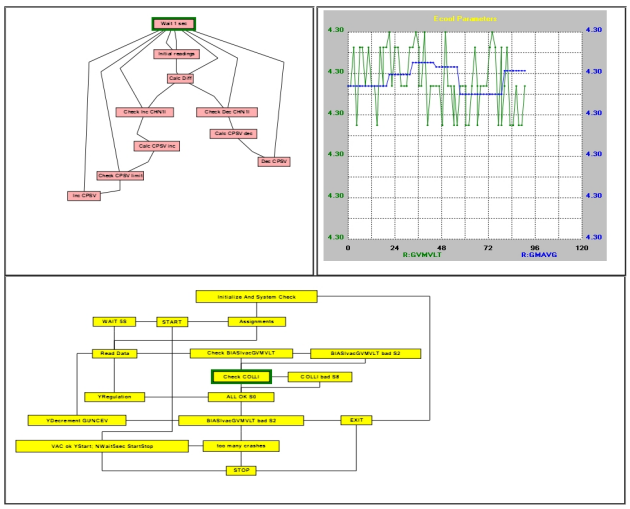

Figure 4: FSM views.

FSM EXAMPLES

The FSM framework is well integrated into the ACNET control system at Fermilab and several operational implementations have been developed for the collider physics program. For example, the electron cooling facility utilizes this FSM framework in the implementation of its Machine Protection System (MPS) recovery scheme. The Electron cooler is based on a 4.3-MV, 0.1-A, DC electrostatic accelerator (Pelletron) for which current losses have to remain low (~10^{-5}) in order to operate reliably. The Pelletron itself is subject to high-voltage discharges and other system interruptions which are monitored and recovered via an FSM.

The FSM provides the functionality to maintain or recover the nominal electron beam current whenever drifts or unexpected changes occur (e.g.: operator errors), it reduces the gun current as needed to compensate for soft fault conditions such as vacuum deterioration and slow (i.e. ~milliseconds) HV drops, and it executes a consistent set of steps and checks whenever the machine needs to recover from a trip. It is particularly important when the trip is due to a large HV discharge. In addition to the crash recovery process during electron cooling operations, separate FSMs operate to perform slow feed-back (1 Hz) of the Pelletron's voltage regulation system and energy set-point respectively. These processes counteract long term drifts of the machine due to temperature changes and other slow effects.

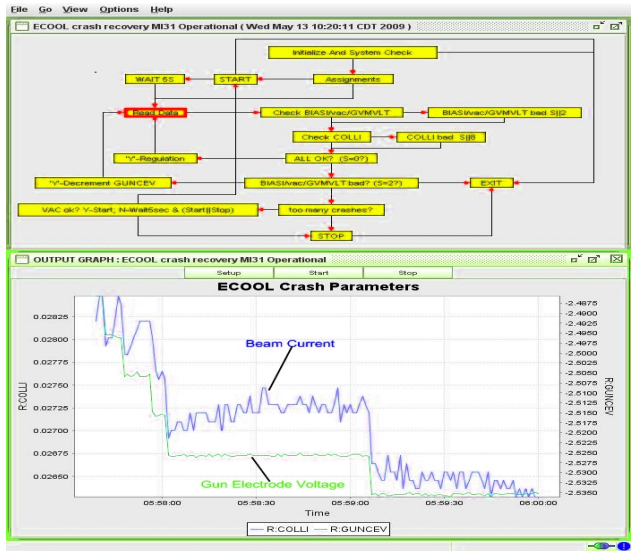

Figure 5: FSM display. The plot shows the FSM regulating the gun control electrode voltage in order to reduce the beam current back to its nominal value.

The Crash recovery FSM has proven to be very useful operationally as it significantly reduces manual interventions while running beam. It in turn increased the electron beam uptime, which is now close to 100% (except for occasional hardware failures). The FSM infrastructure has become an integral part of the Pelletron's machine protection scheme at various levels and helped in streamlining fault analyses. It not only captures operational knowledge in the regulation script, but also allows for the easy testing and implementation of additional scripts to improve operational efficiency.

SUMMARY

The FSM framework is a crucial component of the ACNET control system. Its ability to quickly and robustly build, test and launch automated tasks has become a great asset. It has operated with a high level of reliability as emphasized by its integration into the Pelletron's MPS. The web tier is a recent addition to the FSM framework, but its capacity to provide different views of persistent FSMs is expected to quite useful.

REFERENCES

[1] Guide to Data Acquisition Engine, K. Cahill, Beams Document 666-v1.

[2] Arden Warner, Linden Carmichael, et al, The Design and Implementation of the Machine Protection system for the Fermilab Electron Cooling Facility. DIPAC 09.

[3] Harel, David, StateCharts: A visual Formalism for Complex Systems. Science of Computer Programming (1987).

[4] Simon St. Laurent, "Programming Web sevices with XML-RPC (2001).

[5] Jesse James, AJAX: A new approach to web applications.