OOWG Tutorial
The Design of the VLT
Telescope Control Software

G.Chiozzi

12/06/96

Contents

- Introduction
- TCS General architecture
- Hierarchy of TCS WS modules
- Common architecture of a WS TCS module
- Main loop handling. Error recovery
- Failure handling
- Modular testing
- Reference documents
- Conclusion

INTRODUCTION

The Telescope Control Software for the VLT (and for the NTT Upgrade) has been designed and implemented using OO methodologies.

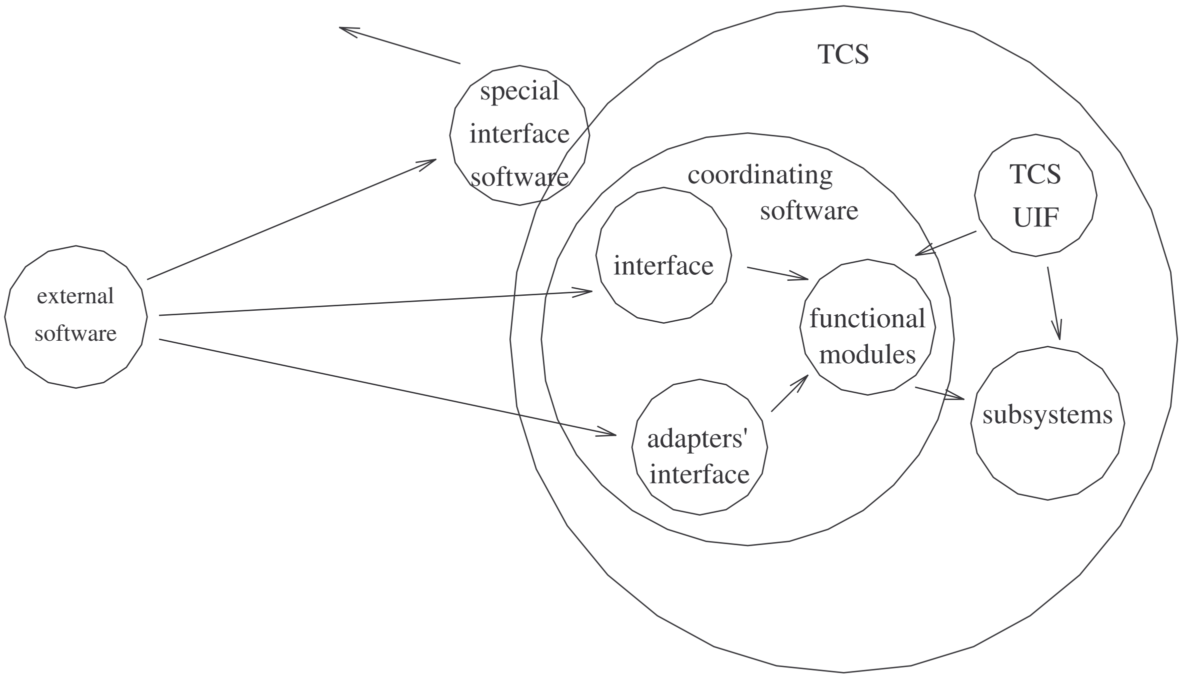
The workstation components are implemented in C++ and are based on the EVH toolkit.

Purpose of this presentation is:

- To analyse the general design architecture of the software
- To show common design patterns that can be reused elsewhere
- To examine specific details of the implementation

The presentation is focused on technical and implementation issues and not on functional specifications

TCS General architecture



TCS General architecture (cont.)

- **The TCS is subdivided in many modules, identified by the functional responsibility. A module consist of:**
  - One/more processes
  - Database
  - Command interface

- **Every module runs on workstation and/or LCU platform.**

- **The tif module provides a single interface for external users. Such users do not see the module's structure and interact only with tif.**

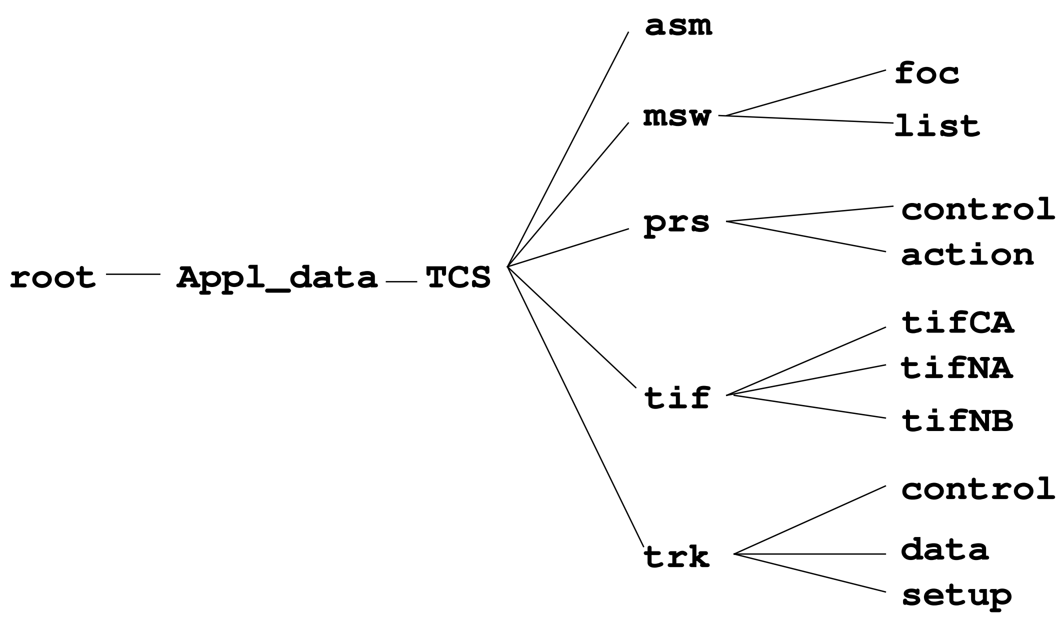
- **The functional modules (msw, prs, tkrws,...) provide coordination features on the workstation side.**

- **The subsystem modules (trk, ag,...) provide real-time functionalities on the LCUs**

TCS Online Database

The TCS workstation contains the TCS Online database.

- The whole TCS database lies under the Appl_data:TCS point.
- Every functional module correspond to a branch in the database
- Every subsystem has its own LCU database and a scan image on the TCS WS database



Hierarchy of TCS WS modules

TCS module's are as much as possible independent,
so that it is possible
to develop and test them one by one

A few unavoidable dependencies are due to the fact that some modules need services provided by other ones. No circular dependencies between modules are allowed.

Common services are provided through:

- Libraries
- Database classes

This define the hierarchy between modules.

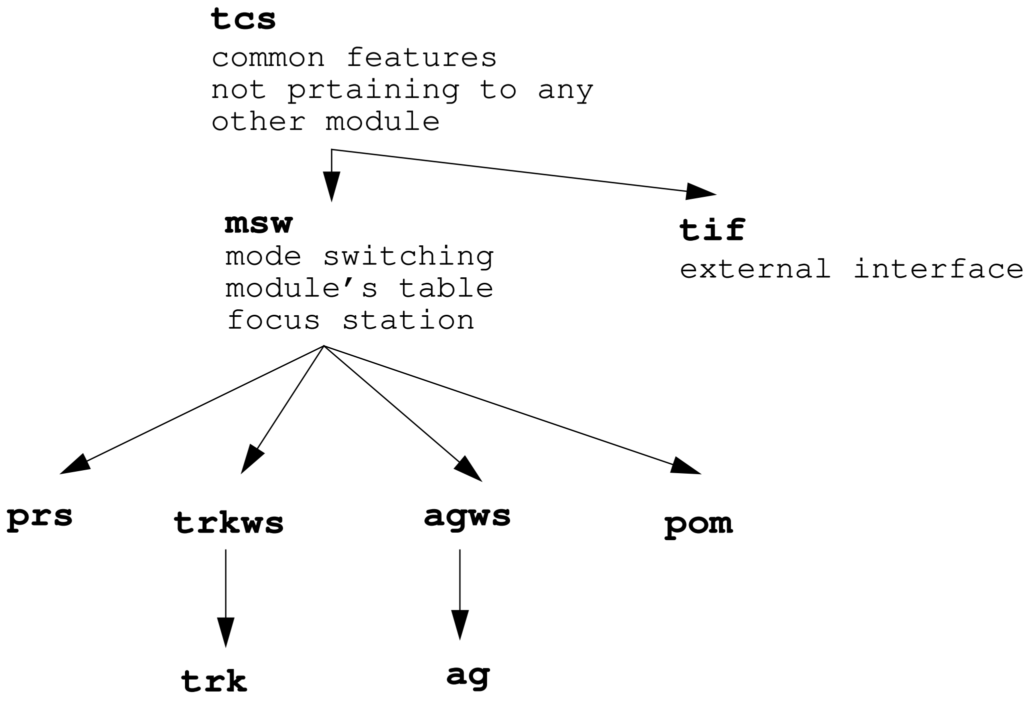
The online database is used to have "configurable" any dependency between modules. For example:

- process names for commands
- address of external db branches

are always stored in the database.

Hierarchy of TCS modules (cont.)

Current TCS modules:



Integration modules: **tcssim**

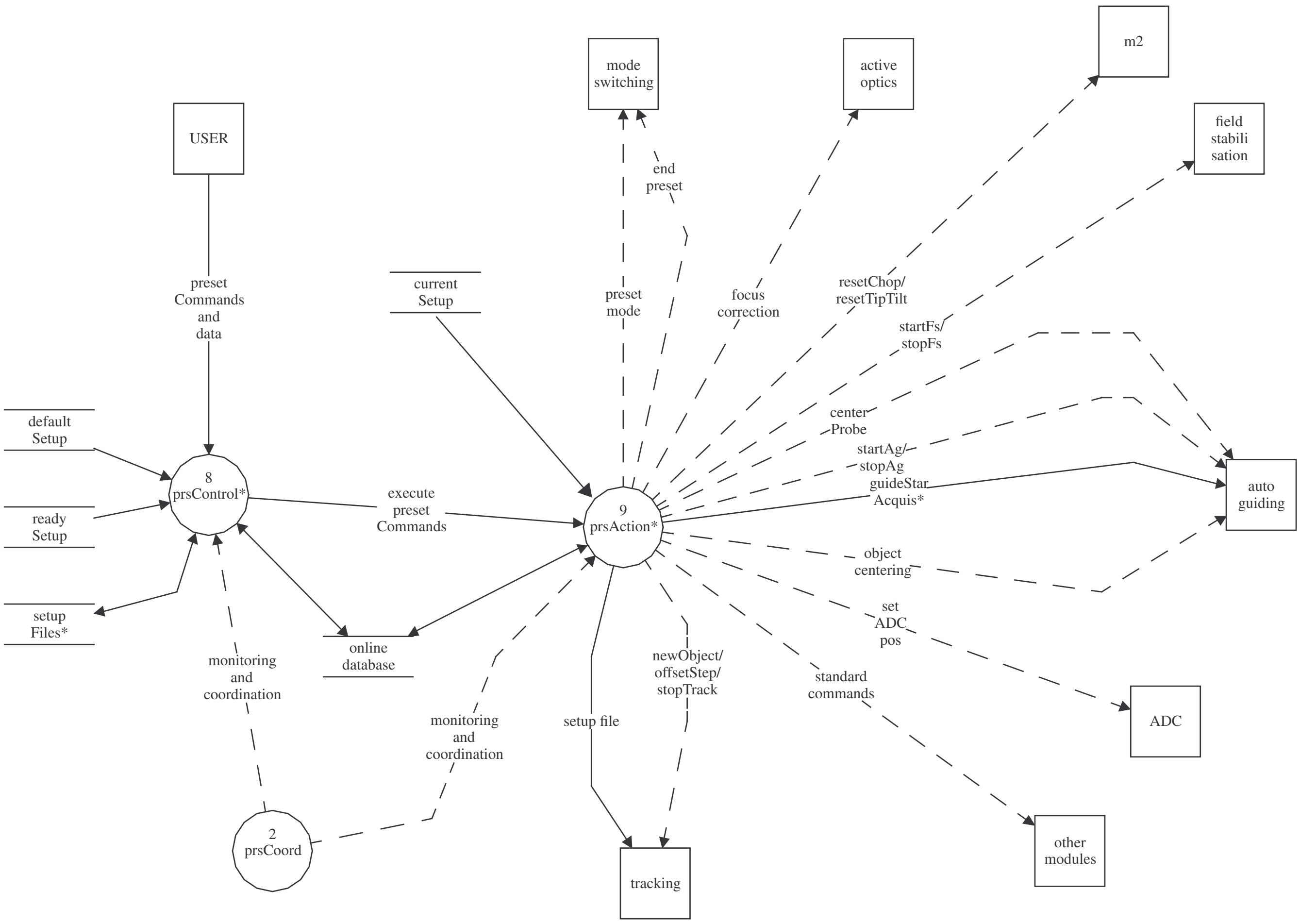
tcsint

Common architecture of a WS TCS module

All workstation modules have the same architecture

- A module is implemented as one or more processes
- One of the processes (called **control**) provides the only interface to other modules and coordinates the other internal processes.
- A module defines a single database branch.
- The root point of the branch provide attributes for:
  - state
  - substate
- Every process has its own private sub-branch

High level DFD for the prs module



The prs global state

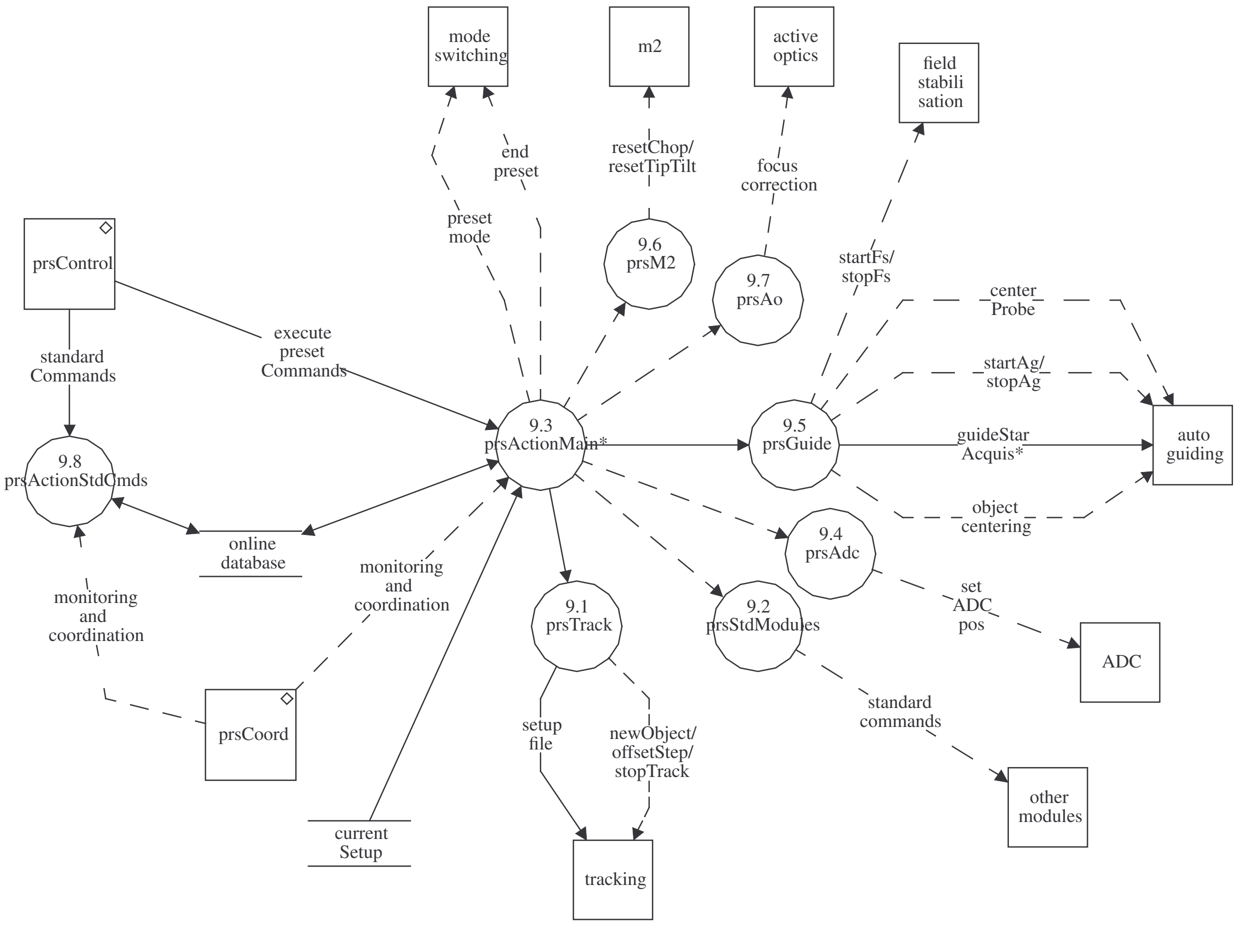
Table 1 - TCS Preset Global State

Control Action	OFF	LOADED/ STAND-BY	READY	PRESETTING	STOPPING	ERROR	UNKNOWN
OFF	OFF IDLE	OFF IDLE	OFF IDLE	OFF IDLE	OFF IDLE	OFF ERROR	UNKNOWN UNKNOWN
LOADED/ STAND-BY	OFF IDLE	LOAD/SB IDLE	LOAD/SB IDLE	LOAD/SB IDLE	LOAD/SB IDLE	LOAD/SB ERROR	UNKNOWN UNKNOWN
READY	OFF IDLE	LOAD/SB IDLE	ON-LINE	ON-LINE	ON-LINE	ON-LINE ERROR	UNKNOWN UNKNOWN
PRESETTING	OFF IDLE	LOAD/SB IDLE	ON-LINE PRESETTING	ON-LINE PRESETTING	ON-LINE PRESETTING	PRESETTING ERROR	UNKNOWN UNKNOWN
STOPPING	OFF IDLE	LOAD/SB IDLE	ON-LINE STOPPING	ON-LINE STOPPING	ON-LINE STOPPING	STOPPING ERROR	UNKNOWN UNKNOWN
ERROR	OFF ERROR	LOAD/SB ERROR	ON-LINE ERROR	PRESETTING ERROR	STOPPING ERROR	prev state ERROR	UNKNOWN UNKNOWN
UNKNOWN	UNKNOWN UNKNOWN	UNKNOWN UNKNOWN	UNKNOWN UNKNOWN	UNKNOWN UNKNOWN	UNKNOWN UNKNOWN	UNKNOWN UNKNOWN	UNKNOWN UNKNOWN

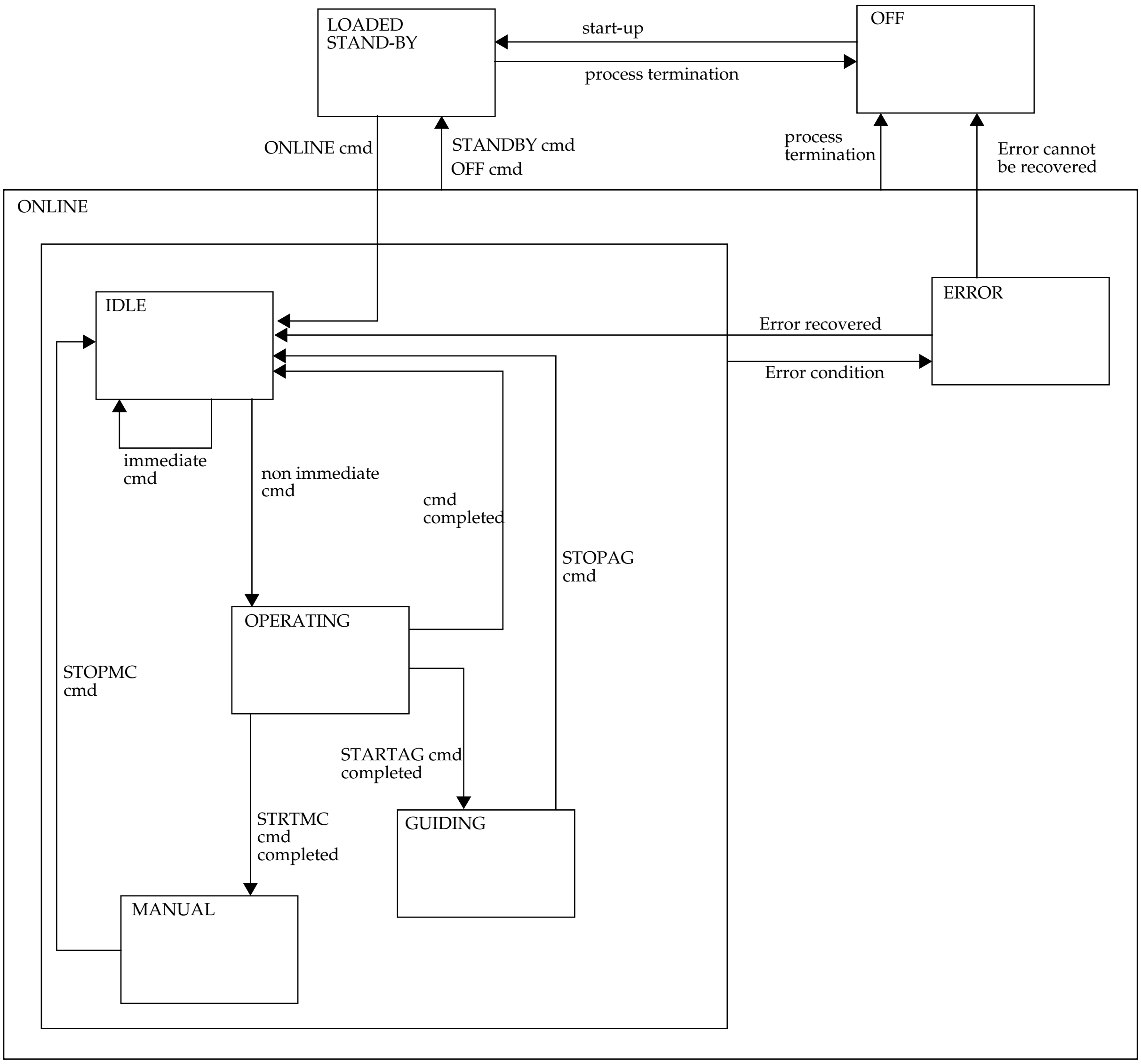
The database calculation engine formula:

```
ATTRIBUTE int32            state      prsSTATE_OFF
    BEGIN
      Definition                                                                  \
       IF([<relative>control.state]=evhSTATE_UNK        OR                        \
          [<relative>action.state] =evhSTATE_UNK,        evhSTATE_UNK,            \
       IF([<relative>control.state]=prsSTATE_OFF        OR                        \
          [<relative>action.state] =prsSTATE_OFF,        prsSTATE_OFF,            \
       IF([<relative>control.state]=prsSTATE_LOADED     OR                        \
          [<relative>action.state] =prsSTATE_LOADED,     prsSTATE_LOADED,         \
       IF([<relative>control.state]=prsSTATE_STANDBY    OR                        \
          [<relative>action.state] =prsSTATE_STANDBY,    prsSTATE_STANDBY,        \
       prsSTATE_ONLINE ))))
    END
   ATTRIBUTE int32            substate   prsSTATE_STEADY
    BEGIN
      Definition                                                                  \
       IF([<relative>control.state]=evhSTATE_UNK        OR                        \
          [<relative>action.state] =evhSTATE_UNK,        evhSTATE_UNK,            \
       IF([<relative>control.state]=prsSTATE_OFF        OR                        \
          [<relative>action.state] =prsSTATE_OFF,        prsSTATE_STEADY,         \
       IF([<relative>control.state]=prsSTATE_LOADED     OR                        \
          [<relative>action.state] =prsSTATE_LOADED,     prsSTATE_STEADY,         \
       IF([<relative>control.state]=prsSTATE_STANDBY    OR                        \
          [<relative>action.state] =prsSTATE_STANDBY,    prsSTATE_STEADY,         \
       IF([<relative>control.state]=prsSTATE_ERROR      OR                        \
          [<relative>action.state] =prsSTATE_ERROR,      prsSTATE_ERROR ,         \
       IF([<relative>control.state]=prsSTATE_STOPPING,   prsSTATE_STOPPING,       \
       IF([<relative>control.state]=prsSTATE_PRESETTING, prsSTATE_PRESETTING,     \
       prsSTATE_READY )))))))
     END
    ATTRIBUTE prsCONTROL       control
    ATTRIBUTE prsACTION        action
END
```

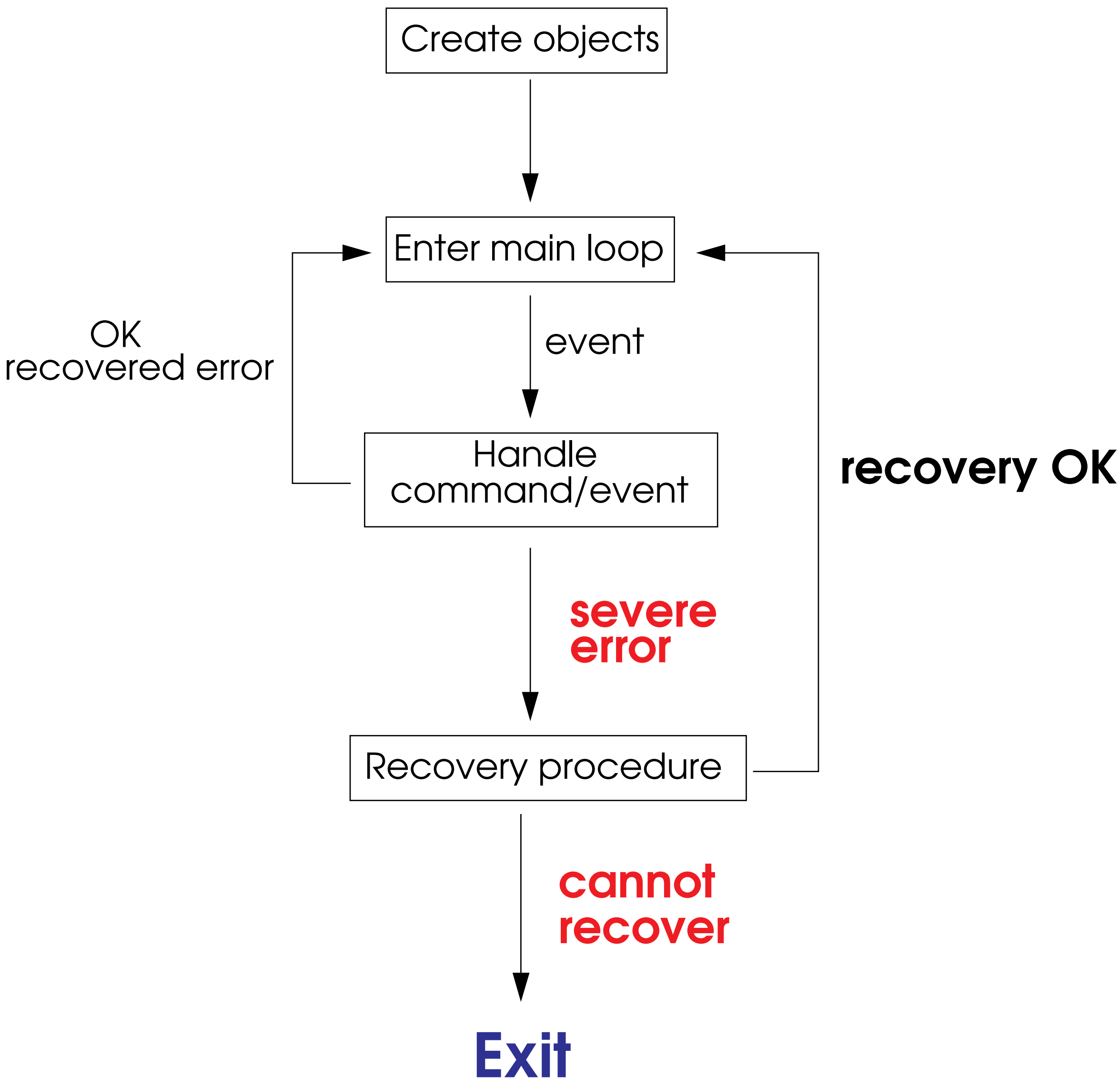
DFD for the prsAction process



State machine for agwsControl



Main loop handling. Error recovery



The tcsMAIN_HANDLER class

- Main loop handling
- Command line parsing
- Severe error recovery

```
template<class T>
class tcsMAIN_HANDLER : public eccsERROR_CLASS
{
    tcsMAIN_HANDLER();
    ccsCOMPL_STAT Init(int argCount, char *arg[]);
    ccsCOMPL_STAT RunMainLoop(T *obj, ccsCOMPL_STAT(T::* recover)());

    const char* DbRoot();
    const char* ProcName();
};

int main (int argc, char *argv[])
{
    tcsMAIN_HANDLER<mswCONTROL_MAIN> mainHandler;

    mainHandler.Init(argc, argv);

    dbSYMADDRESS  dbEntryPoint;
    sprintf(dbEntryPoint,     "%s:%s",
            mainHandler.DbRoot(),mswDB_ENTRY_POINT);
    mswSTD_COMMANDS     mswStandardCmds   (dbEntryPoint);

    swCONTROL_MAIN     mswControlMain(dbEntryPoint);

    mainHandler.RunMainLoop(&mswControlMain, &mswCONTROL_MAIN::Recover);

    return(EXIT_SUCCESS);
}
```

Command handling and dispatching

This area is not yet standardized.
The modules use different strategies
depending on the specific needs

$\Downarrow$

Some action is necessary in this area

prs, msw, trkws:

evhDB_CMD_SEND

- high parallelism
- synchronization issues
- special handling of mode switching list

agws, pom:

evhCOMMAND

- multi step commands

Example: prs preset sequence

```
evhCB_COMPL_STAT prsAH_MAIN_TASK_VLT::ExePresetCB(msgMESSAGE &, void *)
{
    evhCB_COMPL_STAT stat = evhCB_DELETE;
    ccsCOMPL_STAT    res  = SUCCESS;
    prsTRACK_MODE    target;
    evhDB_SEND_ID    id = 0;

    errFlag(res, track.Preset(*currentSetupFile,&target));

    //#= If target != altaz, asks for preset of the other subsystems
    if( (target != prsTARG_NONE) && (target != prsTARG_ALAZ))
    {
    id = 0;

        //#= Executes a focus correction based on calibrated values
    errFlag(res,ao.CalibratedFocusCorrection(*currentSetupFile));

    //#= Preset of Auto Guide
    errFlag(res,guide.Preset(*currentSetupFile));
    errFlag(res,guide.FindGuideStar(*currentSetupFile, &id));
    errFlag(res,guide.ObjectCenter(*currentSetupFile, &id, id));

    //#= Offset step. When step ObjectCentering is completed
    errFlag(res,track.OffsetStep(*currentSetupFile, &id, id));

    //#= Execute an accurate focus correction
    errFlag(res,ao.FocusCorrection(*currentSetupFile, &id, id));

    //#= Acquisition of final guide star and start guiding
    errFlag(res,guide.Start(*currentSetupFile, &id, id));
    }

    //#= Ask the stdMod object to request a SETUP for all the other standard
    //#= modules registered in the application:
    stdMod.Setup(*currentSetupFile, &id, id);
```

```
    //#= Activates a syncronizzation object to wait until all the subsystems
    //#= are ready (or a timeout occurrs):
    //#=        Events:     DB:prsAction.track.state  >= IDLE
    //#=                    DB:prsAction.guide.state  >= IDLE
    //#=                    DB:prsAction.ao.state     >= IDLE
    //#=                    DB:prsAction.stdMod.state >= IDLE
    //#=        Action CB:  ReadyCB
    //#=        Timeout:    ErrorCB   Time:     DB:prsAction.readyTimeout
    dbSYMADDRESS  objDbName;

    waitAllReady.ClearConditions();

    sprintf(objDbName, "%s.%s",track.DbPoint(),"state");
    waitAllReady.AddCondition(objDbName,evhSTATE_IDLE,TRUE);

    sprintf(objDbName, "%s.%s",guide.DbPoint(),"state");
    waitAllReady.AddCondition(objDbName,evhSTATE_IDLE,TRUE);

    sprintf(objDbName, "%s.%s",ao.DbPoint(),"state");
    waitAllReady.AddCondition(objDbName,evhSTATE_IDLE,TRUE);

    sprintf(objDbName, "%s.%s",stdMod.DbPoint(),"state");
    waitAllReady.AddCondition(objDbName,evhSTATE_IDLE,TRUE);

    waitAllReady.Run();

    if(res == FAILURE).....

    return stat;
}
```

Example: pom parallel commands

```
class pomMAIN_TASK : public evhDB_TASK {
  public:
    virtual evhCB_COMPL_STAT ReadPosReceiveCB(msgMESSAGE &msg, void *);
private:
    virtual evhCB_COMPL_STAT ReadPosReplyCB(msgMESSAGE &msg, int seq);
    virtual evhCB_COMPL_STAT ReadPosErrorCB(msgMESSAGE &msg, int seq);

    msgMESSAGE        readPosMsg;
    evhDB_COMMAND     readPos_1Cmd;
    evhDB_COMMAND     readPos_2Cmd;
    vltINT8           readPosFlag;
};

evhCB_COMPL_STAT pomMAIN_TASK::ReadPosReceiveCB(msgMESSAGE &msg,void *)
{
    msgMESSAGE sendMsg;

    ErrReset();

    // Save received message, to send final reply later on
    readPosMsg = msg;

    // Prepare and send the 1st command
    sendMsg.Command(xxxFIRST_CMD).Buffer("test buffer send to process 1");
    readPos_1Cmd.Send(sendMsg);

    // Prepare and send the 2nd command
    sendMsg.Command(xxxSECOND_CMD).Buffer("test buffer send to process 2");
    readPos_2Cmd.Send(sendMsg);

    readPosFlag = 2; //  Set the counter flag

    return evhCB_NO_DELETE;
}
```

```
evhCB_COMPL_STAT pomMAIN_TASK::ReadPosReplyCB(msgMESSAGE &msg, int seq)
{
    evhCB_COMPL_STAT stat = evhCB_NO_DELETE;
    ErrReset();

    // If it is the last reply, I am done
    if( msg.LastReply()){
        readPosFlag = readPosFlag -1;
        if(readPosFlag == 0)
            if( readPosMsg.SendReply(tcsOK_MSG) == FAILURE ){
                ErrAdd("evh",evhERR_SEND_REPLY,__FILE_LINE__);
                ErrStackClose();
                }
        stat = evhCB_DELETE;
    }
    return stat;
}

evhCB_COMPL_STAT pomMAIN_TASK::ReadPosErrorCB(msgMESSAGE &msg, int seq)
{
    ErrReset();
    ErrAdd("eccs",eccsERR_GENERIC, __FILE_LINE__);

    readPosFlag = readPosFlag - 1; // we decrement the flag first

    /* If there is still a pending command we must abort it */
    if(readPosFlag & (seq == 1))readPos_2Cmd.Reset();
    if(readPosFlag & (seq == 2))readPos_1Cmd.Reset();

    // Send back the final error reply
    if( readPosMsg.SendReply(ErrStack()) == FAILURE ) {
        ErrAdd("evh",evhERR_SEND_REPLY,__FILE_LINE__);
        ErrStackClose();
    }
    else
        ErrStackReset();
    return evhCB_DELETE;
}
```

Failure handling

- Error conditions must be recovered before returning in the main loop.
- Only internal severe errors (like database not available) can be passed to the main loop.
- In case of error on handling a command, the error must be reported with an error reply:

```
evhCB_COMPL_STAT agwsCON_MAIN_TASK::GenericCB(msgMESSAGE &msg, void *)
{
    eccsLOG_1((“agwsCON_MAIN_TASK::GenericCB - Command: %s, Buffer: %.60s”,
           msg.Command(), msg.Buffer()));

    ErrReset();

    //#= send back an error reply, telling “command not yet supported”
    ErrAdd(agwsMOD, agwsERR_GENERAL, __FILE_LINE__,
        “Command not yet supported”);

    if (msg.SendReply(ErrStack()) == FAILURE)
    {
        ErrAdd(tcsMOD, tcsERR_SEND_ERROR_REPLY, __FILE_LINE__);
        ErrStackClose(); // Close and log, since error not reported
    }
    else
        ErrStackReset(); // Error recovered. Close, since error reported

    return evhCB_SUCCESS;
}
```

Modular testing

In the modular test it is necessary to test:

- module's functionalities and command interface (application level testing)
- any class, one by one (class level testing).

Tests on different modules are independent one another:

- Dummy processes
- Test database
- Independent environments

Reference documents

- **TCS Functional Specification**
  VLT-SPE-ESO-11720-0001
- **TCS Package Design Description**
  VLT-SPE-ESO-17230-0754
- **TCS Preset Design Description**
  VLT-SPE-ESO-17230-0797
- **TCS Mode Switching Design Description**
  VLT-SPE-ESO-17230-0819
- **TCS Tracking Design Description**
  VLT-SPE-ESO-17230-0755
- **TCS Auto Guiding Design Description**
  VLT-SPE-ESO-17230-0933

Conclusion

All TCS workstation modules have the same general architecture.

This makes the software coherent and easier to maintain.

A periodic parallel cleanup of all modules is very important to factorize common problems and solutions.

Handling of command dispatching is still implemented using different strategies in the various modules.

It is important to codify the allowed strategies in design patterns and to develop common classes to implement them.

This is a major open point.