Onebm: the ELETTRA Framework for Programmable Machine Operations

D. Bulfone, C. Scafuri
Sincrotrone Trieste, ss 14 Km. 163.5, 34012 Basovizza, Trieste, Italy

Abstract

The Onebm (one button machine) project has been developed to automate the routine operations at ELETTRA. Its framework consists of a programmable task manager with a Motif user interface, a set of operation description files and a collection of software modules.

The programmable task manager can spawn and track the status of any valid UNIX executable, either binary object or shell script. The spawn mechanism is based on the standard "fork()" and "exec()" system calls. The task manager also intercepts the standard output stream of each task by redirecting it to a UNIX pipe. The intercepted output stream is displayed on demand to the operator. The task manager checks also the output stream for the presence of messages identified by predefined tokens. These messages are used for error notification and are displyed on a special window. An important point of the task manager is its capability to handle many tasks concurrently, leading to substantial savings in the execution times. These characteristics, associated with the minimization of operator errors, produce an increase of the available beam time.

The description files are simple ASCII files following a predefined syntax containing the logic flow of the tasks and the associated parameters. The rules to start a task are boolean expressions whose factors are the exit status of other tasks. This formalism has been found capable of expressing any operation so far analyzed.

Each software module is a program designed to handle a well defined task on the machine, usually involving a single machine subsystem. A C++ library has been written to facilitate the development of new modules. The classes of this library encapsulate all the ELETTRA field access routines, provide utilities to generate the special error messages recognized by the task manager and terminate the module with the correct exit status.

A detailed explanation of the design issues, implementation choices and techniques is given.

1 Machine automation

The operations needed by a user oriented facility like a synchrotron radiation source are usually very repetitive and firmly established. An "operation" is defined as the set of steps necessary to bring the machine from one operating condition to another, like the switching on of machine equipment after a shutdown or the daily re-filling for user shifts.

The goal of the Onebm project is to automize as much as possible such operations, which should be ideally carried out by "just pressing one button" on the Control Room console. The associated advantages are the minimization of the time needed to perform an operation and the almost complete elimination of operator errors.

The project result is the Onebm framework, which includes the Onebm main program, the specification of the syntax to write an operation description file, the definition of rules for accepting external programs as Onebm modules and a support library for writing new Onebm modules.

1.1 Overview and definitions

The strategy followed to handle the problem of automating an operation is the age old idea of partitioning the problem into sub problems and solve them separately. Therefore the "operation" is subdivided into a series of "tasks". A task usually handles a well defined subsystem of the plant. It is started at a certain moment and must terminate with a clear indication of its failure or success. A task is carried out by a dedicated "module" which consists of a UNIX executable. Modules are started with an arbitrary number of input parameters to add flexibility.

The execution of the tasks must be co-ordinated so that a given task can be run only after the successful termination of a well defined set of tasks. The rules that regulate the task execution together with the associated parameters constitute the so called high level description of the operation [1]. We have devised a very simple syntax to express these rules in a human and machine readable format. An "operation description file" is written using a standard text editor and can be interpreted by the Onebm main program.

Onebm starts the operation modules following the precedence rules written in the operation description file. It concurrently executes all the modules that are not blocked by the successful termination of other modules. This feature, based on the multi-tasking capability of UNIX, minimizes the time needed to perform a given operation.

1.2 Planning an operation

If we change the word “operation” with “project”, we see that we can take advantage of the PERT technique of Management Science [2] to analyze a given machine operation and the schedule of its associated tasks. The PERT algorithm is widely used for business and management planning, thanks also to some very powerful and easy to use PC based implementations. This mathematical tool identifies the critical time path of a given operation pointing out bottlenecks.

A limitation of PERT is that it has no concept of task failure and does not handle alternative routes to start a task. More formally, the condition for the execution of a task is expressed as the logical product (boolean AND) of the successful termination of a set of tasks (a "milestone" in

the PERT terminology). In Onebm we can express the starting condition of a task as a more general boolean expression of sums (boolean OR) and products of both the success or failure of a set of tasks. This richer set of rules is necessary to handle alternative paths for task execution and to plan some failure recovery actions (rules containing negated terms).

Apart from this limitation we have found that a PERT chart (figure 1) is a very handy tool for the definition of a new operation. It shows both the main execution path and the task partitioning. Later we can exploit the capabilities of the Onebm rules to handle the alternative or fault recovery paths of execution.

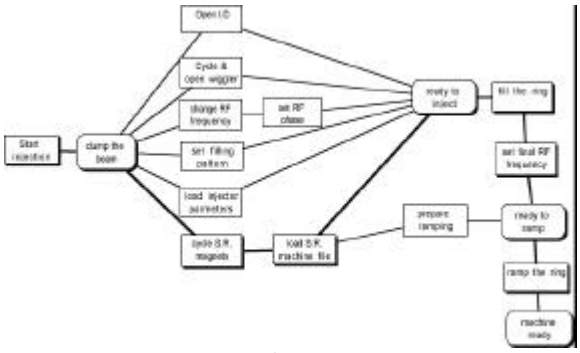

Fig.1: PERT chart of the injection operation.

1.3 Dealing with faults and errors

Errors and failures can occur during the execution of operation tasks. Two schemes are foreseen in this case.

Onebm can trigger the execution of a dedicated error recovery task on the faulty termination of a set of tasks by using negated terms in the starting expression.

Alternatively, Onebm detects the fault condition and allows the operator to intervene manually. The operator can restart the failed task, which in many cases solves the situation, or he can try to solve the problem by other means, e.g. by resetting a faulty equipment. The operator then marks the task as "done", informing Onebm that it has been completed, and automatically unlocks the remaining operation tasks.

It should be noted that Onebm is part of the ELETTRA Control System [3] so that the operator has access to many other tools which support his decisions in case of problems: the list of machine alarms and their history log-file [4], machine physics programs and individual control panels with extensive diagnostics for each machine equipment.

If the problem is not immediately solvable, the operator marks the task as "ended with an unrecoverable error". This usually inhibits the operation termination, which must be repeated from the beginning after the problem has been fixed.

2 Onebm modules

Almost any UNIX executable, binary or shell script, can be used as a Onebm module. The following are the few simple rules that the candidate module must comply with.

- It must be based on UNIX conventions, system calls, libraries and system resources.
- It must take its execution parameters from the command line.
- It must print any information or error message on the standard output.
- It must terminate clearly at a given moment returning a well defined exit code to the calling environment, indicating the success or failure of the execution.
- It must run as a regular process without turning itself into a background process.

These rules are followed by any well behaved UNIX executable. Many of the already existing control room programs could be used by Onebm with little or no modification at all.

2.1 Special strings and exit conventions

Onebm recognizes some special strings written by a module to its standard output. These special strings are displayed on a separate section of the module's message area on the Onebm user interface. These special strings are useful to help the operator to quickly identify any problem or malfunction of the running module.

The Onebm framework defines another set of conventions for the exit codes of a module. These exit codes extend the UNIX conventions for indicating the faulty termination of a process, so that it is possible to notify also the type of failure and force the correct exit status of the task state-machine. The use of these extensions, although highly recommended for newly designed programs, is anyway not mandatory.

All the necessary definitions for formatting the special strings and the extended exit codes are available to programmers in a C-language header file.

3 The task scheduler

Onebm represents each task as a finite state-machine. Figure 2 is a schematic of the implemented transition diagram. Shaded arrows represent transitions which can be forced "manually" by the operator through the user interface buttons.

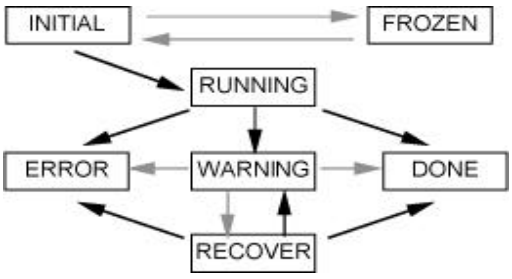

Fig.2: task state transition diagram

The different states are defined as follows:

INITIAL: is the starting state for each task; the task has never executed before.

FROZEN: the task execution is blocked manually by the operator.

RUNNING: the task is executing for the first time.
ERROR: the task execution failed with an unrecoverable error.
WARNING: the task execution failed with a recoverable error
DONE: the task execution terminated successfully.
RECOVER: the task is being re-run after a recoverable error.

3.1 The task spawner

When one of the tasks changes its state Onebm checks the logic conditions for all the other tasks. All the tasks which are found to be ready to run are spawned. The spawning is performed by means of the standard *fork()* and *exec()* UNIX system calls. Onebm forks twice so that the original copy of Onebm (the father) can continue the normal processing where the child copy of it forks again. The newly forked child starts the desired module with an *exec()* call, while the intermediate child performs a *wait()* call and suspends its execution until the spawned module terminates. Upon termination of the spawned module the intermediate child reports the module exit code to the original Onebm.

The standard output of the spawned modules is redirected to a UNIX pipe. The other end of the pipe is read by the father process. All the messages written by a module are thus intercepted and stored by Onebm. The module exit code is also sent to the main process by means of this pipe.

4 The graphical user interface

Onebm has a graphical user interface based on the Motif standard. The interface is designed to present the control room operator only with the information strictly necessary to follow the operation progress.

Each task has a dedicated area on the interface. During the task execution its label changes colour and flashes according to the state of the task. In case of problems the operator can ask for more details about a task by opening a window where all the module messages are logged. The special Onebm strings are emphasized in red and grouped in a reserved area. Another auxiliary window available on request shows some essential help messages about the task. These messages are configured by means of the operation description file.

The auxiliary area of a task contains also the buttons used to manually force the state transitions in case of malfunctions.

5 Support for writing new modules

An extensive analysis of various possible scenarios convinced us that it is better to have a collection of small and well focused programs, each of them dedicated to a single task of the operation. This is a well known idea which is, among the other things, one of the foundations of the UNIX environment.

The analysis of the various tasks showed us that the majority of them has a very repetitive pattern of actions: open the Remote Procedure Call (RPC) connection to the desired device, check for a pre-condition, write a set of values, wait and check until the desired operating point is reached, close the RPC connection; in case of errors write a diagnostic message to the standard output exploiting the special strings recognised by Onebm. All these functions have been encapsulated in a dedicated C++ class [5]. This class hides the programmer of all the details of RPC handling and, exploiting C++ polymorphism, provides a uniform set of calls for all the control system points and data types. This class supplies also the methods to automatically wait for a machine parameter to reach a desired value. The use of this class proved to be very effective. Of the twelve different modules used for the daily injection operation at ELETTRA, seven are based on the special C++ class, one is a shell script starting some standard control panels, two are existing machine physics and only two modules are dedicated Onebm programs which are not exclusively based on the special C++ class. This allowed us to save a lot of developing time and assured a consistent and reliable behaviour of the modules.

6 The injection operation

The daily injection operation at ELETTRA [6] is now fully under Onebm control. The latest improvements are a series of modules to assist the operator during the manual filling of the ring, prepare the energy ramping of the ring and automatically perform the ramp to the final operating condition. These improvements lead to a further reduction of the time needed to perform the injection which can be as short as 20 minutes.

7 Conclusions

The Onebm project met all its goals. The framework is a portable, flexible and easy to use tool.

Thanks to its modularity and to its specialized support library, modifications or extensions of existing operations are implemented in a short time and with very little effort. New operations can also be designed and installed with a reduced investment in time and manpower.

References

[1] F. Potepan “Automize Machine Operation at Control Room Level: One Button Machine Program”, ST/M-TN-96/3, Sincrotrone Trieste, June 1996.

[2] R. E. Markland “Topics in Management Science”, John Wiley & Sons, 1983, pp. 405-416.

[3] D. Bulfone: “Status and Prospects of the ELETTRA Control System”, Nucl. Instr. and Meth. A352, p. 63, 1994.

[4] P. Michelini, C. Scafuri: “A New Alarm System Processor for ELETTRA”, Proc. EPAC96, Sitges, Spain, 1996.

[5] F. Potepan: “ONEBM modules”, ST/M-TN-96/4, Sincrotrone Trieste, June 1996

[6] D. Bulfone, C. Scafuri: Automating ELETTRA operation with "one button machine". Proc. Particle Accelerator Conference, Vancouver 1997.