CPPM CERN

Guide for ECS FSM design in LHCb detectors

LHCb Technical Note

Issue:	1
Revision:	0
Reference:	EDMS 655828
Created:	19 September 2005
Last modified:	18 November 2005

Prepared by: Pierre-Yves DUVAL

Abstract

This note presents the LHCb experiment control system (ECS) FSM based architecture. It is a guide to specify the FSM that should be implemented in all sub detector control system in order to provide the LHCb global ECS system with a common look and feel and behaviour in all its sub detectors.

The hierarchical control architecture model used is presented together with the various ECS domains identified in order to split the LHCb control system in a limited set of homogeneous areas of control. Then the FSM describing the common standardized states and possible transitions published by all controllers in each of those domains are described. Finally a design process is proposed to support a step by step implementation and test of the sub detector control systems.

Document Status Sheet

1. Document title: Guide for ECS FSM design in LHCb detectors			
2. Document Reference Number: EDMS 655828			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
1	0	19 September 2005	Creation

Method

This document is the report of a collaborative work which has concerned several persons working on the LHCb online software and some LHCb detectors ECS experts during the spring and summer 2005. It has been elaborated after several brain storming and working sessions. This work should go on and new experts contributions are waited when their work on the ECS will come to an operational phase.

We expect updates coming from discussion after the real life experience that will make our knowledge of the subject more complete and profound during the next two years.

The following persons have contributed to the first phase of this work:

CERN: Clara Gaspar, Richard Jacobsson, Beat Jost, Niko Neufeld, Eric Van Erwijnen

CPPM: Pierre-Yves Duval,

INFN: Valerio Bocci, Rafael Nobrega Antunes Nobrega

LAL: Olivier Callot,

Table of Contents

1 Introduction

The LHCb experiment control system (ECS) is in charge of supervising and controlling the experiments components. It should provide the means to start, run and stop the whole experiment or some parts of the experiment in a coherent way. It makes possible to use the detectors in the different modes available.

The overall ECS architecture and design guidelines have been presented in [1] . It is made of a hierarchical organisation of control elements each assuming the management of a reduced set of lower level control elements known has children nodes. Software tools PVSS II and JCOP Framework [2] have been proposed to implement this global ECS system and interface it with the experiments real hardware or software components.

In this document we present the hierarchical control architectures proposed for the LHCb ECS (and its sub systems). We explain how it can be implemented with the Finite State Machines (FSM) tool [5] of the JCOP Framework tool. We propose to split the sub detector ECS into 4 identified control domains to cover the 4 types of activities necessary to manage and supervise to run a sub detector. We specify 4 types of FSM with standardized states and actions to be used for the controllers used in the 4 control domains.

2 General models for the LHCb experiment control system

2.1 Overall architecture

The ECS control system is a hierarchical, reactive system built as a tree of interconnected control nodes (see Figure 1). At the higher levels we have control nodes called *control units* (CU) on which an operator can connect to take the control on the associated sub tree of the system and at the very lower level we have the control nodes called *devices units* (DU) connected to real hardware components they supervise.

It is hierarchical because each control node has only one master node from which it receives commands to execute and a limited set of direct children to which it can send commands. A control node implements the entry point to supervise a sub part of the detector. The root node controls all the detector.

It is reactive because the control nodes behaviour is to react to events. Two types of events exist: a command reception coming from the parent node or a state change in one of the children controllers. Event occurrences trigger two types of control flows. The command flow which propagates along the tree from the operator (or control program) connected to a CU down to the lower level nodes DU acting on hardware to have some real action executed. The state change flow which propagates along the tree from the hardware component that changes its status up to the control nodes CU that will make the necessary reaction to cope with this new configuration of states without the need to forward it up its parent node.

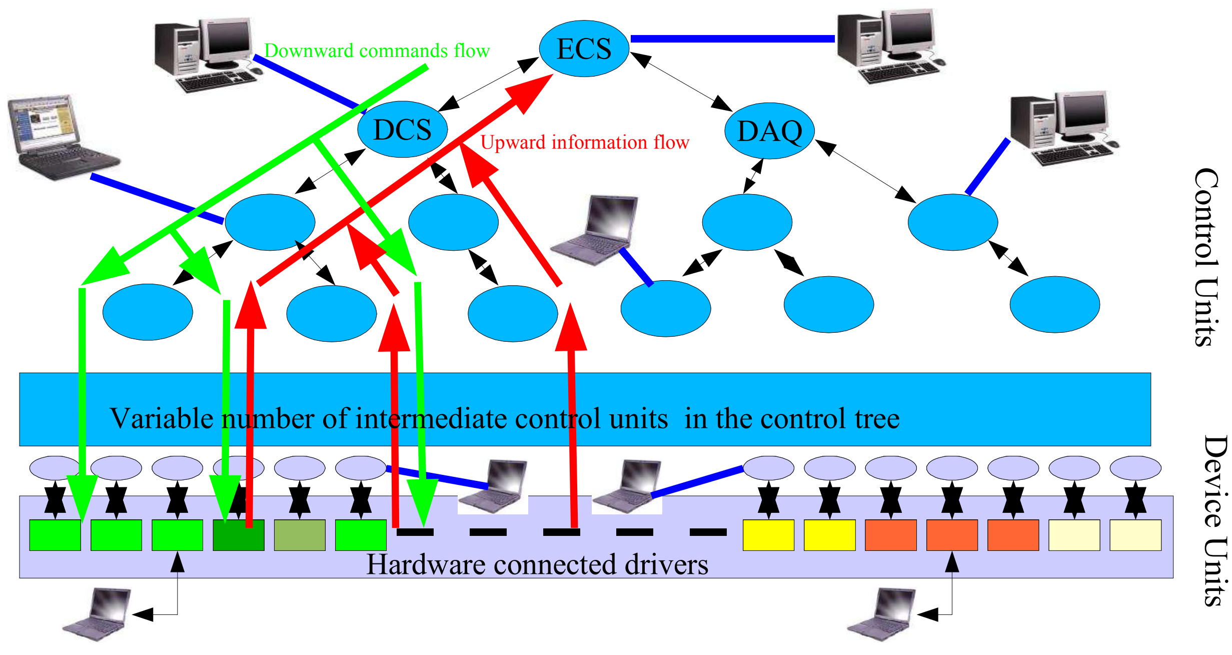
Downward commands flow
ECS
DCS
DAQ
Upward information flow
Control Units
Variable number of intermediate control units in the control tree
Hardware connected drivers
Device Units

Figure 1 Example of a complete control tree

All control nodes publish a unique state representing the state of the sub detector part they control. The way this state is computed is different for CU and DU.

2.2 Control units concept

Control units are controllers located in the higher parts of the control tree. They are the control point to which an operator should connect to take the control and supervise the state of a detector part.

When they receive a command they interpret it and coordinate its forwarding to the lower level nodes whether in parallel or sequential mode. They can translate a command to another one before forwarding it. They can react synchronously to a command in which case they wait for the end of the command propagation and execution by the lower levels in order to compute and publish their new state. They can also react asynchronously and just forward the commands without waiting for the end of its lower levels execution.

When a node changes its state the FSM tool automatically notifies its parent node. When a CU is notified of one of its children state change it recomputes its new state with a rule based script that takes all of its children states as input.

When a CU unit is the parent controlling a large number of children units the state computation decisions can be based upon some majority rules like “95% of children of a certain types” instead of individual testing of each children states. This is useful for example for a muon chamber or a HV board with a lot of channels which we don't want to put in an *ERROR* state if only a few channels are in *ERROR*. Nevertheless this possibility should be carefully used not to definitely mask component failures. The default rule to apply is that all failures should be properly signalled, logged and propagated to the parent.

Each CU is implemented as processes which may overload the control computer. With the FSM tool it is possible to include several CU as threads into one process if only one of them has to be used as operator connection point. This connection point is declared as a CU while the others have to be declared as *logical units* LU associated to this CU.

2.3 Device units concept

Device units are the leaves of the control tree which directly command or react to hardware component change. They encapsulate the PVSS data points connected to the hardware components by the PVSS component driver.

When a DU receives a command it executes a script to write the data point elements values necessary to have the hardware do the requested actions.

When changes occur in the hardware registers or status PVSS automatically changes the corresponding data point elements image which in turns activates a script associated to the DU. This state change script computes the new state of the DU from the data point element values.

2.4 Taking or releasing control on all or part of a sub detector

The users can take the control of a detector part by connecting to its control node. The FSM system guaranties that only one user at a time can use a given part of the detector. When control has been taken at an intermediate level of the control tree it is no more possible for another user to connect to a higher level node to also take control of the same sub part until the first user releases the control node.

2.5 Modifying dynamically the components to control

At any time any part of the detector that has been taken under control by an operator can be excluded from the control tree. This is typically the case when a sub component is out of order and it is makes sense to use the system without it. The control node which manage this component can be temporarily excluded from the control tree waiting for it to be repaired and ready to be used again. The FSM tool provides several modes to exclude and include back any part of the sub-detector control system from the control tree

2.6 Alternate control trees in the same control system

While one and only one hierarchical control tree is meant to take the control of a detector it may be useful to have other views of some parts of the sub detector components. The FSM tool provides the possibility to link nodes already part of the main control tree under an alternate control tree providing a root tree which is not part of the main control tree. This alternate control view of a set of components based upon a different segmentation of the detector gives a way to control a sub part of the detector as an autonomous sub detector with all the necessary components to run stand alone.

An application of this feature is the possibility to take data with only a sector of a detector.

3 The ECS domains and FSM

3.1 Identified control domains for a detector

It has been found useful to split the LHCb detectors ECS systems into 4 different domains which encompass 4 different types of functions necessary to run a detector. Those domains are:

DAQ and trigger domain

This domain include all the components which purpose if to acquire and filter the data. The components are the front end boards, trigger boards and farm programs and data buffers like the TELL1 boards.

DAQ infrastructure

This domain include all the components that are providing power, network and computing resources necessary for the data acquisition system. We found in it the FE, trigger and TELL1 boards crate power, the communication networks for control or data transportation.

Detector infrastructure

This domain includes all the components related to what is also named slow control services. This include monitoring status of sensors related to water leak, pressure, temperature, magnetic fields, internal or external to the experiment safety devices. It also contains the supervision of systems like: gas system, cooling systems, electrical power supply systems.

HV domain

this domain contains all the components that provide the high voltage power supply to the detector instruments and electronic.

The control of each of the detectors of the LHCb experiment should be split into those 4 separate domains each of which providing its root CU node. This will ease the further integration of all the detectors ECS into the LHCb overall ECS system.

In order to simplify the design of the ECS system and to exhibit a common look and feel to the shift operators a common FSM scheme with common standardized states and transitions has been defined to implement in all the CU nodes within a given domain. The same states and actions are also recommended for the DUs in the same domain whenever possible. The following paragraphs describe the FSM proper to the CU of the 4 ECS domains.

3.2 DAQ and trigger domain FSM

The FSM to be implemented in controllers for the front end and data acquisition and hardware trigger electronic is summarised at Figure 2.

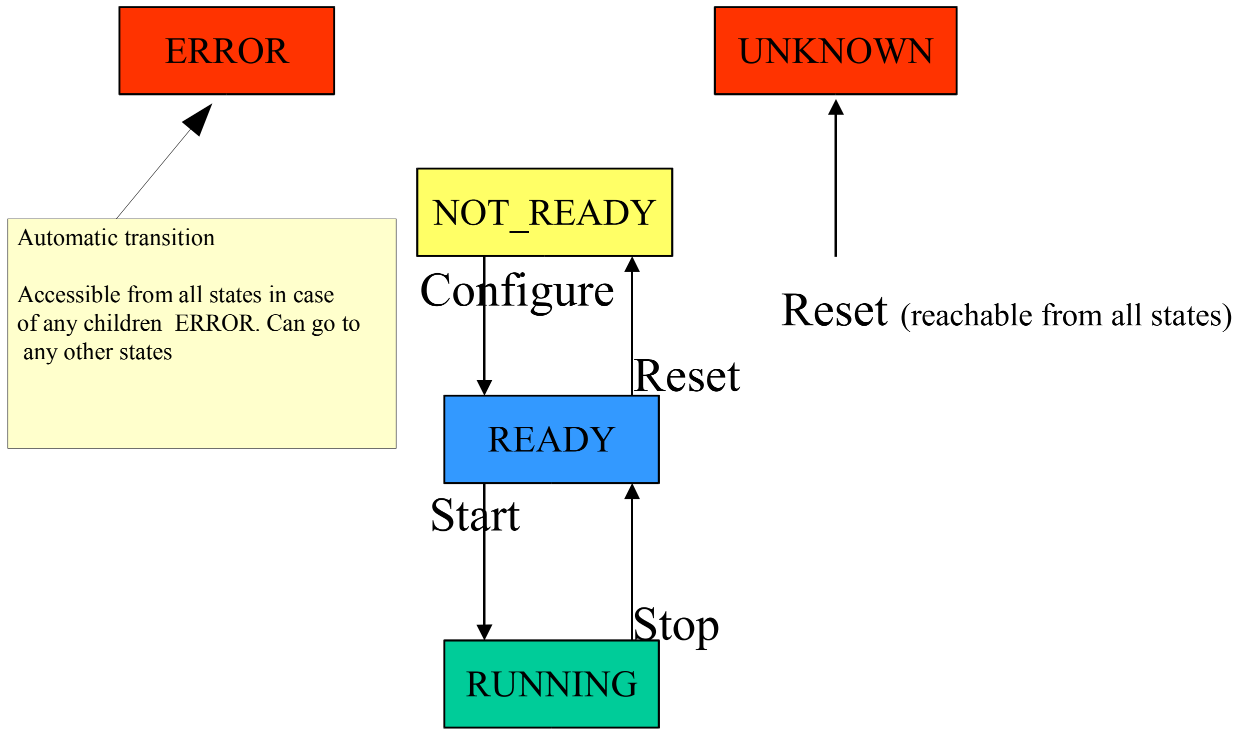
ERROR
UNKNOWN
NOT_READY
Automatic transition
Accessible from all states in case of any children ERROR. Can go to any other states
Configure
Reset (reachable from all states)
Reset
READY
Start
Stop
RUNNING

Figure 2 DAQ and trigger FSM states and transitions

State	Semantic
UNKNOWN	It is not possible to communicate with the component. We have no information about it and we can't apply any command to it.
NOT_READY	The component is under the control of the ECS but it needs a heavy configuration before it can be used.
READY	The component is under the control of the ECS but it needs a light configuration before it can be used.
RUNNING	The component is fully configured and taking data. Its configuration can't be changed while in this state. It should be stopped if we need to change is configuration.
ERROR	At least one of the component children nodes has published an ERROR state which means that it is not able to take valid data.

Table 1 DAQ and trigger FSM states semantic

Before it can be used a DAQ and trigger component should be configured with the proper setting and parameters. We have defined two types of configuration actions. One is called *heavy configuration* activated by the *Configure* command. One specification is that it is an operation that takes more than a couple of seconds with operation like the access to a central external database or whatever operation that takes time. Another specification is that it implies a major change in the way the component will work for instance when changing its mode from normal to calibration mode. The other configuration is called the *light configuration* activated by the *Start* command which is short to execute like just copying some new values from some memory or PVSS local data points elements in hardware registers. It can be executed at each start of run without a perceptible loss of efficiency. The commands *Configure, Start, Stop , Reset* are used to manage those configuration issues. The *Start* one is the one to be applied at a start of run. In order to make possible to set the system children in an homogeneous state after an error recovery the *Stop* and *Reset* commands can be accepted and forwarded to all children in any of the states.

Some transitions are automatic. The *ERROR* one is a consequence of any children nodes state change to publish the *ERROR* state. The *UNKNOWN* one is encountered at the very system start up procedure when the control system is not yet ready to fulfil its functions. It can also appears after the set up of the system when a loss of communication with the control system occurs. This may for example be the consequence of a hardware reset of the controlled board or the reinitialization of the local checking and control software.

Transitions from any of those two *ERROR* or *UNKNOWN* states can go to any of the other states as described by the following rules evaluated in the given following order.

If any child in state ERROR move to ERROR

Else if any child in state UNKNOWN move to UNKNOWN

Else if any child in state NOT_READY move to NOT_READY

Else if any child in state READY move to READY

Else move to RUNNING

If the transition NOT_READY to READY to configure the system is long (more than a few seconds) an intermediate CONFIGURING state shoul be published (see Figure 3) instead of NOT_READY until the transition is finished. The control node will automatically transit out of this CONFIGURING state upon occurrence of the same conditions as the NOT_READY state. Time-out should be implemented to publish an ERROR state in case the transition is too long (see APPENDIX 1: Time-out management with the FSM tool).

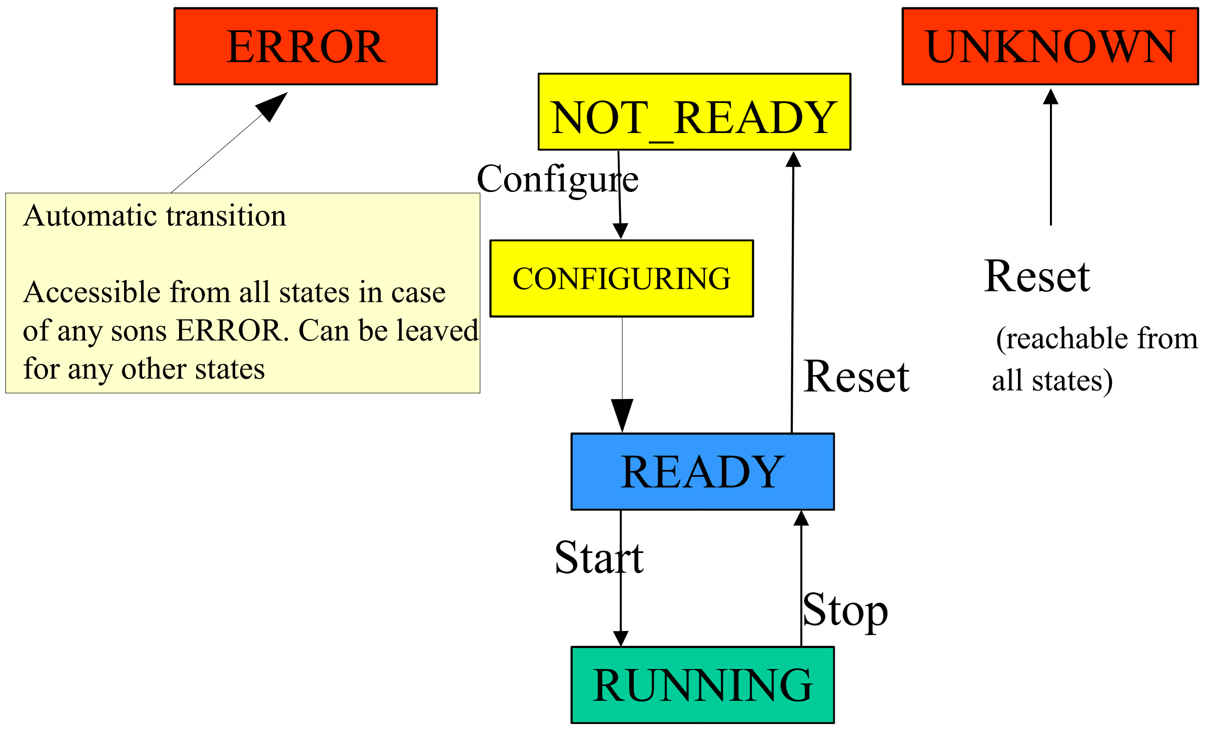
ERROR
UNKNOWN
NOT_READY
Configure
CONFIGURING
Automatic transition
Accessible from all states in case of any sons ERROR. Can be leaved for any other states
Reset
(reachable from all states)
Reset
READY
Start
Stop
RUNNING

Figure 3 DAQ and trigger FSM states and transitions with long configuration

3.3 DAQ infrastructure domain

The FSM to be implemented in controllers for the front end and data acquisition and hardware trigger electronic is summarised in Figure 4.

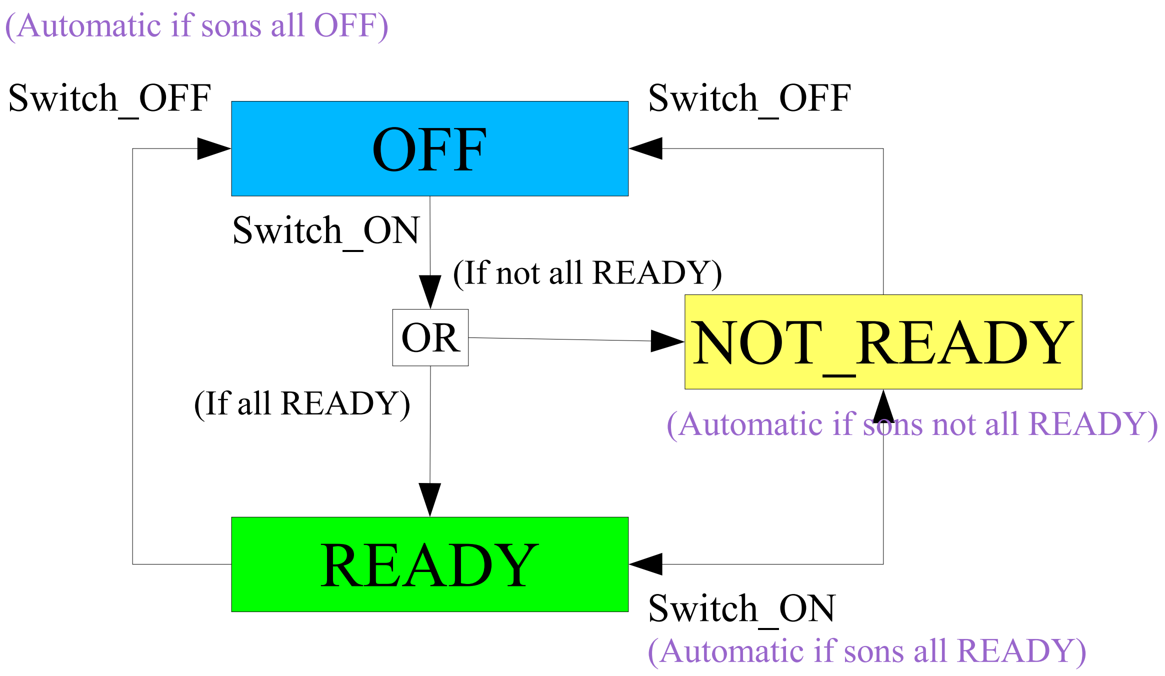
(Automatic if sons all OFF)
Switch_OFF
OFF
Switch_OFF
Switch_ON
(If not all READY)
OR
NOT_READY
(If all READY)
(Automatic if sons not all READY)
READY
Switch_ON
(Automatic if sons all READY)

Figure 4 The DAQ infrastructure FSM

State	Semantic
OFF	No children components are powered or usable
NOT_READY	Only some children components are READY (in DU can signal an error)
READY	All children components are READY

The DAQ infrastructure components publish a state that simply shows if they are READY to be used or not. The reason why they may not be READY is not part of the FSM. This is information is available at the lower level DU where the ECS component specific display panel should provide the necessary information.

3.4 The detector infrastructure FSM

This domain encapsulate a large variety of components with very different behaviour and some of which are simply monitoring devices upon which the ECS has no control. The FSM is the same as for the DAQ infrastructure.

State	Semantic
OFF	Children components are not powered (or the monitoring process is not running)
NOT_READY	Children components are not completely READY (the component parameters have not the correct usage value (alarm end error values are included there))
READY	All children components are READY (the component parameters have the correct usage values)

Like for the DAQ infrastructure the minimum of information is provided there just to know if the system is operational or not. Detailed information about monitored values or why a component is considered NOT_READY should be available at the component specific panel associated to the DU FSM state publication and in the error logging panel.

3.5 HV domain FSM

The HV FSM has been designed in order to give the possibility for the component to have several reference HV values. The main one is the usage value which is the higher one. But in some situations it may be useful not to maintain permanently this usage value in order to protect the detector or avoid some tripping. The raising of HV is a time consuming operation necessary to minimize. The idea is thus to maintain the HV value at a lower standby level when the detector is not used in order to protect it while limiting the time necessary to move from this low standby value to the usage value when needed.

For readability concern the FSM is not represented in Figure 5 with transitions because we can move from any state to any other state.

The “RAMPING_XX” states have been introduced because it is anticipated that those transitions will take time. Thus after any “Go_XX” command the node should be switched into the corresponding “RAMPING_XX” state until a children states changes move it automatically into another state. Any command ends up with the DUs publishing a known state or a time-out "ERROR" if a stable defined voltage level is not reached within a defined delay. When in "RAMPING_XX" state and children states changes are notified, the HV CU computes its possibly new state by the following rules:

If any child in state ERROR move to ERROR

Else if all children in state STANDBY_1 move to STANDBY_1

Else if all children in state STANDBY_2 move to STANDBY_2

Else if all children in state READY move to READY

Else if all children in state OFF move to OFF

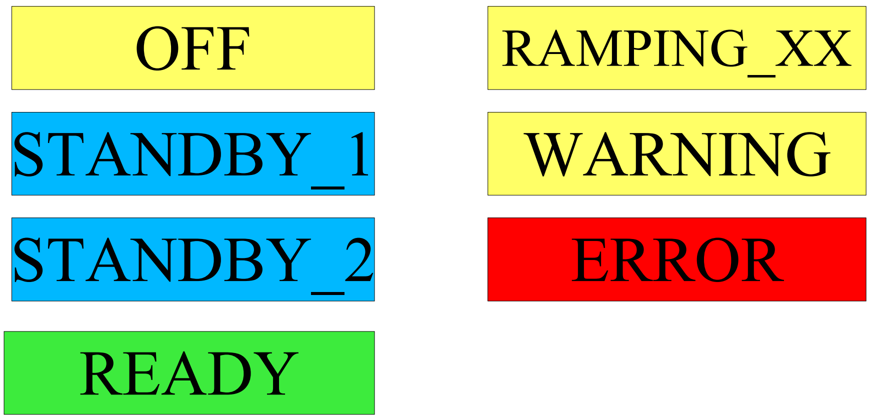

Figure 5 The HV FSM states

State	Semantic
OFF	No power at all or value 0
STANDBY_1	Intermediate low voltage value
STANDBY_2	Intermediate high voltage value
READY	Usage high voltage value
RAMPING_XX	At least one children is moving towards a new value after the "GO_XX" command
WARNING	Children are not in the same state
ERROR	Severe fatal error signalled by one or a majority of children.

Available commands are available from any state to go to any other state.

Commands	Target states
Go_READY	READY
Go_STANDBY1	STANDBY_1
Go_STANDBY2	STANDBY_2
Go_OFF	OFF

The various possibility for the "RAMPING_XX" states are:

Transitional state	After command
RAMPING_READY	Go_READY
RAMPING_STANDBY1	Go_STANDBY1
RAMPING_STANDBY2	Go_STANDBY2
RAMPING_STANDBY2	Go_OFF

3.6 Nodes naming convention

The name of each node should be significant and clearly shows what detector subsystem and domain it is part of.

The domains identification characters are shown in Table 2.

domain	Identification string
DAQ and trigger domain	DAQ
Data acquisition infrastructure	DAQI
Detector infrastructure	DCS
High voltage	HV

Table 2 The ECS domains identification strings

The subsystems names are the same as the one defined for the identification in the configuration database as listed in Table 3.

Subsystem name	**prefix**
Vertex locator	VELO_
Pile-Up	PUS_
RICH1	RICH1_
RICH2	RICH2_
Outer tracker	OT_
Inner tracker	IT_
Trigger Tracker	TT_
ECAL	ECAL_
HCAL	HCAL_
PreShower	PRS_
SPD	SPD_
MUON	MUON_
L0 Muon	L0MUON_
L0 Calo	L0CALO_
L0 Decision Unit	L0DU_
TFC	TFC_
DAQ	DAQ_

Table 3 Prefixes for subsystems

For example the L0 MUON trigger system is composed of 4 crates each covering a quarter of the detector. Using this convention the Control Unit controlling this crate could be named L0MUON_DAQ_Q1 and the Device Unit controlling the electric power of this crate L0MUON_DAQI_Q1.

4 Guidelines to design a sub-detector ECS system

In this part we make the assumption that designers know PVSS and PVSS concepts [2]. Some courses are available at CERN [3] in order to learn PVSS and the associated JCOP Framework which contains the FSM tool [4] .

It is highly recommended to follow the PVSS and FSM JCOP framework courses in order to get a full understanding of the ways they can be used.

4.1 Component object modelling and PVSS integration

PVSS software is organized around a dynamic database which objects are named data points which are structures composed of data point elements. All objects of the hardware real world that should be monitored by the ECS should have an associated image data point in the PVSS data base which is kept up to date with the real hardware one. The first thing to do is a breakdown structure of all the objects controlled by ECS till the lower level objects. Then the corresponding data points types should be created and a data point of this type instantiated for each object.

Once this is done a PVSS application should be written to:

- Connect those data points to the real hardware available informations (registers) to have them updated when the real hardware changes.
- Provide command data points to send write requests to hardware.
- Develop scripts to recompute states or any application data points values when the hardware connected data points values change.
- Create display panels to show the data point information and to send commands to the hardware.

Care should be taken that any object that should be controlled by a DU should have an associated data point and not a simple data point element or be the combination of several data points.

The implementation depends of the type of driver used: CAN (EMLB) or DIM (CCPC and SPECs) based, but the logic doesn't change.

4.2 Object integration in the FSM tool by domains

The next step is to create a DU of a type that exhibits the same states as the ones specified for the FSM of the domain the components belongs to. The DU is associated to the target data point in the FSM tool and the PVSS monitoring panel associated to this data point can be associated to the DU as component specific panel.

The designer has to write the scripts that:

- computes the DU states out the PVSS data point elements current values
- writes the proper values in the PVSS data point elements in order to act upon the hardware when a command is received by the DU

The FSM tool provides code generation features that help in the writing of such scripts.

Then this DU has to be integrated under a CU because a DU can't be used alone. For this a CU of the type that is associated to the FSM of the domain the component belongs to is instantiated. The DU can then be declared as the child of this CU.

At this point we have a simple control tree that can be used to control the correct behaviour of the component and its DU. It is possible now to build the rest of the control tree adding other DUs and the various levels of CUs in the tree.

4.3 Designing the detector control trees

4.3.1 Trees breakdown logic

Each detector should provide 4 independent control trees (if applicable) for the 4 control domains.

The breakdown of those trees may follow different types of logic.

The **physical aggregation logic** based upon the physical dependencies between the component is a natural one. For example we can have channels that are implemented by boards which themselves are in crates inside racks for a full detector. Such a logic seems the most suitable when we deal with infrastructure components (like power supply control).

The **logical aggregation logic** based upon the detector area segmentation can also be used. For example for the muon detector we can have the concept of pads included into regions themselves included into quarters which themselves are part of the detector.

The decision to favour one point of view against another should be taken according to the functional system structure and the way the detector has to be used.

The FSM tool makes possible to build several control trees using the same DUs [4] . Thus the ECS designer can create as many trees as they need in the system during their debugging and commissioning phase. This will provide real life experience which we expect will provide the necessary understanding to choose the most suitable one for the final detector integration in LHCb. Alternate ones could be kept to use for special purposes (expert debugging).

Finally we will have to understand not only the main normal data taking scenario but also special modes for calibration and testing of the detector. Those modes may raise special requirements on the control architecture which we are not able to anticipate now.

4.3.2 System scaling

The number of PCs needed to run the control system may depend on hardware constraints or software load.

- The SPECs and EMLB needs special network interfaces which number are limited by the number of PCI slots in the PC, the type of communication board used and how many nodes are linked on the control network.
- The FSM tree control units are implemented has separate processes. Some optimization may be used by implementing some as logical units associated to a control unit. Nevertheless it is necessary to have an idea on the number of needed nodes to plan their distribution on the various detector control PCs.

It is useful to have quickly some figures about the number of controllers in order to assess the hardware needs. Figures available show that 500 DUs of HV channel type can be controlled by one CU with a transition time of 5 seconds from state to state on a 256 MB PC [5] .

References:

[1] LHCb Online System Data TDR7, CERN-LHCC-2001-040, December 2001

[2] PVSS page at CERN:

http://itcobe.web.cern.ch/itcobe/Services/Pvss/

[3] PVSS and JCOP (FSM) courses at CERN:

http://itcobe.web.cern.ch/itcobe/Services/Pvss/Training/welcome.html

[4] FSM tutorials

http://clara.home.cern.ch/clara/fw/FSMConfig.pdf

[5] Architecture FSM and SMI language presentation

http://itcobe.web.cern.ch/itcobe/Services/Pvss/Training/PVSSJCOPFwCourse/welcome.html get document *Finite State Machine (pdf)*

[6] Conventions and use of colours in PVSS (ECS) developments

http://itcobe.web.cern.ch/itcobe/Services/Pvss/Training/PVSSJCOPFwCourse/welcome.html get document *JCOP Framework Guidelines and Conventions (pdf)*

APPENDIX 1: Time-out management with the FSM tool

1- How to program a time-out in a DU

A time-out trigger can be inserted into DU in order to limit the time to switch from one state to another with the *fwDU_startTimeout()* library function:

http://clara.home.cern.ch/clara/fw/FwFSMv24r1/fwDU/index.html

With this function it is possible to program the switch of a DU into the "ERROR" state in case a target state or any new state change does not occurs within a given delay. This function has to be inserted inside the "Action" script function of the DU associated to some command.

The following example shows how to implement a time-out 10 seconds to have the DU move to the "ERROR" state if it didn't reach the "ON" state within 10 sec , after the command "SWITCH_ON" was issued.

```
if (command == "SWITCH_ON")
{
      /* set the dpelement value to request the action to the component driver */
       dpSet(device+".flag",1);
      /* fwDU_startTimeout(delay,domain,device,errorState,targetState); */
        fwDU_startTimeout(10,domain,device,"ERROR",”ON”);
}
```

The targetState is optional. If not specified any state change will stop the timer.

The propagation of the ERROR state to the higher level CU will notify the transition failure at all levels in the tree.

2- How to implement a transitional state in a CU with a time-out on the transition duration

A transitional state is a state into which a CU switches as a response to a command which starts a long transition. The CU remains in this transitional state as long as the transition last.

The CU will automatically transit out of this state when all the children will publish the target state showing that they have all finished the transition. In case one of the children publishes a new state different from the target state the CU "When List" rules define the state the CU has to switch into. The general rule is that the CU will move to "ERROR" if any of the children publishes the "ERROR" state.

Thus the implementation of time-out into the low level DU as explained in paragraph 1 above is enough to have time-out management on any command send in one of the higher nodes in the control tree.

3- How to implement a timer DU dedicated to time-out management

If it is not easy or possible to implement a time-out function into the children DU associated to the experiment devices it is still possible to develop a simple timer DU which will only be used as a delay counter. This timer DU is connected as a direct child of the CU that need a time-out security. This timer only needs the following set of commands and states.

State	meaning
OFF	Timer is not counting
ON	Timer is counting
ERROR	Timer has reached the time-out delay

Table 4 Time-out timer states

Command	Action
START	Start the timer with the fwDU_startTransition() library function
RESET	Put the timer in the OFF state stopping the counting process or erasing the ERROR state.

Table 5 Time-out timer commands

This timer DU can be connected at any CU in the control tree and used by it as a time-out trigger. The master CU starts the timer with the START command when it receives a command to execute a delay guarded transition. It should RESET the timer when it is notified of the transition end. If the transition and is not notified within the proper delay the timer switches into the "ERROR" state which propagated to the master CU will end the transition by switching it also into the

"ERROR" state.

A simple PVSS data point with one integer flag is associated to the timer. The flag value gives the timer state. In the following example flag values are OFF=0, ON=1. It is not necessary to associate a flag value to the "ERROR" state. The following example of code is entered with the "Configure Device" button for Initialization, States or Actions of the Device Editor Navigator of the FSM tool.

```
Timer_initialize(string domain, string device){
dpSet(device+".flag",0);
}
Timer_valueChanged( string domain, string device,
    int flag, string &fwState )
{
        if (flag == 0)
        {
                fwState = "OFF";
        }
        else if (flag == 1)
        {
                fwState = "ON";
        }
        else
        {
                fwState = "ERROR";
        }
}
Timer_doCommand(string domain, string device, string command)
{
        if (command == "START")
        {
                dpSet(device+".flag",1);
               /* 10 sec to put it back to OFF */
                fwDU_startTimeout(10, domain, device, "ERROR","OFF");
        }
        if (command == "RESET")
        {
                dpSet(device+".flag",0);
        }
}
```