Automated Task Scheduling Using Multiple FSMs at Fermilab

Linden Carmichael
Fermilab, Batavia, IL, U.S.A.

Abstract

Fermilab will enter a new operational era with the installation of the Main Injector (MI), a circular accelerator and the Recycler (RCYC), a anti-proton storage ring. This transition will induce a significant increase in the complexity of the operational scenarios undertaken and has prompted the design of a more sophisticated task scheduler, one that is capable of providing a level of versatility, efficiency and automation to the process of generating task schedules. Task schedules, which serve to co-ordinate the various Fermilab machines, are currently defined by utilizing a de-centralized repository of heuristics related to how the different machines interact. The proposed task scheduler encapsulates these heuristics by resolving task schedules into sets of task groupings termed modules and then assigning various attributes to these modules. These attributes, which denote heuristics such as how quicky MI resets can be generated, serve to define the relationship of tasks within modules and modules within task schedules. The implementation of this task scheduler involves a Graphical User Interface (GUI) to create schedule specifications, multiple Finite-State Machines (FSMs) to resolve modules and attributes into a task timeline and a hardware module which dispenses the tasks to the various machines. This paper will illustrate a sample task schedule and provide a brief discussion of the implementation of the FSMs.

1 Introduction

The physical layout of the Fermilab facility includes a linear accelerator (LINAC), three circular accelerators (BSTR, MI, TEV), three storage rings (ACC, DBNCHR, RCYC) and an assorted number of beam lines. These different machines are connected and co-ordinated through a 10 MHZ hardware link (T-CLK) utilizing 256 manchester encoded events. Each machine task is denoted by a unique 8-bit event mask which is generated by the Time-Line Generator (TLG) hardware module. The prior incarnation of the task scheduler [Ref 1.] required that each time slot in a schedule be independently specified with an event mask. These schedules were then downloaded as flat files to the TLG which would then place the events on T-CLK.

The design for the new task scheduler is based, in part, on the work done at CERN with its PS accelerator network [Ref. 2]. The emphasis, at CERN, was in creating a comprehensive Knowledge-Base of its various accelerators and then designing a rule-based task scheduler that utilized this Knowledge-Base in order to generate task schedules. The task scheduler under development at Fermilab represents more of a parameterization of task schedules through the module definitions. The attributes attached to each module provide a method of automatically adjusting task placement as the various operational scenarios tend to fluctuate somewhat, initially, before becoming well defined.

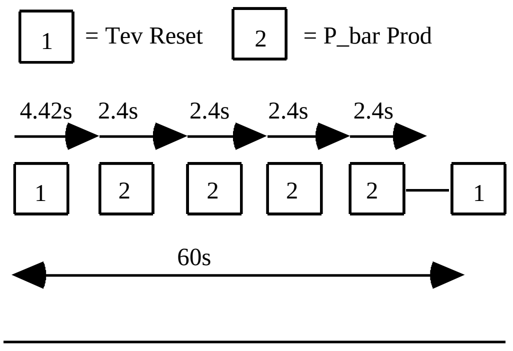

Figure 1. Sample Schedule

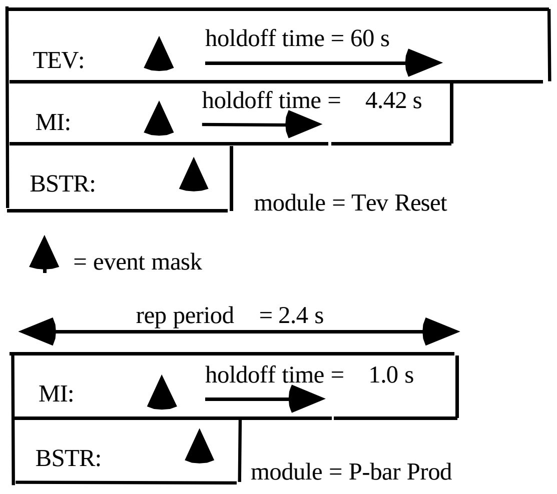

Figure 2. Sample Modules

2 Task schedules

The process of defining task schedules is greatly simplified by taking advantage of the inherent dependency that exists between tasks and creating task groupings termed modules, as illustrated in Figure 2. Each module has a set of attributes assigned to it which serve to characterize the behavior of tasks within the module. These attributes fall into one of the following categories: *source* attributes which dictate the influence of outside parameters on module tasks, *placement* attributes which describe where tasks are placed within the time range covered by the module and *constraint* attributes which define constraints on task placement. Figure 2. desribes two typical modules that are used to inject beam into the TEV and to stack anti-protons in a storage ring. The Tev Reset module co-ordinates the tasks in the BSTR, MI and TEV accelerators and has *source* and *constraint* attributes attached to it. The *source* attribute dictates how many BSTR batches to inject into the MI while the *constraint* attribute, ***holdoff time,*** limits how soon another TEV reset can be generated after the one in this module is placed. The P_bar Prod module controls the stacking of the anti-proton storage ring and has a *constraint* attribute, ***rep period,*** attached to it which limits the number of stacking cycles that can be packed into a given time range. Figure 1. illustates the output of the task scheduler when given the two previously defined modules and the specifications to inject beam into the TEV before stacking as many cycles as possible. Adjusting the attributes, ***rep period*** and ***holdoff time***, allows the operators to easily tailor the task schedule to the desired operational needs. The key benefits obtained from this formulation of the task scheduler is the flexiblity imbeded in the task schedules, as illustrated in this example and the level of automation provided, as will be shown in the following section.

3 Scheduler implementation

The procedure for creating and executing task schedules involves three layers of processing, as illustrated in Figure 3. The first layer defines the console environment and consists of a dedicated GUI for creating task schedule specifications and defining modules which are then downloaded to the FSMs at a second layer . Also, there exists an assorted number of applications which, through the *source* attributes, are capable of adjusting key elements of the various modules being generated by these FSMs. The second layer resides on a VME board and consists of FSMs which compute task placement and then directs the resultant task timeline to the TLG module. This module denotes the final level of processing and serves to place the 8-bit event representation of the tasks on T-CLK.

The primary emphasis of this section is a description of the second layer. This layer, as illustrated in Figure 3., consists of an interface and three FSMs. The interface, designated by MOOC (Minimally Object Oriented Communication), is a C-library that utilizes reading and setting methods [Ref. 3] to communicate between the in-house ACNET (Accelerator Network) protocol and the VxWorks operating system being utilized by the FSMs.

The FSM **Test** takes the schedule specifications and attribute parameters downloaded by the GUI and determines the corresponding time slots for the various tasks. These time slots are then reported back to the GUI. The FSM **Place** operates similarily with the exception being that other applications, in addition to the GUI schedule editor, are capable of influencing task placement. Once a valid timeline of tasks is produced, this FSM sends a signal to the FSM **Play** which then proceeds to send the tasks to the TLG module at the designated times. Furthermore, the FSM **Play** denotes task schedules as being either primary or secondary with the distinction being that on completion of a secondary schedule, the primary schedule resumes playing.

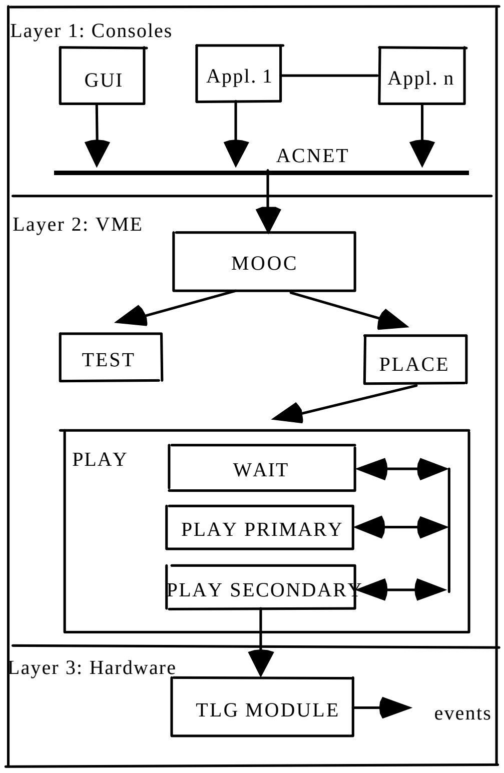

Figure 3: Architecture

4 Conclusion

The focus of this work has been on creating a task scheduler that is versatile in order to handle the initial, variable nature of the operational scenarios before they become well defined. Additionally, efficiency and an adequate level of automation were also motivating factors for developing this formulation of the task scheduler. The developmental cycle for this project is characterized by three phases. Phase I, which has been completed, involved a predominantly hardware effort in moving the TLG module from a CAMAC to a VME implementation. Phase II, which nears completion , involves the creation of the GUI at layer one and designing the FSMs to communicate with the new TLG module. The final phase involves the utilization of multiple DSPs on an ALEX board in a multi-processing environment .

References

[1] R. Johnson, W. Knopf, A. Thomas: "Time-Line Generator and Controller" Controls Software Release #109, Sept. 14, 1983

[2] F. Perriollat, C. Serre: "The new CERN PS control system overview and status", ICALEPCS, Berlin, Germany, Oct. 18-23, 1993, Nucl. Insr. and Meth. A352(1994) 86

[3] J. Utterback, "Making Data Accessible to ACNET from aVxWorks Front End",ADControls, Aug.21,1995.