THE PROCEDURE EXECUTION MANAGER AND ITS APPLICATION TO ADVANCED PHOTON SOURCE OPERATION

M. Borland
Advanced Photon Source, Argonne National Laboratory
9700 South Cass Avenue, Argonne, Illinois 60439 USA

Abstract

The Procedure Execution Manager (PEM) combines a complete scripting environment for coding accelerator operation procedures with a manager application for executing and monitoring the procedures. PEM is based on Tcl/Tk, a supporting widget library, and the dp-tcl extension for distributed processing. The scripting environment provides support for distributed, parallel execution of procedures along with join and abort operations. Nesting of procedures is supported, permiting the same code to run as a top-level procedure under operator control or as a subroutine under control of another procedure. The manager application allows an operator to execute one or more procedures in automatic, semi-automatic, or manual modes. It also provides a standard way for operators to interact with procedures.

A number of successful applications of PEM to accelerator operations have been made to date. These include start-up, shutdown, and other control of the positron accumulator ring (PAR), low-energy transport (LET) lines, and the booster rf systems. The PAR/LET procedures make nested use of PEM's ability to run parallel procedures. There are also a number of procedures to guide and assist tune-up operations, to make accelerator physics measurements, and to diagnose equipment. Because of the success of the existing procedures, expanded use of PEM is planned.

1 INTRODUCTION

The motivation for creating PEM is the observation that many accelerator operations take the form of written or memorized procedures. It is not unheard of for accelerator operators to control a sophisticated accelerator with the guidance of a checklist written on paper. While this works, it is a long way from being automated. Sometimes automation of an activity is hampered by lack of computer control of a part of the facility or by the difficult nature of the equipment being controlled. Even in such cases, a combination of automatic and manual steps can often be used to produce partially automated operation. This partially automated procedure can include instructions to and advice for the operator and, as such, will result in a greater uniformity of operation. For example, some PEM scripts used at APS will set up a diagnostic device, then ask the operator to perform a specific tuning operation that would be difficult if not impossible for a computer to perform with current technology.

PEM's ability to execute procedures in parallel was similarly developed from the observation that often times several operations can be done simultaneously. For example, one can standardize DC magnets while warming up pulsed power supplies. With a small system, this can be done manually. However, with a large system this becomes difficult, and operators tend to do things somewhat sequentially, with attendant loss of time. For this reason, a design goal for PEM was to allow parallel procedure execution.

Finally, many procedural activities have individual steps that are themselves procedures. These steps may in some circumstances be performed separately as a standalone procedure. Based on this observation, another design goal for PEM was that any procedure should be easily embeddable within another.

Given that some procedures require human input and guidance, even while largely automated, PEM has three modes in which it may execute a procedure. *Automatic* mode accepts no human input at any point. *Semi-automatic* mode accepts human input at the beginning of a procedure, in order to allow selection of options and changing of parameters. *Manual* is like *Semi-automatic* except that at each defined step of the procedure the user must press a button indicating that he wants to proceed. A procedure that is called from within another procedure is run in *Automatic* mode, thus removing any user interaction. In this case, the calling procedure is responsible for providing input to the called procedure in place of human input. PEM procedures may switch modes internally to circumvent this, but it is uncommon and is discouraged.

Another principle underlying the creation of PEM is the desire and need to separate the interface from the algorithm, in order to make higher-level automation possible. For example, some storage ring facilities use orbit correction algorithms that are embedded in a graphical user interface, making it impossible to automate. Orbit correction and beamline steering at APS [1,2] are performed from several graphical user interfaces (GUIs) (including PEM), all of which use a common generic feedback program to execute the actual correction. This feedback program has no GUI and is used not only for orbit correction but for other tasks that are mathematically identical. The PEM orbit steering procedures also use Tcl/Tk library routines that are shared with various GUI applications.

When a developer creates a new PEM procedure, he does not create a GUI. PEM provides a GUI, as well as mechanisms for determining what mode the interface is in, so that procedures may run outside of a GUI environment. Typically, the developer of even a complicated PEM procedure will only create an initialization dialog box. This dialog box will appear only when the procedure is run in the proper context (i.e., by a person).

2 PEM OPERATOR INTERFACE

The PEM manager GUI consists of five areas. The first is a scrollable text window for status and error message reporting. The second is a scrollable list of available procedures. The third is a scrollable text box used to display detailed information about individual procedures.

0-7803-4376-X/98/$10.00

The fourth is a scrollable list of currently executing procedures. Finally, there are various buttons that select execution mode, lock and unlock execution, start procedures, and abort procedures.

PEM is configured by a file that contains a list of procedure names, along with optional arguments that specify default values for the procedure. These arguments are specified using a convention given in the next section. From this configuration file, PEM creates a procedure selection list. Clicking on the name of any procedure in this list causes the manager application to display information about the procedure, if the programmer has supplied any.

Prior to executing a procedure, the user may select the execution mode, i.e., *Automatic*, *Semi-Automatic*, or *Manual*. The manager application also features an execution lockout that reduces the chance of accidentally executing a procedure. Once a procedure has been selected, the mode chosen, and the lockout released, the user clicks on the **Execute** button to execute the procedure. Following this, PEM launches a subprocess to execute the procedure. The name of the subprocess appears in a list of executing processes that is part of the manager window. The user may select any process from this list and abort it using the **Abort** button on the manager window.

PEM automatically creates an execution dialog to show the progress of the subprocess. Except in *Automatic* mode, the subprocess automatically displays an initialization dialog if one has been set up by the programmer; this is described below in Section 3.2. The execution dialog has its own scrollable text window for status output, as well as a scrollable area showing the names of the procedure steps as they are executed (again, except in *Automatic* mode). These names are coded green when being executed and red when halted for user interaction. The dialog has a **Continue** button and **Done** button that are used in *Manual* mode to proceed from step to step and to acknowledge termination of the procedure. There is also a button to acknowledge errors.

3 PEM PROGRAMMING FACILITIES

PEM provides a number of facilities for creation of procedures [3]. All PEM procedures are written in Tcl/Tk and may be used outside of PEM like any other Tcl/Tk code. Because we have many other Tcl/Tk procedures, we distinguish PEM procedures with the letters **Mp** (Machine procedure) in the name. The **Mp** designation indicates either that a procedure is part of the PEM environment or it uses aspects of that environment. The PEM environment defines several procedures that the programmer invokes when writing machine procedures. Among these are **APSMpStep**, **APSMpParallel**, **APSMpJoin**, **APSMpAbort**, and **APSMpInterface**. These procedures and other aspects of the PEM development environment are described below.

3.1 Argument Passing

In order to make Tcl/Tk procedures easier to use and upgrade, we have adopted a procedure calling convention. A procedure call is always of the form: *procName* [*fixedArgs*] [*variableArgs*], where items in square brackets may be absent for any particular procedure. *fixedArgs* are arguments that must be given in an established order; not doing so will result in a Tcl error. *variableArgs* are always optional, may come in any order, and take the form of tag-value pairs, specifically *-tag value*. The vast majority of procedures have only variable arguments in order to enhance code readability.

3.2 Initialization Dialogs

As described above, PEM's interaction with the user is context-dependent. At the beginning of each PEM procedure, the programmer inserts an **APSMpStep** with an **init** tag and the name of a procedure (the "initialization" procedure). In *Manual* and *Semi-Automatic* modes, PEM executes the initialization procedure, which typically brings up a GUI dialog to allow the user to specify initial parameters for execution of the main procedure. If a PEM procedure is called by another PEM procedure, the mode is set to *Automatic*, and hence the initialization dialog is never invoked. Instead, the calling procedure is responsible for providing proper initialization for the called procedure. The argument-passing mechanism discussed in the previous paragraph is important here, since it allows a procedure to be called with only the arguments needed in a given context. When a procedure is executed as a subroutine, it is generally supplied with a large number of arguments to substitute for the parameters normally supplied by a user.

3.3 Labelling of Steps

The second function of **APSMpStep** is to permit labeling of sections of a procedure, in order to define steps within the procedure. A step may comprise any number of actual operations. When a procedure is invoked in *Manual* and *Semi-Automatic* modes, the PEM execution dialog shows the progress of the script through these labeled steps. In *Manual* mode, the user must press the **Continue** button on the PEM dialog to permit execution of each step.

Labelling of steps also permits PEM to ensure servicing of user abort requests. When a user asks to have a PEM procedure aborted (via the PEM manager application), this request is serviced at the next occurrence of a Tcl/Tk **update** or **tkwait** call. Such a call is guaranteed to happen prior to each step of the procedure.

3.4 Parallel Processes

Creating parallel processes with PEM is simply matter of providing the **APSMpParallel** procedure with the Tcl/Tk command that is to be executed. One may also specify the host machine on which to execute the command and the PEM mode under which the command should be executed. The defaults are to execute the command on the present host and in the present PEM mode. (In most cases, the mode is set to *Automatic*.) For each process created in this fashion, **APSMpParallel** returns a unique identifier that may be used to control the process.

Having created one or more PEM parallel processes, it is typical to want to wait for these processes to complete.

This is done using **APSMpJoin.** If **APSMpJoin** is not used within a procedure that creates parallel processes, an implied join is performed prior to returning from the procedure. **APSMpJoin** takes a single argument, namely the identifier of a process for which to wait. A series of **APSMpJoin** statements is used to ensure that all parallel processes are complete. Each **APSMpJoin** returns a catchable error. In the case of an error, the programmer may elect to abort other parallel processes.

Aborting parallel processes is accomplished using **APSMpAbort.** This procedure accepts a list of identifiers for processes to abort or it can abort all processes. It can also be used to set up a conditional abort, wherein if any of the listed parallel processes returns an error, then all of the listed processes are aborted. This prevents blocking of **APSMpJoin** on one process when another process has encountered an error.

4 EXAMPLES

4.1 PAR/LET Start-Up Procedures

This was one of the first applications of PEM and is the most mature and sophisticated one to date. The PAR/LET consists of three main systems: the PAR itself, the linac-to-PAR transport line (LTP), and the PAR-to-booster transport line (PTB). While the LTP and PTB are simple and nearly identical, the PAR is somewhat complicated. The LTP and PTB both contain only DC magnets, vacuum hardware (including remotely controlled valves), and various diagnostics (including insertable screens). The PAR contains these components, plus two rf systems and four pulsed power supplies.

The PTB and LTP each have seven low-level machine procedures for functions like turning power supplies on and off, conditioning magnets, and clearing the aperture of obstructing valves and screens. These functions are implemented as stand-alone machine procedures and may be individually executed from PEM. In addition, there are start-up and shut-down procedures for the LTP and PTB that make use of these low-level procedures.

The PAR procedures show a similar structure, with the addition of procedures for the rf and pulsed magnet systems, for a total of 17 procedures. Again, these are potentially stand-alone procedures but are also integrated into a pair of start-up and shut-down procedures. The relatively time-consuming operations of pulsed power supply warm-up and DC magnet conditioning are run in parallel.

Finally, at the highest level there is a pair of start-up and shut-down procedures for the entire PAR/LET system. These execute the PAR, PTB, and LTP procedures in parallel. In addition, the start-up procedure guides the operator through several manual steps necessary to obtain beam.

4.2 APS Ring Procedures

One difficult aspect of APS ring operation has been tuning up injection for high efficiency. A tuning procedure was developed by physicists in the course of machine studies and subsequently implemented as a PEM procedure. Because much of the tuning involves an operator looking at a scope trace, insertable screen, or similar diagnostic, this might at first seem a difficult task for automation. However, if the goal is simply to improve convenience for the operator and standardize operational methods, PEM is quite useful in this application. Specifically, the PEM procedure walks the operator through the tune-up. For some steps, the procedure helps the operator by inserting a screen into the beam or bringing up a needed control system display. For such steps, it gives instructions on what to look at and what to try to achieve. This procedure makes use of the *Manual* execution mode of PEM and is not intended to be embedded within another procedure.

Beamline steering in the APS involves a complicated series of steps to ensure that the orbit is maintained within acceptable tolerances. This procedure is implemented using PEM and makes use of the ability to switch execution mode inside of a procedure. It does this in order to allow operators to complete activities using other interfaces. In this way, PEM allows the use of other code as part of a procedure without any "real" connection between PEM and the other code. Again, there is no intention of embedding this type of procedure in a larger one.

5 CONCLUDING REMARKS

The success of the PAR/LET procedure makes it clear that such procedures are desirable for the other systems. It is also clear that more effort is involved in writing a procedure robust enough that it can be embedded deep inside another procedure for automatic execution. The problem is partly one of anticipating all of the potential problems with the hardware. Still, we intend eventually to write start-up and shut-down procedures for all of the systems. In principle, this would allow starting up or shutting down the entire facility with a single operation, though the procedures are clearly desirable even if one never wants to run them all in parallel.

Since PEM was designed with multi-workstation and multi-operator tasks in mind, it would be desirable to be able to execute procedures on the least loaded workstation of a cluster. This would be particularly useful for computationally intensive procedures. Presently, the multi-workstation capabilities in PEM are disabled due to problems with IPC software from the vendor.

6 ACKNOWLEDGMENT

PEM was implemented by C. W. Saunders, formerly of APS, based on concepts developed by M. Borland and C. W. Saunders. Several of the scripts described above were created or contributed to by L. Emery (APS). Work is supported by the U.S. Department of Energy, Office of Basic Energy Sciences, under Contract No. W-31-109-ENG-38.

7 REFERENCES

[1] L. Emery, M. Borland, "Advancements in Orbit Drift Correction in the Advanced Photon Source Storage Ring," these proceedings.

[2] M. Borland, "Applications Toolkit for Accelerator Control and Analysis," these proceedings.

[3] C. W. Saunders, "PEM-Procedure Execution Manager," http://www.aps.anl.gov/asd/oag/manuals/APSPEM/APSPEM4.html.