A new plant control software for the TCV tokamak

Y.R.Martin, S.Coda, B.P.Duval, X.Llobet, J.-M.Moret

CRPP - EPFL
Centre de Recherches en Physique des Plasmas, Association EURATOM-Confédération Suisse,
Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland

Abstract A new plant control software has been implemented on the TCV tokamak. The new system allows different tasks to be performed simultaneously. Individual systems can be selected for different tasks. When a task starts, it loads the selected systems which execute their actions. The task then releases each system as soon as it has finished with it. All tasks and systems have their own processes to ensure a parallel treatment. Moreover, each task or system is controlled by two processes, one for navigation in the state machine and the other for executing the corresponding actions. This was implemented to prevent errors in user defined action codes from perturbing the navigation in the state machine.

1. Introduction

The first plasma in the TCV tokamak was produced in 1992. Since then, new ancillary systems such as diagnostics and heating systems have been progressively added and have been incorporated into the TCV plant control resulting in a manifold increase in complexity. The implementation of these systems has revealed some limitations and weaknesses in the scalability of the TCV plant control architecture. Therefore, a major upgrade was recently implemented.

A set of technical requirements for the new plant control was defined. The structure and algorithm for the control of the TCV plant were then designed and the control parameters were defined. Once translated into a computer language, the algorithm was successfully tested by simulating a virtual plant. The move from the existing control sequences to the new ones is being performed gradually, system by system. Many components of the TCV plant are now successfully controlled by this new software.

Section 2 of this paper describes the technical requirements, while the structure of the new plant control software is introduced in section 3. Then, section 4 presents the algorithms used for control. Section 5 contains a discussion of the implementation.

2. Technical requirements

The first control system of TCV was composed of a few separate codes named “sequences”, each of which contained all actions to be performed on a series of different systems as well as the conditions for moving further in the execution of a task such as a plasma discharge, for instance. The structure of the state machine was then implicitly defined within the code. Each new modification required a more and more difficult intervention and the few original sequences controlled more and more systems most of them being independent from each other. Sequences were based on strictly serial execution, which sometimes prevented rapid action on a given system in case of failure, for instance. On the other hand, a system could also simultaneously receive orders from different sequences.

Therefore, the main objectives for the new system were

- clear separation between the state machine definition, the navigation within the state machine and the actions to be performed in the different states

- parallelisation of the control of different systems
- robustness of the whole plant control against programming errors on one particular system
- ability to detect time-outs / errors in systems
- easy implementation of new equipment
- easy implementation of new states in the state machine
- smooth migration from old to new plant control

Several choices made in 1992 for the old system, however, appeared to be excellent, such as the choice of VSystem Databases (a commercial product of Vista Control Systems) as centralised real time databases for all plant parameters. This system allows actions on these parameters from any process. Bridges were written to access the databases from most software used for control around TCV. A GUI also facilitates the communication with the databases. This efficient centralised database, together with its graphical interface, was therefore retained and, as will be shown later, is used to transfer information between processes.

On the other hand, the tool for writing scripts included in the Vsystem package is not very efficient and had limitations. Therefore, the new TCV plant control software is written in the TDI (Tree Data Interface) language which is provided with MDS+ [http://www.mdsplus.org].

Finally, the new system had to be built with existing equipment, to reduce the costs.

3. New plant control structure

The new structure, built to satisfy the points described above, is the following. The whole plant is controlled through two types of objects: *systems* and *tasks*. A *system* corresponds to a piece of equipment such as power supplies, acquisition system, vacuum vessel pumping units or a diagnostic. A *task* corresponds to a series of actions to be performed by a subset of systems. Typically, 'do a plasma discharge' is a *task*.

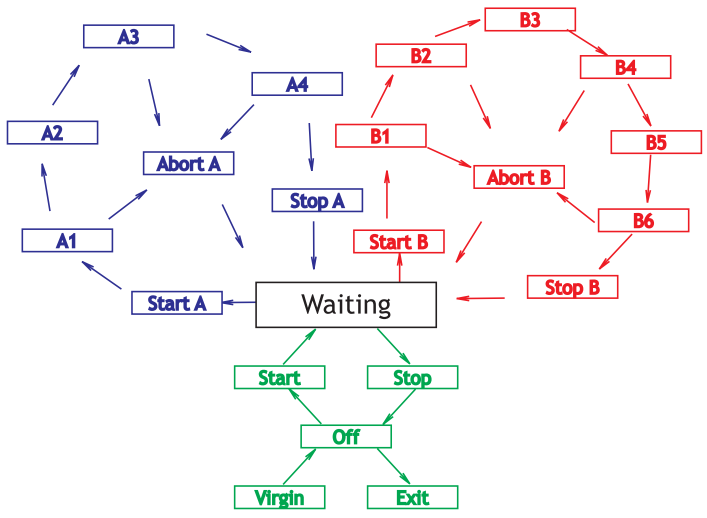


Figure 1: *State machines for tasks A, B and startup.*

A task is described by a state machine, starting from and returning to the state 'waiting'. For instance, 'do a plasma discharge' goes through the states 'preparing discharge', 'loading timing system', 'firing plasma' and 'data acquisition'. Since the TCV plant control has to perform other tasks as well, such as 'start/stop' or 'do glow discharge cleaning', different tasks, with their own state machine description, have to be managed simultaneously, as shown in Fig.1.

Each task requires its own, modifiable, subset of systems. For instance, a particular diagnostic might be required for a plasma discharge. Therefore a dynamic matrix contains the requirements for all systems for all tasks. When starting, a task takes with it all the required systems and all follow the same state machine, sometimes asynchronously but with meeting points. Since systems can be loaded by different tasks, in the first state the task waits until all required systems become available, i.e. released by the other tasks. The last state of a task manages the release of the systems to the 'waiting' state. Hence, different tasks can run simultaneously, as long as they require different systems. If one task requiring a system already working for another task starts, it will wait in the first state until the required system is released by the other task. This structure allows, for instance, doing a cleaning discharge while acquiring data from a plasma discharge.

If a system requires a finer state machine definition, e.g. to match an internal state machine, sub-states can be declared for this system.

Each task and each system is controlled by two distinct processes. One is devoted to the navigation in the state machine and the other to the execution of the actions to be performed in each state.

Each task navigation process runs the same task navigation code. Similarly, there is only one task execution code, one system navigation code and one system execution code. In fact, the codes for tasks or systems only differ by a few lines. Hence, only two codes are used to manage the whole TCV plant control. The distinction between processes dealing with different systems is simply obtained by entering the name of the system as an input parameter. The functionality of these two master codes is described in detail in the following chapter.

Finally, the execution code for tasks and systems calls the action programs corresponding to the relevant *task* or *system* for the current state. The collection of such programs, whose number is approximately *(# tasks + # systems) * # states*, contains all the knowledge necessary to control the different parts of the TCV plant. Each of these programs is produced by the person in charge of the *system* or by the *task* manager. There are no restrictions in the code included in these files except for the constraint that it must return a parameter indicating the status of the action.

If one of these programs contains errors, the execution process catches the error and sends a warning message. If the failure is fatal so that the execution process stops, the disappearance of the process is by an independent supervising process, whose only function is to check the presence of all *task* and *system* processes. The missing process will immediately be signalled to the TCV operator. On the other hand, the navigation process will keep asking the execution process to execute the file: therefore, once the faulty process is repaired the system will fully recover. Finally, testing these files in almost real conditions is possible since the full state machine can be manually run for an individual system. With these characteristics the robustness of the coding is ensured.

In parallel, the navigation process checks whether the execution process spends too much time in executing the specific file. A warning message is generated in that case. Similarly, if one task or one system stays too long in a state, a warning message is addressed to the operator. Since any system in any state of any task can be aborted, the operator might deselect the failing system to continue with the other ones. In that case, the failing system would return to the waiting state via the abort state. More generally, the operator can abort one running task. Here, all systems loaded by the task will be sent to the abort and then waiting state. Finally, all systems have an additional flag indicating whether they are indispensable for a given task. If one system is not necessary and develops a problem blocking a state for too long, it is automatically sent to abort and the task is allowed to continue.

This approach has the advantage that individual programs are compact and, since the state of any system is always well known, the identification of the misbehaving program and its repair or improvement is remarkably easy.

A new system is easily added to the plant control. The new system name merely has to be added to the list of systems. The database is then automatically compiled with the new system. Once this system is duly attributed to the different tasks, it can be loaded. The major work resides in the implementation of the execution files for each state along the state machines of the different tasks.

A new task is also easily implemented. The new task name has to be added to the list of tasks, the state machine has to be described and the actions for the required systems defined.

Finally, a new state in an existing state machine is also easily added. The state characteristics, as well as the modifications of existing states targeting the new one, have to be inserted into the specific state machine description.

4. Description of the major components

4.1 Task & system definition file

The tasks and systems are simply listed in a text file, named tcvpc_structure.txt, under the corresponding title.

4.2 State machine definition files

Each task has its own state machine starting from and returning to the waiting state. The state machine is described in a text file, named tcvpc_*taskname*_states.txt, which comprises a series of blocks, one for each state, with the following syntax:

State	*statename*
Pause	*number*
Timout	*number*
Maxdura	*number*
Cond	*condition*
Dest	*destination statename*

The *pause* number corresponds to the waiting time in seconds between two successive runs of an execution file. *Timout* contains the duration after which the alarm would ring if the execution file has not returned, while *maxdura* contains the maximum duration in the state (before the alarm). The condition, expressed in the TDI language, manages the transition to the next state, indicated under *dest*. Many *cond* / *dest* pairs can be included in the description, if necessary.

4.3 Database

A database is built to transfer commands and parameters between the different processes. This real time database is automatically created from the structure and state files after any modification of any of those files. The main variables used in the processes are:

- Triggers to navigate within a state machine, for the corresponding task and all systems
- Flags for the requisition and necessity, demand and status, for the matrix of tasks and systems
- Strings for the state, status, command, synchronisation, error, … for all systems plus all tasks
- String for the current master, sub-command, … for all systems
- Numbers for time-out, counts, … for all systems plus all tasks

4.4 Navigation and execution programs for tasks and systems

The navigation and execution codes are the two central pieces of the plant control. They are devoted to the management of all systems. All systems run replicas of the same two codes with their name as input parameter. All tasks run very similar versions of the two codes also with the task name as input parameter. The algorithms of the two files described below correspond to the tasks.

4.4.1 Navigation code for tasks

During the initialisation phase, the counters are reset to zero and the state to virgin. The task and system definition file and the state definition file for the corresponding task are read and processed.

The process then enters a loop. It increments the counters and checks for time-outs and excessive time spent in the current state. Then, the process checks whether some required but previously aborted system could be resynchronised. Then it checks the synchronisation of all required systems and calculates a summary status indicating if the task itself is OK and ALL required systems are OK and synchronised. Then, if the execution process is ready to receive a new command, the conditions to move to another state are evaluated. As soon as one condition is found true, the task moves to the new state, changes the name of the state, sets the status to NOTOK and resets some counters. If all conditions are false, the task remains in the same state. The name of the state is then written in the command field, which will be asynchronously read by the execution code to perform the corresponding action. The process then waits for a predefined, state specific, delay.

The process remains in this loop as long as the state is not ‘Exit’.

4.4.2 Navigation code for systems

The navigation code for systems only differs from the description above in a few points. Since the systems can a priori follow all tasks, all state machine files are read. The evaluation of the synchronisation is different: it only compares its state to the state of its current master task. Otherwise, the treatment of the conditions, the eventual move to another state and the assignment of the command are identical.

4.4.3 Execution code for tasks

This program directly starts with a loop. It reads the command field and if a state name is present it set the status to NOTOK and a WORKING flag to ON. Then it looks for a file named ‘tcvpc_*taskname_state*’. Such a file contains the action code for the corresponding task and state. If such file exists, it is executed. When executed, a latched status is updated with the status that the file had to return in one database channel. In the case of an error during execution, a message is displayed on the operator screen. If the file does not exist, the status is automatically set to OK. When all done, the WORKING flag is set to OFF and the command field erased. The process then waits for 1 second and starts another loop. If the command field is empty the process directly goes to the waiting phase. The process remains in this loop as long as the command is not ‘Exit’.

4.4.4 Execution code for systems

This version only differs by the treatment of possible sub-commands. If a *string* is given in the sub-command field the process will search for the file ‘tcvpc_*system_string*’ and execute it as for the main command. In that case, the *string* is defined by the person in charge of the system.

4.5 Action codes

The action code files contain the actions which have to be performed in a given state for a given system or task. The action can be as simple as checking the pressure in the torus or, as complex as igniting a glow cleaning discharge. The only rules are that its execution should be non blocking and that it must return the status.

4.6 Launching of processes

All these processes run on a single computer (ALPHA DS20, 2 processors) operating under VMS. A command file, written in DCL, starts 5 detached processes, one for task navigation, one for task execution, one for system navigation, one for system execution and one for supervision processes. The first 4 processes read the task and system definition file and automatically start all required processes as sub-processes.

4.7 Graphical output

A graphical interface is a convenient tool for the operator to verify the evolution of the processes. VSystem GUIs, as shown in Fig.2, were implemented for this purpose. On the left, the ‘TCV Shot Cycle Control’ window shows the evolution of the 2 main tasks: DOSHOT is in its last phase (Post) and DOGLOW is ready to ignite the discharge (Pret). Below, the state and comment for all tasks (first 3 lines) and systems. The color of the box with the system name indicates the status. For instance the MDS system which deals with data acquisition is red because the acquisition is not yet finished. The DOSHOT task is waiting for the MDS system to return to waiting. All other systems except GLOW are released in the waiting state. GLOW is now loaded by DOGLOW.

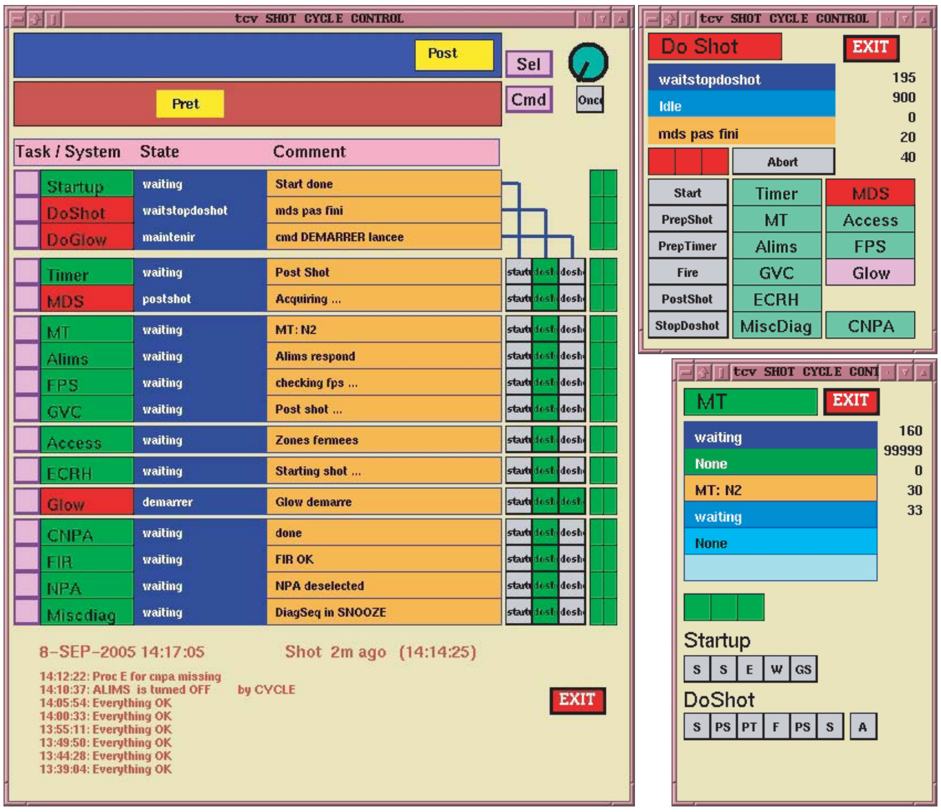
The three columns on the right hand side of the comments show, in green, the systems which are required for the different tasks. The GLOW system is required for both DOGLOW and DOSHOT tasks. The last 2 thin columns show that all processes are running (green): one indicator would turn red in case of failure. The circle with the needle, above these colums, indicates, when turning, that the process which surveys the presence of all processes is running. The text at the bottom of the window contains the warning messages.

The window in the top right corner shows details for the DOSHOT process. The colour of the title corresponds to the status, the 3 boxes below contain the state, the command and the comment. Below on the left, some partial status indicators are now red and, in grey, one sees the buttons to navigate in the state machine. On the right, the numbers correspond to the different counters. Below, in green and red the status of the systems as seen from the task: in full colour the status of the loaded systems, in light colour the status of the other systems. In this case, green means that the system is available, i.e. ready and in waiting state.

The window in the bottom right corner shows a similar view but for a system. The additional boxes show the current master (now on ‘None’ since the MT (motor generator) is in the waiting state), the sub-command (also on ‘None’) and the file currently being executed (empty now since the actions to be performed during the waiting state are rapidly executed and the waiting time between executions is long). At the bottom, two rows of buttons for navigation in the two tasks in which the MT is involved.

5 Discussion – Conclusion

The new TCV plant control was started 8 months ago (Feb 2005) with 2 tasks and a few systems. The transfer of a few systems or one task from the old sequence architecture to this new architecture was successfully performed during short breaks of TCV operation. The new system now controls 3 tasks and 13 systems. It has proven particularly effective in terms of both control efficiency and ease of problem solving.



Task / System	State	Comment
Startup	waiting	Start done
DoShot	waitstopdoshot	mds pas fini
DoGlow	maintenir	cmd DEMARRER lancee
Timer	waiting	Post Shot
MDS	postshot	Acquiring ...
MT	waiting	MT: N2
Alims	waiting	Alims respond
FPS	waiting	checking fps ...
GVC	waiting	Post shot ...
Access	waiting	Zones fermees
ECRH	waiting	Starting shot ...
Glow	demarrer	Glow demarre
CNPA	waiting	done
FIR	waiting	FIR OK
NPA	waiting	NPA deselected
Miscdiag	waiting	DiagSeq in SNOOZE

```
8-SEP-2005 14:17:05          Shot 2m ago (14:14:25)

14:12:22: Proc E for cnpa missing
14:10:37: ALIMS is turned OFF      by CYCLE
14:05:54: Everything OK
14:00:33: Everything OK
13:55:11: Everything OK
13:49:50: Everything OK
13:44:28: Everything OK
13:39:04: Everything OK
```

```
Do Shot
waitstopdoshot     195
Idle               900
mds pas fini         0
                    20
                    40
Timer      MDS
MT         Access
Alims      FPS
GVC        Glow
ECRH
MiscDiag   CNPA
```

```
MT
waiting            160
None             99999
MT: N2               0
waiting             30
None                33
```

***Figure 2**: Graphical interface showing the state and status of different tasks and systems*