# Status of the ELT control software development

Gianluca Chiozzi[1], Luigi Andolfato, Javier Argomedo, Carlos Diaz Cano, Robert Frahm, Josef Hofer, Bogdan Jeram, Nick Kornweibel, Federico Pellegrin, Marcus Schilling, Heiko Sommer, Arturo Hoffstadt Urrutia
European Southern Observatory, Garching bei Muenchen, Germany

## ABSTRACT

The Extremely Large Telescope (ELT) is a 39 meters optical telescope under construction in the Chilean Atacama desert.

The control software is under advanced development and the system is slowly taking shape for first light in 2028.

ESO is directly responsible for coordination functions and control strategies requiring astronomical domain knowledge. Industrial contractors are instead developing the low-level control of individual subsystems.

We are now implementing the coordination recipes and integrating the local control systems being delivered by contractors. System tests are performed in the ELT Control Model in Garching, while waiting for the availability of individual subsystems at the telescope.

This paper describes the status of development for individual subsystems, of the high-level coordination software and of the system integration on the ELT Control Model (ECM), focusing on testing and integration challenges.

**Keywords:** ELT, telescope control software, architecture, design

## 1. INTRODUCTION

The ELT is the largest and most complex ground-based telescope under construction, having a large segmented primary mirror with a diameter of 39m and a novel 5-mirror optical design.

To provide the science instruments in the Nasmyth foci with a wavefront having an error in the nm range, the operation of the ELT's five mirrors must be carefully orchestrated. This is the task of the ELT Control System. In particular,

- The M1 segments are individually moved in piston, tip and tilt to control the shape of the mirror, keeping the relative edge displacements within several nm.
- The deformable M4, with a diameter of 2.4m, can adjust its shape with 5352 degrees of freedom at rates up to 1 kHz [9] to perform adaptive optics with natural guide stars or with the support of up to 6 laser guide stars.
- The M5 is a flat mirror that includes a fast tip-tilt system for image stabilization, whose purpose is to compensate perturbations caused by wind, atmospheric turbulence, and the deformations of the telescope itself.

Given the limited stroke (100 nm) of the M4 actuators and the limited capture range of the wavefront sensors in the guide probes, all mirrors shall be adjusted cooperatively to redistribute the slowly building non-zero-mean components in low order modes of M4 to the other degrees of freedom.

The project has surpassed the 50%-complete milestone, and the overall status is presented at this conference in [1].

In this paper we focus on the status of the Control System.

We will first provide some necessary context information for the readers not already familiar with the architecture, for which more details can be found in [2] and [7].

We will give information about the status of implementation of the different components, providing more details also in terms of design and architecture, for some of them that are not extensively described in other papers.

We will also analyze how our workflow and testing strategies have evolved during this construction phase.

[1] gchiozzi@eso.org; phone +49-89-32006543

Software and Cyberinfrastructure for Astronomy VIII, edited by Jorge Ibsen, Gianluca Chiozzi, Proc. of SPIE Vol. 13101, 1310105 · © 2024 SPIE
0277-786X · doi: 10.1117/12.3019706

## 2. THE ELT CONTROL SYSTEM

The ELT Control System (CS) implements the overall control of the telescope (and dome), including the computers, communication, and software infrastructure. The architecture has been described in [2] and [7] and since then we have consolidated and further developed it in detail. Here we describe just some key aspects to provide a context.

Figure 1 shows an overview of the ELT CS, with some simplifications and omissions for readability (*numbers circled in orange are used to identify items referenced with* "Figure 1-*(#)" in the text below*).

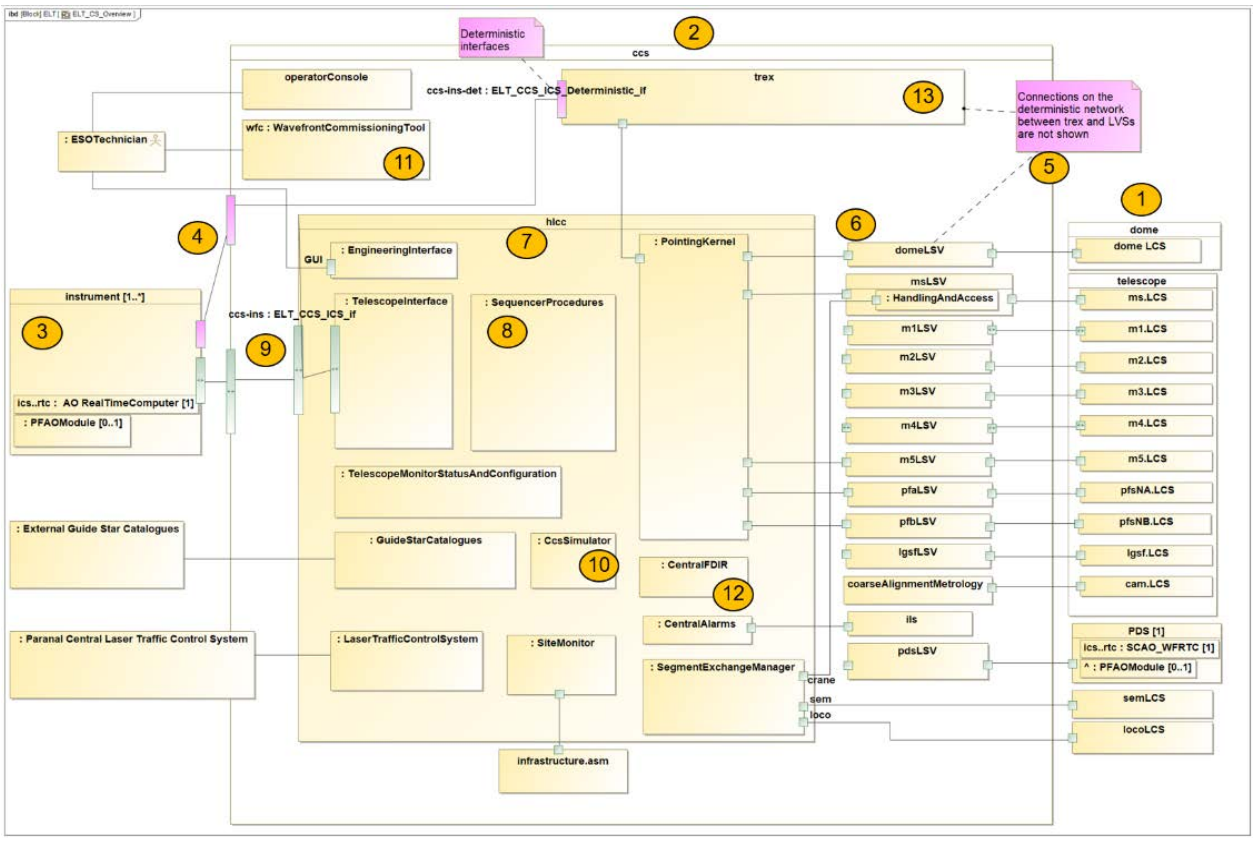


Figure 1: ELT Control System overview

The first breakdown of the ELT CS is into the many individual control systems associated with Telescope subsystems (called the Local Control Systems (LCS)) (Figure 1-(1)), and the single system that integrates these, the Central Control System (CCS) (Figure 1-(2)).

A subsystem is responsible for autonomous control, monitoring and safety through its LCS. All LCSs are integrated into the ELT Control System via a dedicated LCS-CCS interface (Figure 1-(5)). The interface is strictly defined (and implemented) as a suite of standard protocols running over switched Ethernet.

The separation between CCS and the LCSs permits a clear distintion between unit-level, and telescope-level control and safety. Further, it matches organizational boundaries in-line with the procurement strategy: individual subsystems are designed, built and delivered by industrial partners; integration into the telescope, and the CCS, is responsibility of ESO.

Each LCS has a partner component in the CCS called a Local Supervisor (LSV), (Figure 1-(6)). The LSVs provide several functions: they contain operation- and observation-domain knowledge associated with a subsystem (e.g. tracking a celestial target), they permit hardware adapation in the event interface deviaions or non-compliances in the delivered LCSs, and, through design patterns, the LSVs streamline supervisory control and monitoring of the subsystems. For example, sky tracking algorithms are implemented inside the Main Structure (MS) LSV code, which will command the MS LCS to move the axes to a specific (alt, az) position.

On the other side of CCS are the instruments (Figure 1-(3)), developed by the consortia of ESO partner institutes. Each instrument includes an independent Instrument Control System (ICS) developed following the ELT standards[5], and interfacing with the telescope through the INS-CCS interface. The INS-CCS interface covers supervisory control and monitoring (e.g. preset and offsets), over the control network, real-time communication (e.g. AO commands) over the deterministic network (Figure 1-(4)), and safety signals and monitoring, over the Interlock and Safety Network.

The CCS integrates the many LCSs into a single system implementing the coordinated control, system level safety, monitoring and user interfaces required to operate the telescope. It provides monitoring, logging and archiving for long term trending and configuration control. Control Room terminals, GUIs and debugging and diagnostic tools belong to the CCS.

CCS applications are organized in a shallow hierarchy of loosely coupled components as can be seen in Figure 1-(2).

The coordination of all the subsystems and the implementation of the high level strategies that define their interactions is the responsibility of the High Level Coordination and Control (HLCC) (Figure 1-(7)) and of the Telescope Real-time Executor (TREx) (Figure 1-(13)) components of the Central Control System (CCS).

The Wavefront Commissioning Tool (WFTool, Figure 1-(14)) includes hardware, software, and development environment connected (in a possibly privileged manner) to the Control System for verification and monitoring purposes, in particular during commissioning. This tool manages the data model defining the telescope. The model is updated as a result of changes in the subsystems (e.g. M1 segment replacements, hardware failures or repairs), new calibrations or changes in the wavefront control strategy. Data products from the telescope model are pushed to the CCS, LSVs and TREx in a version-controlled manner ensuring an integrated and coherent control of the telescope. The term “Commissioning” in this tool’s name comes from the need for its flexibility and scriptablity for use during telescope commissioning. The product will later evolve into the Wavefront Operations Tool.

The flow of communication between the components of the CS, and the most important design patterns, have been previously described in [2] and [8]. We summarize here the most important aspects relevant for this paper:

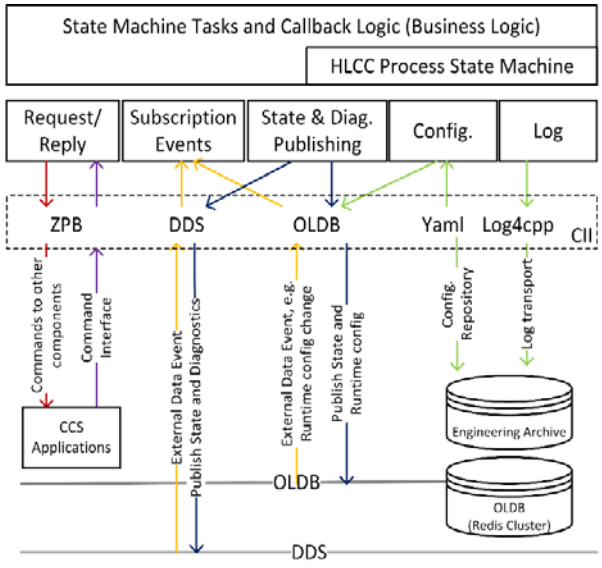


Figure 2: HLCC process software stack

- Interfaces are formally defined using an interface specification language.
- Commands are implemented using a request/reply design pattern, where the reply does not typically indicate the completion of a command, rather only correct reception and commencement of the associated action.
- Every component publishes its own state information using a publisher/subscriber design pattern. State information shall be used by clients to assess the state of a service, for example, to confirm it corresponds to what was requested via commands. State information is always published, starting from when the control system is powered on.
- Status and configuration information is also published on a low performance online database, primarily for observability from UIs and other clients.

The implementation relies on the Core Integration Infrastructure (CII) that we have designed to ensure that everybody uses a common software inftrastructure that wraps the chosen technologies, making uniform how they are applied.

Figure 2 exemplifies the usage of CII in the HLCC case.

## 3. CONTROL SYSTEM DEVELOPMENT PLAN AND STATUS

The Local Control Systems are developed by the contractors implementing the hardware (with the exception of the M1 LCS that is developed in-house) and their delivery is therefore directly linked with the delivery of the hardware.

The subsystems are delivered by the contractor and their control system is accepted by ESO together with the hardware, stand alone. At that point, integration with other components of the system starts and the LSVs are connected with the corresponding LCS.

Table 1 shows the current high-level schedule for the subsystems, from the point of view of the CCS team. Significant in this schedule are the dates of control system testing with the supplier's LCS and the point in time at which the LCS design may be considered consolidated enough to rely on the interface documentation for LSV development (worst case is Preliminary Acceptance, though in many cases there are earlier milestones which can be considered significant for LCS consolidation and testing).

Table 1 - Subsystems schedule until end of 2026

| System | 2024 | | | | 2025 | | | | 2026 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2024 Q1 | Q2 | Q3 | Q4 | Q1 | Q2 | Q3 | Q4 | Q1 | Q2 | Q3 | Q4 |
| Dome | | | | | | | | | | | | |
| MS | | | | | | | PAC | | | | | |
| M2 Cell | | | | | | | | | | | | ARM |
| M3 Cell | | | | | | ARM | | | | | | |
| M4 | | | | | | ARM | | | | | | |
| M5 Cell | | | | | | | | ARM | | | | |
| PFS MS | | | | | | | | | | IRR | | |
| PDS | | | | | | | | PAE | ARM | | | |
| LGS | | | | | PAE | ARM | | | | | | |

| | |
|---|---|
| | Test in-house with real unit |
| | Test campaign mission |
| | Periodic subsystem delivery from vendor |
| PAE | Preliminary Acceptance Europe |
| PAC | Preliminary Acceptance Chile |
| ARM | Handover in Armazones to start of AIV workflow activities. |
| IRR | Integration Readiness ReviewThe system will be installed on the telescope. |

Follow up on the LCS evolution for each subsystem is done typically by the same small team developing the corresponding LSV and relies on periodic releases delivered to ESO by the vendors.

Representative LCS hardware (LCUs) of all subsystems listed above are already installed and available in the ECM. The state of the software deployed on those systems is evolving with the contract, as are the interface details and simulation functions.

Several subsystems will be complete around the end of 2024. Thus 2024 is critical to finalize the CCS interfaces and implement critical LSV functions involving those interfaces. Ideally these critical LSV functions may be tested directly against the LCSs at the supplier's premises prior to acceptance.

The LSVs and other components of CCS are developed in ESO and are tightly linked with the needs of AIV at the observatory, to satisfy first of all the requirements of integration and testing of the subsystems as they are delivered.

As can be seen in Figure 3, it is planned to have the computing infrastructure ready in Armazones for the beginning of 2026 and the first LSVs will then be integrated with the LCSs, starting with the Dome and Main Structure and followed by the M1 and the Pre-Focal Station (PFS) containing the guide probes with the sensors used for guiding and the WFS for the telescope-based adaptive optics.

With the beginning of 2027 we plan to install HLCC with high level functions to coordinate the available subsystems. The functionality will be extended as new subsystems are integrated.

During 2027, the integration of the other mirrors (M2, M3, M4 and M5) in the optical path will take place, followed by the LSVs and together with TREx, the Wavefront Real Time Computer (WFRTC) and additional HLCC functions needed to coordinate the operation of the mirrors.

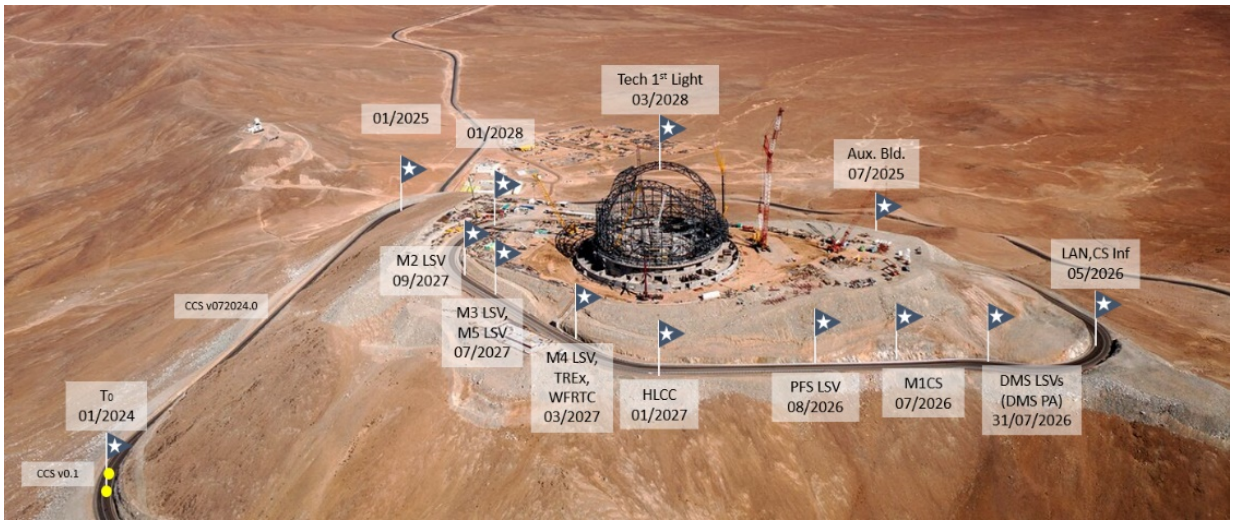


Figure 3 - The road to Armazones

The objective is to be ready for technical first light at the end of Q1 2028.

To monitor the progress and ensure that we can reach these objectives, we plan two integrated and incremental releases of CCS per year, to be installed and tested first on the ECM using simulation models for the LCSs and then on the actual platforms that will be made step by step available.

In 2023 we distributed to the consortia developing ELT instrumentation, two releases of a package including the Instrumentation Framework (presented at this conference in [5]) and the telescope simulator (based on HLCC and implementing the interfaces between telescopes and instruments).

In January 2024 the first "CCS integrated release" was installed on the ELT Control Model (ECM). This is the first release of CCS combining deliveries of several software projects, including the full software infrastructure and libraries. The release comprises the MS LSV, M2/M3 LSV, and HLCC with all subsystem functions present, though not fully functional. The Alt and Az axes subsystem functions are implemented as well as the interface between HLCC and MS LSV for pointing, tracking, and offsetting.

Each of the upcoming releases will add new functions to the already integrated components, integrate new components. HLCC will implement high level coordination functions across the newly integrated functions of the LSVs and TREx.

What follows is a summary of the most important features of the CCS planned until the end of 2027:

- 2024
  - Integration of PFS, M4, M5 LSVs and TREx in the ELT Control Model.
  - Outstanding features of M2/M3 LSV, including feedback after testing with real M2 cell.
  - PFS LSV interface with TREx and almost complete Guide Probe control.
  - Integration of Dome LSV with initial implementation of thermal and wind flow control.
  - Integration of M1 LSV with complete figure loop implementation.
  - TREx will implement field stabilization, AO interfaces with instruments and guide probe image analysis.
  - HLCC focusses on the CCS-INS interface, Preset, Tracking, Offsetting and Guiding use cases.
- 2025
  - Telescope axes are fully functional including deterministic guiding interfaces with TREx.
  - PFS includes basic interface to the cameras.
  - TREx delivers initial implementations of various functions on the AO path (GPAO, M4 projection, anti-windup, M4 reference shape, and M4 registration) and performance computations.
  - TREx guiding interfaces (to PFS probes and MS) are complete, as are counterparts in PFS and MS LSVs.
  - The M4 LSV delivers a completed wave front correction function.
  - The M5 LSV implements the alignment function.

- HLCC delivers complete functions for acquisition and field stabilization use cases and what needed for TREx supervisory control.

- 2026
  - M4 LSV implements last low-risk functions: Position Control, Focus Selection and Thermal Control.
  - HLCC completes any accumulated modifications needed to core functionality: Pointing, Tracking and Offsetting, new release of the CCS Simulator, and updates to the sequences.
  - The M5 LSV is the focus in these later iterations: sequencer scripts and Tip/Tilt function are completed.
  - The TREx repeats performance verification, with a possible hardware upgrade if needed.
    The TREx, HLCC, M4 LSV, PFS LSV, MS LSV, Dome LSV and M1 CS are complete, ready for AIV.
- 2027
  - Most of the CCS is already involved in AIV.
  - Only M2 and M5 remain to be completed and are under development in Garching, preparing for AIV.
  - The release focusses on testing and documentation.

## 4. LOCAL SUPERVISORS

The LSVs are all subject to a common set of requirements and to a common architecture.

A subsystem is presented to the other components of CCS via its Subsystem Functions (defined as the operational activities or purposes for which that Subsystem was designed. For example, tracking a target, imaging a target, correcting wavefront, and so forth).

At the design and implementation level this concept has been strongly modularized and an application library has been provided to support the common design across all LSVs and their development teams.

The Subsystem Functions are implemented within the LSV (either via interaction with the LCS, or within the LSV itself) and the knowledge of the astronomical domain resides here. For example, sky tracking algorithms are implemented inside the Main Structure LSV code, which will command the MS LCS to move the axes to a specific (alt, az) position.

As already described in [2], the LSV common architecture relies on the *Estimator-Controller-Adapter* pattern (Figure 4).

Each Subsystem Function can implement multiple controllers and estimators. It exposes an interface implemented by a Primary Controller process and this eventually dispatches requests to other controller processes. The Primary Controller is a single point of access for the function, responsible for controlling the whole Subsystem Function. The public interface of the primary controller can be used by the HLCC SW to control the associated Subsystem Function.

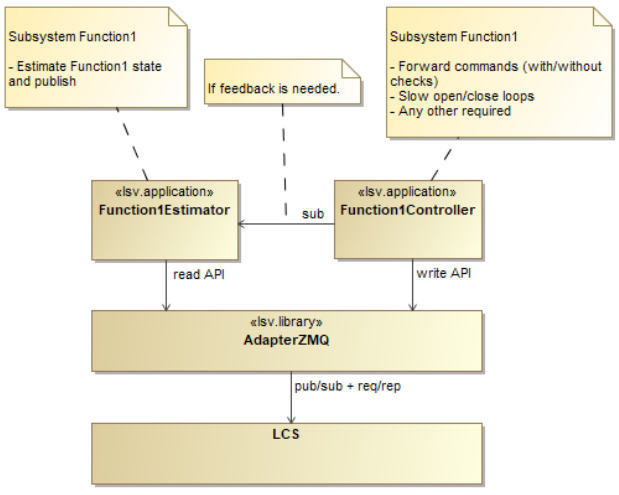


Figure 4 - Estimator-Controller-Adapter pattern applied to an LSV Subsystem Function

In the common LSV design that has emerged from the architecture in the last years, each estimator and each controller are implemented as individual processes to make the system resilient to the crash of individual components.

We have also introduced the concept of Target Variables.

While State Variables provide information on the estimated state of the system under control, Target Variables describe the intention of the user: where the State Variable should converge to. For example, a Target Variable could be the target coordinates of the telescope axes and the associated State Variable could be the actual position of the axes.

Target Variables are a simplification of the Goal concept[11] and are published, when needed, by the Controller.

Target Variables can also be used to transport information on the control system state (e.g. Controllers and, if needed, Estimators).

Whenever possible, there should be a clear association between Target and State Variables.

Target Variables should be published by the Controller (and not by the Estimator) because:

- The Controller must know the target (which is the set-point for the control algorithm). The Estimator is not required to know the target, on the contrary it should be able to work only with the measurements from the LCS.
- To avoid sending the target to both Controller and Estimator which would have a cost on the CCS HLCC.
- To avoid circular dependencies between Controllers and Estimators, since a Controller may need to subscribe to the Estimator (to receive State Variables).

Local Supervisors have no safety critical functions, though they may interface with LCS safety unit for observability of the subsystem.

The following subsections provide some information on the implementation of specific LSVs.

### 4.1 M1 LSV

The M1 Local Supervisor (M1 LSV) is a component of the ELT Central Control Software (CCS), interfacing with the M1 Local Control System (M1 LCS) and the CCS High Level Coordination and Control Software. It is responsible, together with M1 LCS, for controlling and monitoring the ELT primary mirror (M1).

M1 is composed of 798 quasi-hexagonal mirror segments of approximately 1.45m in size. Each mirror segment is equipped with Edge Sensors (ES), position actuators (PACT), and a surface deformation warping harness (WH).

While M1 LCS is in charge of collecting all the measurements from the sensors and distributing the target reference positions to the actuators[6], the main goal of M1 LSV is to compute the actuators' positions to maintain the alignment and the shape of the segmented mirror and to compensate for telescope structure deformations (e.g. due to gravity), thermal and wind effects. The control algorithm is split into 3 stages (measurements acquisition, references computation, references distribution) which are executed in pipeline (see Figure 5). The start of each stage is triggered by a SYNC message which is generated every 2ms and is multi-casted to all servers and devices. Each stage of the algorithm is therefore executed at 500Hz.

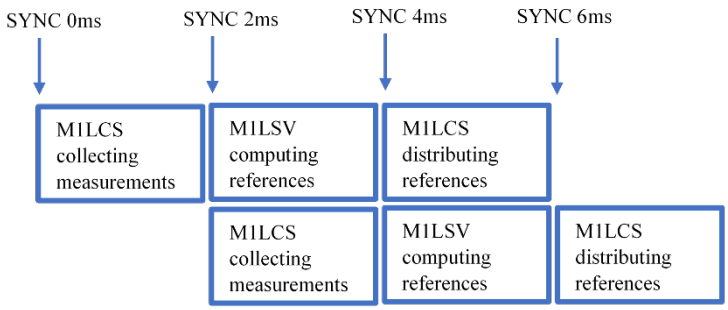


Figure 5 – M1 control algorithm pipeline

The part of the control algorithm implemented by M1 LSV (see Figure 6) is made of four loops running in parallel:

- a feed-forward command loop responsible for computing corrections from the position of the altitude axis and the ambient temperature.
- a PACT driven control loop responsible for computing corrections from the PACT piston, tip, and tilt measurements.
- a Focus driven control loop responsible for computing corrections from the ES piston, shear, and gap measurements.
- an ES driven control loop responsible for computing the new actuators position from the ES piston measurements, to add the corrections from the other loops, and provide the final results to M1 LCS.

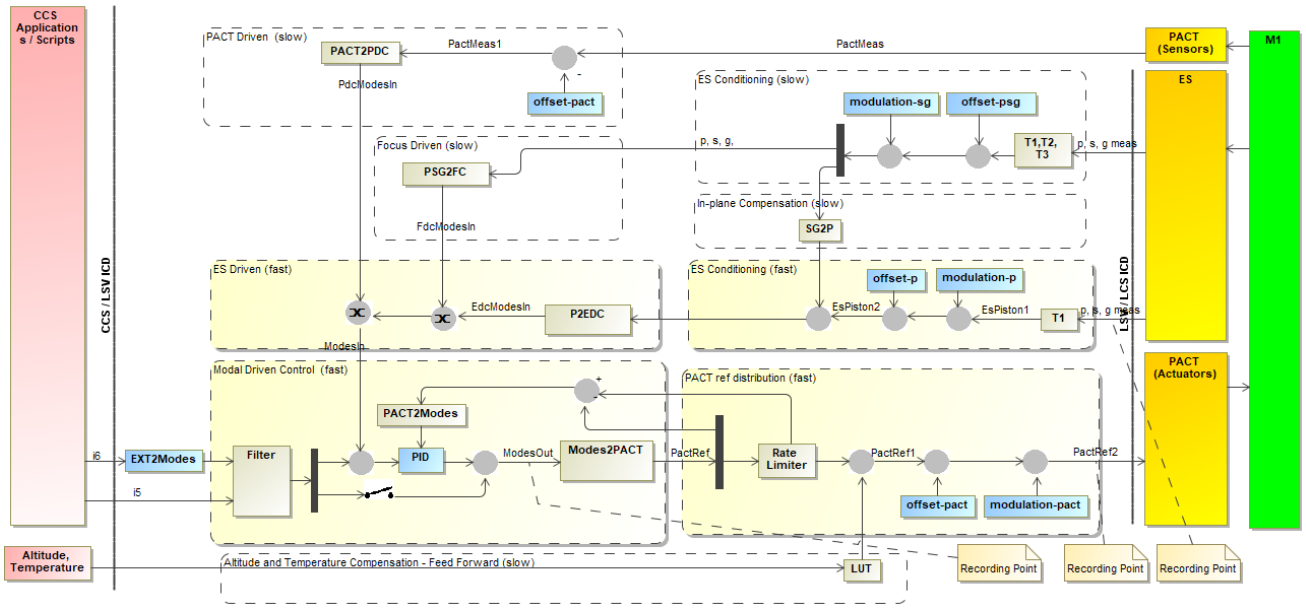


Figure 6 – M1 LSV control algorithm.

The controller offers also the possibility to inject disturbances and offsets and to record partial results at different points of the algorithm execution.

Moreover, M1 LSV is responsible for implementing the Fault Detection, Isolation, and Recovery (FDIR) strategies to improve the overall M1 system reliability. From the FMECA analysis on M1, a list of failure modes has been identified. For each failure mode the detection algorithm, the maximum time to detect, and the recovery actions have been specified. The FDIR component of M1 LSV allows to configure/enable/disable each detection algorithm and recovery action for a given period of time. All enabled and configured detections are published to the central alarm system (Figure 1-(12)). Among the main recovery actions are the reconfiguration of the control algorithm (by changing the control matrices) and the generation of sensor measurements to replace invalid values from broken devices.

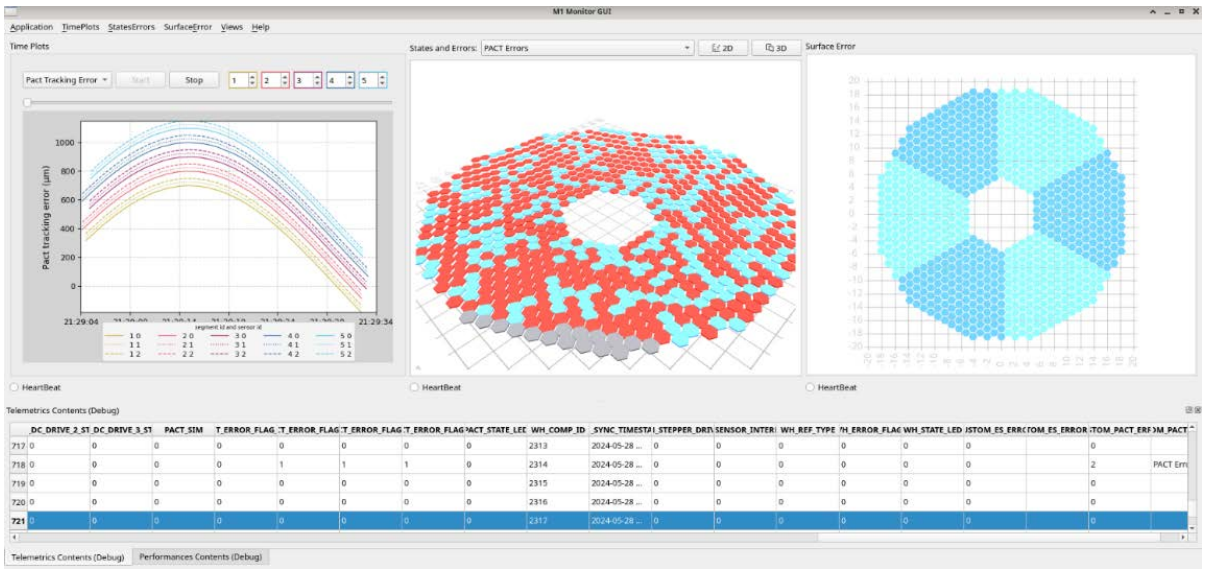


Figure 7 - m1mongio visualizes PACT and ES values, controller states and reported errors.

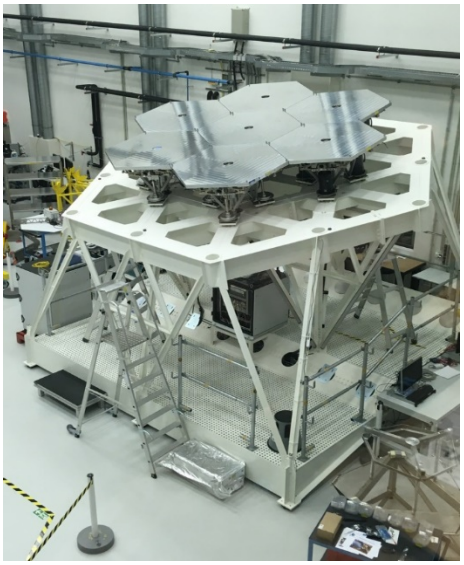

Figure 8 - M1 test stand at ESO HQ

The M1 LSV also includes operational and engineering GUIs which are under development (see Figure 7). *m1mongui* is one of them and provides monitoring of all segment's PACT and ES devices. It visualizes controller errors, status, availability and reported PTT positions or calculated surface errors for the M1 segments in 2D or 3D and prepares trends of selected segments data.

The M1 control software (M1 LSV + LCS) is tested on two different deployments: the M1 test stand[13] and the M1 Lab.

The first deployment consists of seven fully equipped and functional segments. This setup is used for functional and performance testing with real HW (see Figure 8).

The M1 lab deployment instead is made of a simulator, able to generate the traffic of all the M1 devices, and the servers running M1 LCS and LSV SW. This setup is used to simulate a 12h observation and record performance indicators such as the control loops latency and the number of packets received and distributed. The test is fully automated and executed every night. Test results are stored in a times series database based on InfluxDb and displayed via Grafana.

### 4.2 M2 and M3 Cell Local Supervisors

The M2 and M3 Mirror Cells are almost identical, supporting 4m mirrors on hexapods. A single LSV software project covers both mirrors and shall be instantiated twice with mirror-specific parameters sets.

The LSV interfaces with the cell LCS and provides the CCS with the following subsystem functions:

- Positioning Stage: This function provides active positioning capabilities of the mirror along six degrees of freedom. It is used to compensate for alignment and focus errors, and to allow the Telescope to change the optical configuration.
- Support Assembly: This function compensates wavefront errors by deforming the mirror surface, which modifies the low spatial frequency modes of the mirror. The function also allows compensating for manufacturing errors.
- System Level Interface: This function provides status information for all cell sensors that do not belong to the functions "Positioning Stage" and "Support Assembly". Examples being temperature sensors and accelerometers. Additionally, the function provides built-in test and reconfiguration commands.

### 4.3 M4 Local Supervisor

The M4 LSV provides supervisory control and monitoring of the M4 Adaptive Mirror, Hexapod, Positioning Stage and Gas Cooling System. It is not active in the AO data path, which drives the mirror at up to 1kHz communicating directly with the LCS though a high-performance real-time protocol on a separate deterministic network. Still, it does present diagnostics on any current activities with the mirror.

The M4 LSV presents the other components of CCS with the following Subsystem Functions:

- Wavefront Correction - this function monitors the adaptive mirror hardware and configuration. In addition, it monitors the real-time shaping and fast steering of the M4 adaptive mirror, reporting running means, last errors such as packet loss, skipping and saturation events. The LSV controls access to the deterministic interface to the mirror, and when disabled, permits setting mirror positions via the supervisory interface (e.g. for calibration and maintenance operations). Earthquake and wind shake events are also monitored by this function.
- Positioning – this function provides control and monitoring of the M4 Kinematic Support (Hexapod). From this point of view, it has a strong overlap with the M2/M3 Cell Local Supervisor function. Combined with the M4 Focus Selection function, the M4 LSV Positioning function publishes the Mirror Attitude Coordinates with respect to the Telescope Altitude Structure.
- Focus Selection - This function pertains to the turntable supporting the hexapod and allow to switch the mirror configuration between the two Nasmyth foci of the telescope. Although a separate mechanism from the hexapod,

the switching mechanism shares many of the control hardware and software of the hexapod and therefore has overlaps in terms of interfaces.
- Thermal Control - This function provides monitoring and supervisory control over the dedicated gas cooling system used in the M4 unit. In addition, it covers various humidity and temperature sensors present in the M4.

### 4.4 M5 Local Supervisor

The M5 Cell LSV provides supervisory control and monitoring of the M5 fast tip-tilt stabilization mirror unit of the telescope. As with the M4 LSV, the M5 LSV is not active in the AO data path, which drives the mirror at up to 1kHz, but does present diagnostics on any current activities with the mirror.

The M5 LSV presents the CCS with the following Subsystem Functions:

- Tip-Tilt – this function monitors the three piezo actuators and associated controllers of the tip-tilt stage. In addition, it provides access control for real-time driving of the mirror over the deterministic network, and commands for static position control over the control non-deterministic network.
- Alignment – this function pertains to basic operation of its three motor actuators intended for alignment of the M5 Cell and associated maintenance procedures.

### 4.5 Main Structure Local Supervisor

The Main Structure Local Supervisor (MS LSV) implements the pointing model and computations to track celestial targets and provides supervisory control and monitoring of the numerous subsystems of the ELT Main Structure.

The MS LSV presents the CCS with the following subsystem functions:

- Altitude Positioning - allows configuring, controlling and monitoring the telescope Altitude axis. It allows positioning the axis to fixed positions and performing engineering functions.
- Azimuth Positioning - allows configuring, controlling and monitoring the telescope Azimuth axis. It allows positioning the axis to fixed positions and performing engineering functions.
- Mount Tracking – provides the pointing computations to track celestial targets. This function operates directly over PROFINET IRT to the Main Structure axes controller Siemens PLC, providing time synchronization and Alt/Az trajectories to the PLC.
- M5 Positioning – involves the repositioning of the M5 Unit to direct the optical beam either toward one of the two Nasmyth platforms or in parking position.
- Handling and Access - monitors the handling and access equipment status (e.g. crane positions) and associated interlocks. This function does not provide any supervisory control interface, it is monitoring only.
- Electrical Power Distribution - allows individual control and monitoring over power supplies to the various subsystems and hosted equipment.
- Fluids Provision and Distribution - allows individual control and monitoring over cooling liquid supplies to the various subsystems and hosted equipment.

The design and development status of the MS LSV and of the other tracking supervisors are described with details in [4].

### 4.6 Dome Local Supervisor

The Dome Local Supervisor ensures dome rotation and windscreen control to follow the Main Structure as it tracks celestial targets. In addition, it implements the thermal models to maintain air conditioning during the day and dome flushing at night.

The MS LSV presents the CCS with the following subsystem functions:

- Dome Slit Doors – operation of the dome slit doors and associated control and safety equipment.
- Azimuth Rotation - allows configuring, controlling and monitoring the dome Azimuth rotation axis, to track the MS Azimuth axis, or for performing engineering functions.
- Wind Screen – control and monitoring of the wind screen, to follow the MS altitude axis as well prevent wind shake on the altitude structure.
- Wind Flow - the fixed and moving parts of the Dome have apertures that provide natural ventilation of the Telescope chamber. A pivoting panel, called a Louver, controls the size of these openings. Depending on wind,

weather and telescope tracking, the Wind Flow function operated the louvres in groups to optimize air flow through the telescope chamber.

- Thermal Control – control and monitoring of the dome cooling and air conditioning system.
- Dome Monitoring – a grouping of auxiliary systems: emergency light and power generation, rooms and machine access.

### 4.7 Pre-Focal Station Local Supervisor

The PFS LSV moves the three guide probe arms to position the sensor on celestial targets. It includes the sensor optics, camera systems and additional mirror for directing the light path to science foci.

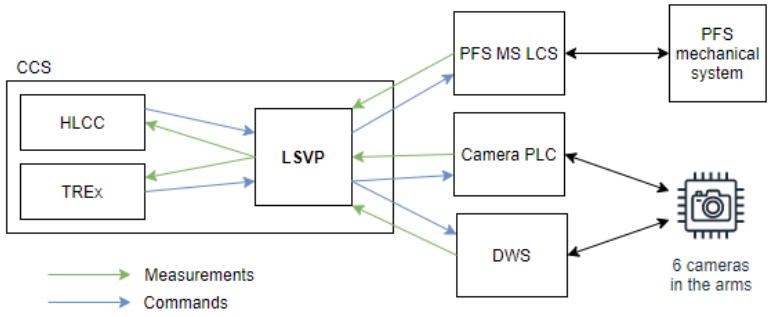


Figure 9 - PFS LSV (LSVP) with connection to Main System (PFS MS) and sensors.

The PFS LSV presents the CCS with the following subsystem functions:

- Guide probes control – manages the three sensor arms, and their optics boxes. The function converts sky coordinates (Ra, Dec) into straight-through focal plane coordinates (X, Y, theta, z), via a calibrated transformation, to position the azimuth and radial devices of the sensor arms, the pick-off-mirror and the focusing devices of the optics boxes.
- Optical beam propagation - manages the mirrors directing the central part of the optical beam to the three Nasmyth Foci, or the Coudé train.
- Cable wraps monitoring - monitors the state of the cable wraps of the sensor arms.
- Camera PLC support – monitors the camera housekeeping PLC, providing a supervisory interface to telemetry sensors in the camera, power control, alarms and fault signals.
- Camera Support - monitors and controls the six cameras mounted in the three sensors arms. The cameras are connected from one side to the Camera PLC that controls the hardware signals and, from the other side, to the detector control software deployed in the detector workstation (DWS, on Figure 9).
- Calibration source - controls the calibration light source, the calibration filter wheel and the wheel to position the fiber in front of the beam.
- Hosted metrology positioning – control of the 2-axis positioning stage hosting a laser tracker (a part of the ELT auxiliary equipment for use in coarse alignment).
- Cooling and power distribution - monitors and controls the state of the power distribution and the cooling of the PFS Main System.

### 4.8 Phasing and Diagnostic Station Local Supervisor

The PFS-A Phasing and Diagnostic Station (PDS) provides the sensors and part of the control system to bring the shape and position of the ELT mirrors from integration tolerances to deliver diffraction-limited performance in the ELT focal planes during the AIV commissioning phase of the ELT.

It also provides the sensoric means to phase the primary mirror, close high order AO loops, and diagnose the telescope on optical aspects throughout its lifetime during regular operations.

An architectural solution on the integration of the Phasing and Diagnostics Station control system, into the CCS was a recent development. The current baseline has it interfacing with CCS via an LSV, following the requirements and common software of all other Local Supervisors.

A specification for the PDS Local Supervisor, identifying its Subsystem Functions is still being drafted.

## 5. HIGH LEVEL COORDINATION AND CONTROL (HLCC)

The HLCC offers a single interface for the whole telescope toward operators and the instrument control software (Figure 1-(9)), except for deterministic interfaces that are provided by TREx (Figure 1-(4)). Its main task is for supervisory applications to coordinate the various telescope subsystems.

The HLCC architecture and design have been already described with details in [2].

HLCC relies on the same basic architectural principles used for the LSVs and uses the common software infrastructure and libraries, in particular the RAD framework [10] in combination with CII, but with some differences motivated by the specific role of HLCC as a telescope high level supervisor.

Each of the HLCC components in Figure 1-(7) is implemented as a single RAD process and both estimators and controllers are inside the same process.

Since 2022 we have implemented the design patterns necessary to provide the infrastructure to

- Command and retrieve information from the LSVs.
- Publish data for the instruments and other subsystems.
- Implement extendible and dynamically definable state and target variables using python functions.

We have also worked on the implementation of core high level use cases, like telescope presetting and offsetting. Knowing that these will evolve substantially during commissioning, they are implemented by independent supervisory applications designed to perform a complete operational function/use case of value to the users of the system. These procedures are written and executed using the Sequencer tool (see [5] for more details on the Sequencer) and are decoupled from the rest of the system (Figure 1-(8)). We are doing in some cases a special usage of the Sequencer, using the feature of running sequences directly from C++ code instead of requesting an external Sequencer Server to run them. In general HLCC makes extensive use of the possibility of running python code from within C++ processes. In this way also members of the AIV and commissioning teams can directly modify them.

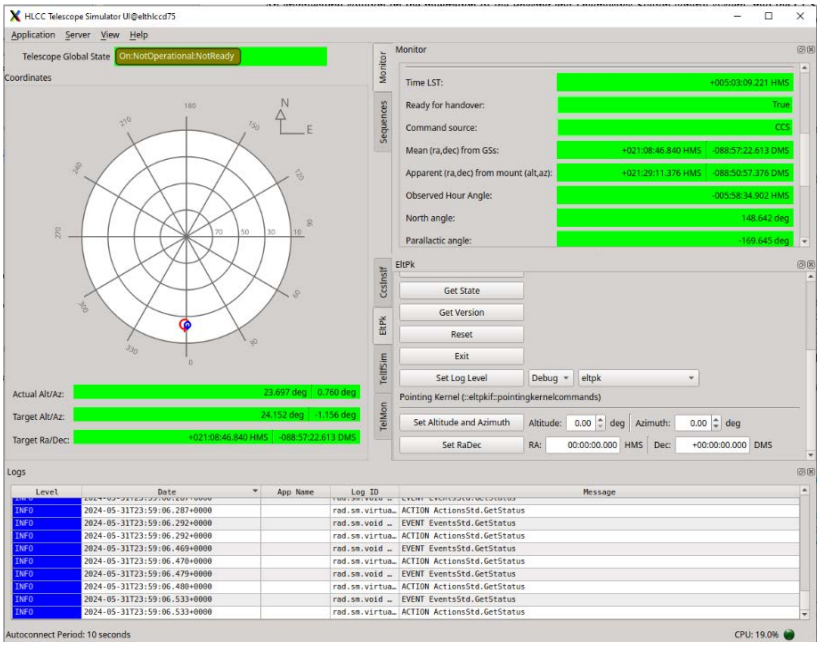


Figure 10 - Telescope simulator UI.

The Telescope Simulator has been released to consortia developing software for the ELT instruments (Figure 10 shows the main panel of the UI, implemented using the Taurus and the CUT library described with more details in [3]). The simulator implements all interfaces defined between telescope and instruments, including some of the deterministic interfaces that in the real system will be provided by TREx. The Simulator is designed to reuse as much as possible the same software components and code of the actual HLCC, so that we can use it as a testbed and receive feedback from the users. Configuration of the HLCC processes or the implementation of the interfaces of HLCC with LSVs and other components of CCS within simulators allow to reach this objective (Figure 1-(10)).

# 6. TELESCOPE REAL-TIME EXECUTOR (TREX)

Both HLCC and TREx (Figure 1-(13)) implement telescope-level control tasks. While HLCC addresses supervisory monitoring and low/slow real-time (up to 20Hz), TREx addresses on-sky loops running at rates up to 1kHz.

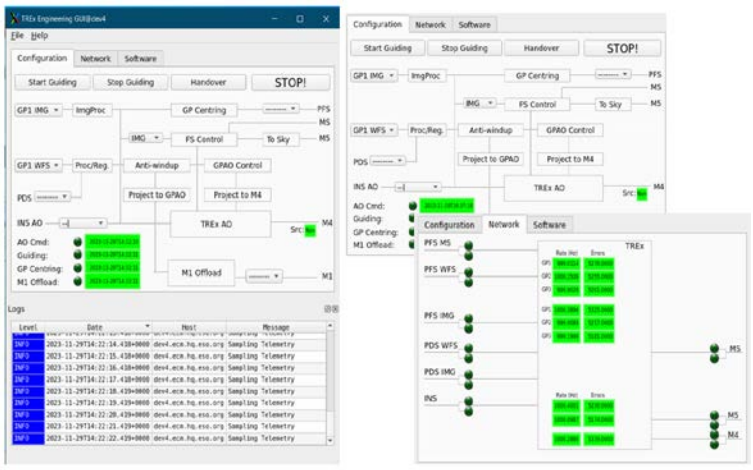

Figure 11 - TREx engineering GUI

TREx receives sensor data from the PFS guide probe and WFS cameras, as well as from the PDS, cameras or post-focal AO modules (the PDS WFRTC or an Instrument AO RTC). It produces simultaneous commands to the M4, M5, MS and M1 units to achieve system level goals of image stability and image quality, while assuring the M4 Adaptive Mirror always has sufficient stroke. More details on the architecture and on the design are available in [2].

A 1st version of the TREx real-time algorithm, consisting of a pure field stabilization implementation, has been developed with *GNU Radio*[2]. Custom blocks were created in C++ for certain sub-tasks (like centroiding or projection to focal plane). Other parts of the algorithm consist of standard blocks, as provided by GNU Radio itself. A GUI (GNU Radio Companion) is used to wire together the blocks and create Python code from the flowgraph. This allows for high flexibility, especially non-SW staff can modify the algorithm during AIV with little training. For example, Figure 12 shows the TREx Field Stabilization Controller algorithm.

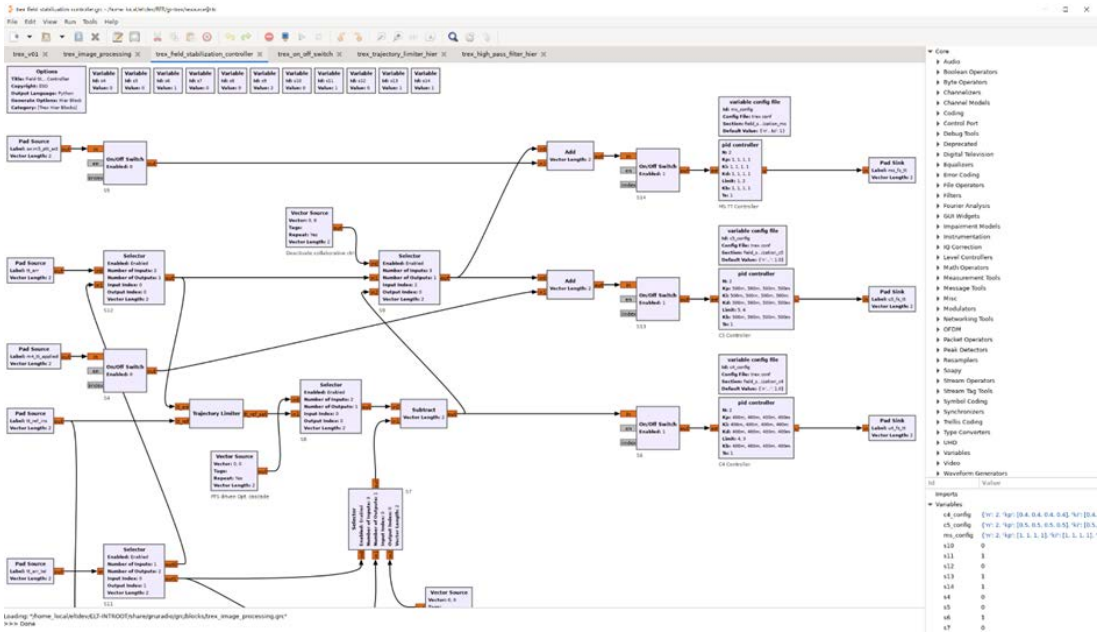

Figure 12 – TREx Field Stabilization Controller algorithm

[2] GNU Radio, https://gnuradio.org

The TREx Engineering GUI (Figure 11) provides the user with access to all functions of the TREx system: configuration, monitoring and performance. It is in essence a front end to the TREx supervisor.

The next steps will include the implementation of the algorithm blocks needed for GPAO, M4 projection, anti-windup, M4 reference shape, and M4 registration. TREx will also have to implement telescope performance estimations.

## 7. WAVEFRONT COMMISSIONING TOOL

To face the uncertainties of operating the ELT and reaching its performance goals, the CCS must be flexible and adaptable to changes in the wave front strategy. As such, one of the goals of the CCS is to act as a commissioning toolbox (rather than a turnkey solution for science, which will come later).
The commissioning interface to the telescope is the Wavefront Commissioning Tool (WFC Tool, Figure 1-(11)). The WFC Tool supports the needs of flexibility and script-ability described above. The tool is a python-based abstraction of all relevant CCS interfaces, running on a powerful server capable of running the existing python-based telescope simulations, perform computations, data reduction and generate control parameters to be pushed to the subsystem control systems (Figure 13).

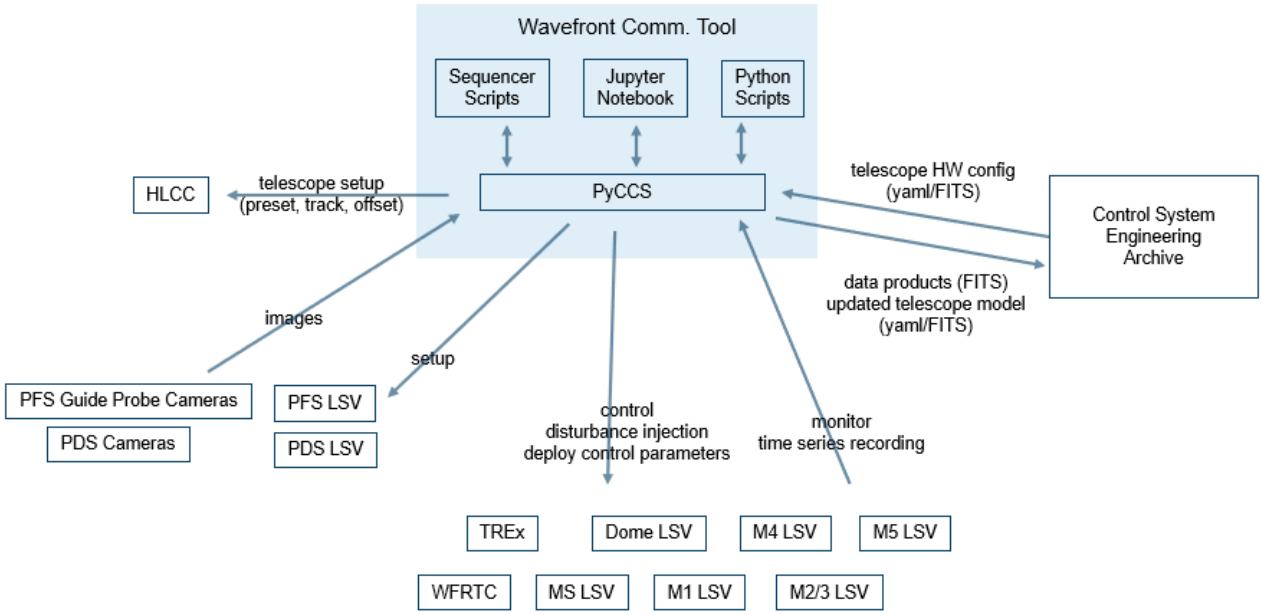


Figure 13 - The Wavefront Commissioning Tool

It is the WFC Tool that *defines the telescope*. This process involves the following general (cyclic) series of steps:

- From the Engineering Archive, load in the WFC Tool the current telescope hardware configuration (M1 segment population, M4 adaptive mirror actuator status, calibration data, etc.).
- Setup the telescope as required for the commissioning task, e.g. preset and track a guide star near the pole, setup PDS or PFS sensors, etc. This is typically performed via a sequencer interacting with the HLCC, which coordinates the setup process.
- Commence the commission task: this is performed via direct interaction with subsystem LSVs and camera systems, commanding offsets to actuators, triggering disturbance injection in controllers, recording loop data and sensor statistics, and triggering exposures on the various telescope cameras. All data is timestamped at the source, and data correlation is performed via these timestamps. Ensuring the subsystem interfaces can support the functions required by the WFC Tool is an on-going process, taking requirements from various activities: simulation activities, the MELT testbench (see section 10 below) and phasing experiments at GTC. It is also an on-going activity to work the subsystem interfaces back into the MELT software and simulation models, to ensure the work is compatible with the interfaces.

- The WFC Tool generates control solutions for the telescope: phasing solutions for the M1, M4-to-M1 interaction matrices, calibration files for data reduction, etc. This set of data artifacts is termed the *telescope definition*, and is stored, version-controlled, in the Engineering Archive.
- The WFC Tool generates subsystem-specific control and configuration data, from a loaded *telescope definition.* These are pushed to various subsystems (TREx, M1, M4, WFRTC, etc.). It is the responsibility of the WFC Tool and its versioning system to ensure the control and configuration data is coherent across the full control system. These data contain version information linking it to source.
- The Control System will apply the control parameters pushed from the WFC Tool. Each subsystem will check the configuration it loads, against a global version published in the Online Database. A subsystem will report an error if its configuration does not match the globally published version. (Note: this version check relates only to control parameters generated by the WFC Tool).

PyCCS, the python abstraction to the CCS and subsystems, already exists in a similar form in the MELT project, simplifying and abstracting access to the underlying hardware and control system services. The layer will provide a single interface to telescope cameras, sensors, all LSVs, the engineering archive and control system services such as logging, online database, and so on.
The layer will be equally usable from the Sequencer, Jupyter notebooks or directly within python applications.

## 8. RTC TOOLKIT

The RTC Toolkit is a suite of software tools, libraries, guidelines and example implementations for the Soft Real-Time Cluster (SRTC) of an ELT-scale Adaptive Optics Real-Time Computer (AO RTC), as part of an Instrument's Adaptive Optics Control System (AOCS). Its primary focus is on the performance and determinism of transferring, storing, and retrieving large amounts of data, as well as conducting calculations efficiently within specified time constraints.

The SRTC is in charge of the high-level supervision and optimization of the Hard Real Time Core (HRTC) operation. The SRTC operation is driven by the interaction with the ICS and by the reception and processing of large sets of HRTC telemetry data via complex computations (Figure 14).

The RTC toolkit is developed as part of the ELT control system. It was initiated at the end of 2019 with the design phase. The design was completed by mid-2020, and development commenced thereafter. The first alpha release was launched at the beginning of 2021, followed by the first official release at the end of the same year. Subsequently, there has been an annual release schedule, with the final release planned as part of the ELT construction phase set for 2026. After that, a maintenance phase should follow.

The users of RTC toolkit include the ELT telescope WFRTC, AO RTC developers of ELT 1st generation instruments (MICADO, METIS, HARMONI, and MORFEO,) as well as the already planned 2nd generation instrument ANDES, along with the VLT instrument MAVIS.

**Product Breakdown / Functionality**

- Reference Architecture - defines a common structure for all SRTC systems using the RTC Toolkit.
- Component Framework - provides a Component Model that defines fundamental principles and high-level concepts for software components and a Component Infrastructure that allows to create custom components.

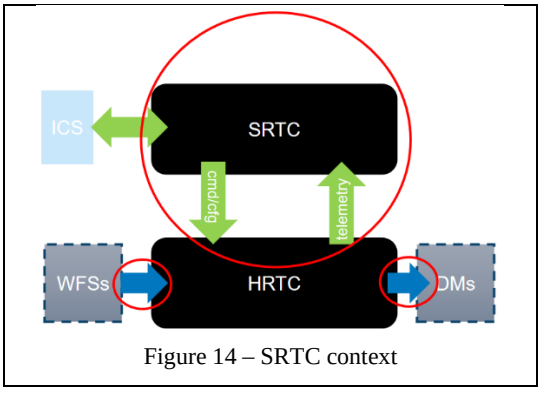

Figure 14 – SRTC context

- Reusable Components - predefined and customizable deliverables of the Toolkit that can be used by RTC system developers to build custom SRTC systems.
- Clients - command-line and graphical user interface tools to configure and interact with RTC Components.
- Standalone Tools – auxiliary deliverables that are designed for specific purposes like troubleshooting the system. Concrete examples are deterministic simulators for ELT wave-front sensors (WFSs), deformable mirror (DM) echo signals as well as raw telemetry data recorders and tools for generating telemetry data.
- Documentation - demonstrates the applied principles and gives examples of how to use and customize the provided assets to build complete systems.

- Reference Implementation - concrete examples how the Toolkit can be used to instantiate an SRTC system.

**Quality Goals**

- Maintainability - The RTC Toolkit supports the development of standardized SRTC systems, simplifies software evolution, and enables simple testing of main building blocks. To support maintainability, all SRTC systems that are using the Toolkit follow a common Reference Architecture and apply similar concepts and solutions.
- Usability - The deliverables of the Toolkit are well documented and comprehensible. Application of the principles explained in the architecture is straightforward. By following a common Reference Architecture all SRTC systems are streamlined in design and construction. The result is systems that are uncomplicated. This allows developers and maintainers to focus on a reduced set of dedicated tools and software.
- Extensibility - The Reference Architecture of the RTC Toolkit enables the system to be extended at well- defined Extension Points. It is not a goal to be open to extension in every conceivable way.
- Performance - The RTC Toolkit enables users to build software systems that meet expected performance envelopes. Artificial bottlenecks that impede performance should be avoided.
- Scalability - It is foreseen that the RTC Toolkit is used by a variety of different SRTC systems with distinctive characteristics. The Reference Architecture allows to build scalable systems.

**Solution Strategy**

- Software Product Line Engineering: The RTC Toolkit makes a clear distinction between domain engineering and application engineering: The Toolkit Development Team is responsible for domain engineering, where the focus is on development for re-use. Various RTC System Development Teams are responsible for application engineering, they build concrete SRTC systems. Here the focus is on development with re-use.
- The Toolkit imposes a Reference Architecture for derived AO RTC systems. It provides reusable core assets that are stable, have high quality and can be customized. Permitted customization techniques are black- and grey-box-adaptation. White Box adaptation is not foreseen as it makes maintenance and integration difficult.
- Component Based Software Development: The RTC Toolkit enforces a component-based software system, where the function of the system is realized via a network of collaborating but self-contained software components.
- Layered Architecture - An SRTC system follows a layered architectural style: The Presentation Tier hosts user interfaces for monitoring, control, and system configuration. The Application Tier contains the business logic that drives the system's core capabilities. This is the main location for user customizations. The Data Tier comprises infrastructure for data-exchange, persistent data storage and access to central services.
- Shared Repository Architecture - The AO RTC is a distributed control system. Several database systems act as shared repositories with different responsibilities. Software components make use of these repositories to exchange data. The following repositories are being used: Persistent Configuration Repository: a large, persistent database containing the full configuration data of the RTC. Runtime Configuration Repository: a fast, distributed database holding the Runtime Configuration and the shared state of the system. Online Database: a central database used to publish monitoring data to other systems and user interfaces.

**Reference Architecture - RTC toolkit building blocks.**

Figure 15 shows the decomposition of a generic SRTC system as a set of Toolkit-Provided Components that can be customized and instantiated, and User-Provided Components that are foreseen in an SRTC instance but are not delivered as part of the Toolkit. These components are marked with stereotype «userProvided».

- Deployment Daemon – responsible for deploying and un-deploying components belonging to the selected Deployment Set. It uses Nomad and Consul for service handling.
- RTC Supervisor – monitors all other SRTC components and acts as entry point for state control and error recovery. Is also central for populating the Runtime Configuration.
- Telemetry Re-publisher - receives raw Telemetry Data via unreliable UDP, performs conversions and republishes pre-processed Telemetry Data as DDS-based reliable Multicast. Data buffering and use of DDS ensure that network or server-side glitches/congestions do not cause problems.
- Telemetry Subscriber - receives pre-processed Telemetry Data via DDS and distributes it via a shared memory queue to Data Tasks on the same node. The use of a shared memory queue ensures that a subscriber can quickly acknowledge the published data and does not create a bottleneck.

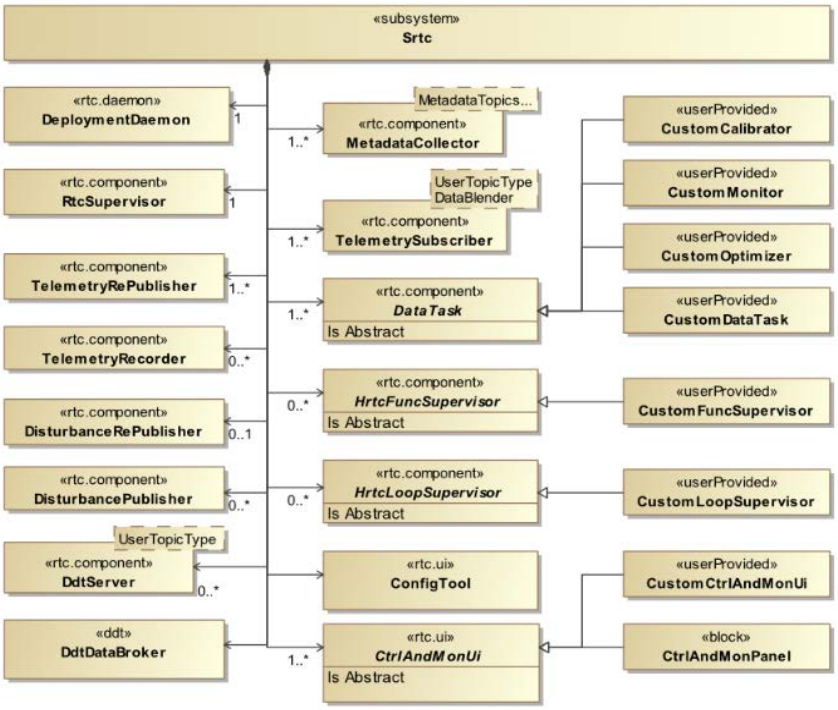


Figure 15 – SRTC decomposition

- Telemetry Recorder – ingests time series of Telemetry Data, configuration items and events and produces FITS files with user defined layout.
- Data Task – Group of components used for data processing, concrete components are Calibrator, Monitor and Optimizer. Data Tasks ingest data from shared memory or the Runtime Configuration Repository and perform their computations. Results are typically stored back to the Runtime Configuration Repository. The computation could be done on CPU using C++ or Python mathematical libraries, or GPU using CUDA.
- Metadata Collector - Implements the interfaces required by the instrument Data Product Manager acquiring Meta Data Topics characterizing RTC performance and operational conditions during an observation generated by Data Tasks and storing them in FITS format.
- HRTC Supervisor – Components that control, configure, and monitor processing stages within the HRTC.
- DDT Server - Server-side instance of the Data Display Tool that acts as data source for SRTC specific DDT Data Streams. This allows interfacing with the ELT DDT API for visualization of pixels, slopes, and mirror positions.
- Configuration Tool - Provides means to query, display and edit runtime and persistent configuration data.
- Control and Monitoring Tool - Generic tool providing introspection capabilities to browse components and to send commands, the panel may also be extended and customized by users.

## 9. ELT CONTROL MODEL

The ELT Control Model implements a scale-1 replica of all control system infrastructure and services: Logs, Alarms, Online Database, Control, Time and Deterministic Networks. In addition, it contains Local Control System control units (in simulation mode) and CCS products, as available and delivered.

Physically the ECM consists of several racks of computing nodes and networking equipment in a computer room, connected via single and multimode fiber to various laboratories hosting field equipment (Figure 16). The laboratory interface to the ECM is via “Service Connection Points”, like those that will be available around the telescope and dome of the ELT.

The ECM provides integrated testing of in-house CCS products, with external contractor deliveries, and end-to-end testing. Cameras may be physically connected to the ECM or connected in the form of Ethernet-based simulators.

An accurate time reference system, and hardware timestamping Ethernet sniffers are available for verification of Ethernet interfaces between systems.

We are also implementing as part of the ECM a “telescope control room” that will be used to design and test the control room layout in Armazones and that will be now used to control the hardware that will be delivered in Garching, and in particular the M1 test stand.

The ECM is also the place where we are testing our deployment strategy. For the deployment of the system, we are exploring the usage of Nomad and Consul from HashiCorp[3].

Nomad is a scheduler and orchestrator to deploy applications (both plain processes and containers). Consul is a tool for discovering and configuring services in the infrastructure. Consul's key features include service discovery, health checking, a Keyword/Value store, and robust support for multi-datacenter deployments.

Through Nomad and Consul, we are able to startup/shutdown/reconfigure the whole system, deploying processes dynamically on available virtual machines or bare metal systems that satisfy the individual requirements of the applications (device drivers, memory, CPU, specific interface cards). The tools can decide where the individual applications are better (re-)deployed, also taking into account load balancing optimization, and other applications can use the discovery services to interface with them in a transparent way.

At the moment we are analyzing a set of (re-)deployment use cases that include the debugging situations we expect we will have to deal with during AIV and commissioning. For example, we want to be able to pin applications to a specific

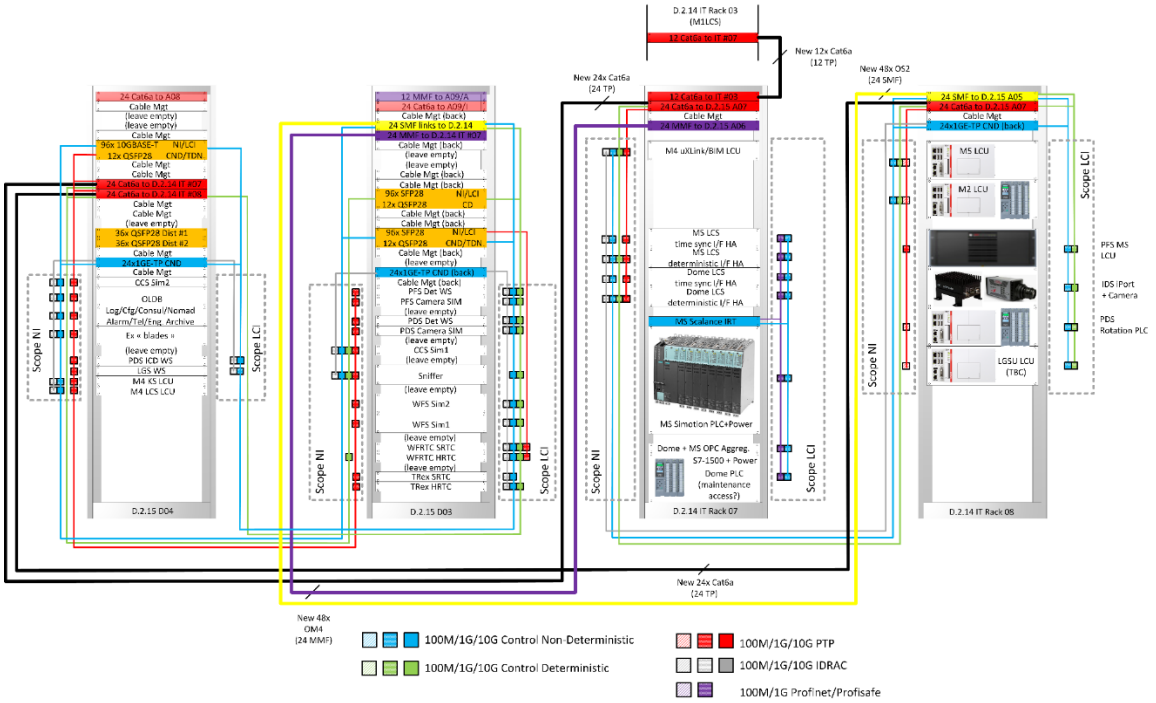


Figure 16 - ECM Cabinet Layout

host and to easily attach a debugger at run time.

[3] https://developer.hashicorp.com/

# 10. THE MINUSCULE ELT (MELT)

The Miniscule ELT (MELT) [13] is an optomechanical test bench comprised of key components such as a segmented primary mirror, a secondary mirror on a hexapod, an adaptive fourth mirror, and a fast tip/tilt mirror together that mimic certain functionalities of the ELT.

It is meant for testing and validating key functionalities to be used on the ELT during system verification, wavefront control commissioning, through the handover to science, up to regular diagnostic, monitoring, or validation during operations.

The main objectives of MELT are to deploy and validate the telescope control system as well as wavefront control algorithms for commissioning and operations.

**Advancements Towards a Fully Integrated Control System:**

PLC motors and devices are now accessible through two interfaces: the traditional raw OPC UA and an integrated interface with the Wavefront Control (WFC) tool via the Instrument Framework. This integration exposes the same interfaces as those of a real ELT instrument or the PDS, providing a practical use case for testing.

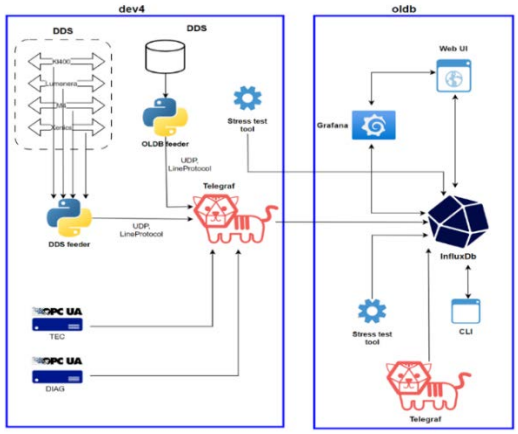


Figure 17 - Time series database prototype

In preparation for integration with the Wavefront Real-Time Computer, several interventions were undertaken, aimed at performance characterization of the bench:

- An ad-hoc benchmarking tool was developed to measure the performance of MELT subsystems.
- The network infrastructure was upgraded: MELT was incorporated into the ECM Network Infrastructure, enhancing data load capacity and usage. All servers and PLCs were upgraded with Precision Time Protocol (PTP) modules, non-deterministic and deterministic VLANs were separated.

A prototype of the time series database, integrated within the ELT Control System Engineering Archive using InfluxDB, was tested in MELT as a real use case (Figure 17). Performance measurements met the required specifications, and functional validation was confirmed.

In collaboration between the Wavefront Control Commissioning and Control System Validation teams, two key objectives were achieved: a new Pyramid Wavefront Sensor (PWFS) and associated lenses were assembled into the bench, controlled via a modulation mirror directly from the OPC UA interface (Figure 18). Additionally, a Commercial Off-The-Shelf (COTS) GigE vision camera was integrated into the Instrument Framework (IFW) Camera Control Framework (CCF).

A new Shaps Lenslet array was designed, manufactured, and assembled in the MELT bench. Key characteristics include 1159 inner sub-apertures, 300 phasing sub-apertures, and 120 registration sub-apertures (Figure 19).

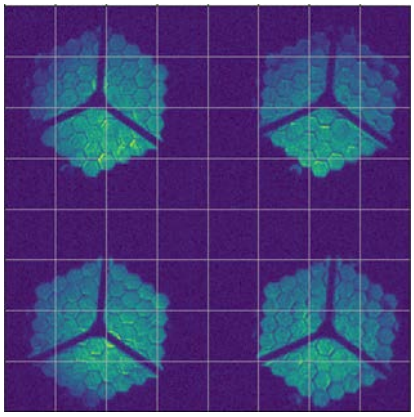

Figure 18 - Quadrants of the Pyramid WFS

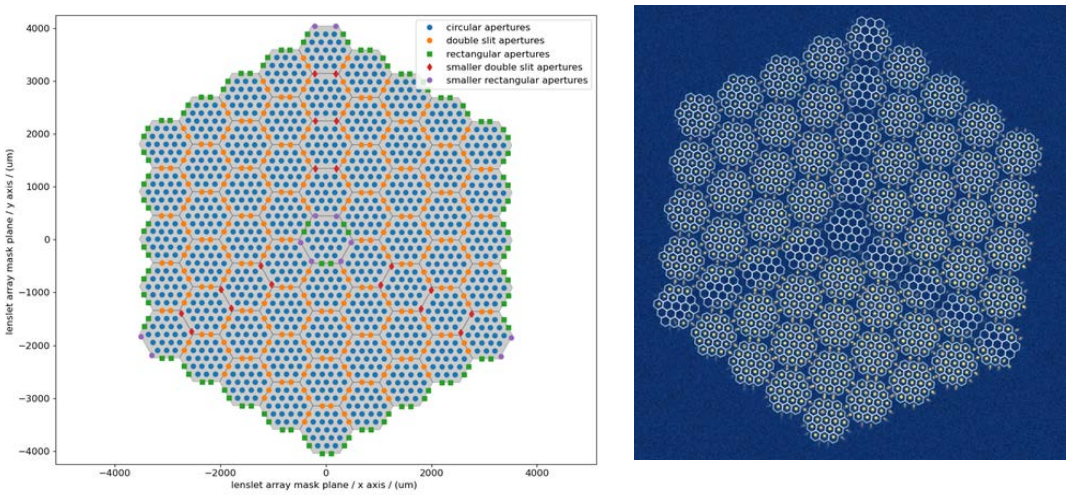


Figure 19 - Shaps Lenslet array: design vs final implementation

## 11. DEVELOPMENT PROCESS

Since 2022 (See [2], Development Process), all teams have adapted to the agile process and performed multiple iterations.

We are now in a phase where at the end of each iteration a team produces and releases its product in the form of a potentially shippable RPM packages that are made available to the other teams that depend on them.

This forces to pay attention to the quality of what is released, to managing semantic versioning in order to notify of breaking changes, and to synchronize the iteration cycles based on the dependencies.

Continue Integration allows to spot early unexpected breaking changes and incompatibilities as well as bugs that affect other components of the system. This is implemented through large sets of Jenkins pipelines that run modular and integration tests at commit time. We also implement pipelines that verify core modules (for example CII) by running downstream and not only upstream pipelines that include wide sets of dependent projects.

In this phase we still need to tune very often the synchronization between the releases of the projects, because the interfaces are still evolving and behavior is changing. We have to allocate still a sizable amount of time in adapting our code to changes in dependencies. But we aim at reaching by the end of this year a much more stable workflow, where at the end of each iteration for all components we can rely on a stable set of RPMs that everybody can use, with the option to remain for a short period of time on a more stable but older set of versions.

During each iteration, “beta RPMs” will be made available to facilitate the transition to the updated dependencies.

This means that we aim at switching from a “big bang porting” approaching at the time of a main release into a continuous and smooth evolution of the system. The effort necessary to port packages to changing dependencies is expected to rapidly become smaller with the interfaces becoming more stable and the development teams aware of the difficulties they might create to other groups if they introduce breaking changes (semantic versioning is important to notify of changes and deprecation of changed features instead of immediate deletion of old behavior is the preferred way to provide a time window to adapt to changes).

The agile/iterative process is well supported by Jira[4] for task tracking and planning, with agile plugins and we can rely on an integration between Jira, GitLab and Jenkins to support the whole development process.

In the last two years we have also increased significantly the communication across the development teams: for example, general architecture, design issues as well as common conventions are discussed in periodic meetings that involve

[4] Jira, https://www.atlassian.com/software/jira

representative of all teams; also start/end of iteration meetings are extended to all other teams having interfaces or, more in general, dependencies.

This introduces overhead but reduces the overall friction and therefore we believe it will in the end pay off giving us a more uniform and solid system. Still, we have to be very careful in finding the right balance.

## 12. CONCLUSION

Since the last conference in 2022 the development of the ELT Control System has advanced significantly in all areas.

We have consolidated and tested performance and reliability of the software platform (CII, application frameworks), implemented foundational design patterns and core functionality of LSVs and CCS control coordination components.

Most important, we have started to integrate together all the elements of the control system.

During this phase, moving from mainly architecture and design to the actual implementation, and evaluating the experience done so far, we have identified areas where it has been necessary to improve/change the initial implementation, and opportunities for simplification (for example removing support for the Java language and restricting ourselves to C++ and Python or restricting publisher/subscriber to DDS), in order to reduce the code base and maintenance costs.

Now we are preparing to move into the most exciting phase of the project, when the control software will have to be integrated, tested and commissioned on the real hardware at ESO HQ and in Chile. To make this phase effective, we know that we will have to tune our development and deployment workflow and we are testing this on the ECM and on MELT.

Our development plan is ambitious and to keep the telescope schedule with the resources available will require to increase our efficiency. But we also believe that our agile approach will allow us to set priorities and to identify lower impact areas where we will be able to descope if needed.

We will also have to find the right balance between working at the observatory, remotely from Garching and working from home or other premises (our team includes several contractors that are not at ESO HQ). Reducing our travel carbon footprint or keeping a high social and family/work balance have to be traded off with being effective, building a great team spirit and the necessity of “touching the real hardware” to understand it.

## REFERENCES


[1] Tamai, R. et al., "ESO’s ELT halfway through construction ", these proc. SPIE 13094-43 (2024)
[2] Chiozzi, G. et al., “The ELT high level coordination and control”, Proc. SPIE 12189-48 (2022)
[3] Hoffstadt, A. et al., “Taurus Integration to ELT Control Software”, these proc. SPIE 13101-142 (2024)
[4] Argomedo Zazzali, J., Andolfato L., Caproni A.,Kornweibel N., “ELT Tracking Local Supervisors design and development status”, these proc. SPIE 13101-42 (2024)
[5] Kiekebusch, M.J., “Advancements in Astronomical Instrumentation: A New Control Software Framework for ELT and VLT instruments at ESO”, these proc. SPIE 13101-17 (2024)
[6] Andolfato L. et al, "The ELT M1 Local Control Software: from requirements to implementation", Proc. ICALEPCS 2019, New York, USA (2019)
[7] Chiozzi, G. et al., “The ELT control System”, Proc. SPIE 10707-31 (2018)
[8] Chiozzi, G. et al., “The ELT Control System: Recent Developments”, Proc. ICALEPCS2021, Shanghai (2021)
[9] R.Biasi et al., “E-ELT M4 adaptive unit final design and construction: a progress report”, Proc. SPIE 9909, Paper 9909-7Y (2016)
[10] ELT ICS Rapid Application Development (RAD), http://www.eso.org/~eltmgr/ICS/documents-latest/RAD/sphinx_doc/html/
[11] M.Ingham et al., “Engineering Complex Embedded Systems with State Analysis and the Mission Data System,” Proceedings of First AIAA Intelligent Systems Technical Conference (2004), https://trs.jpl.nasa.gov/handle/2014/38225
[12] T.Pfrommer et al., “MELT: an optomechanical emulation, testbench for ELT wavefront control and phasing strategy”, Proc. SPIE 10700, Paper 10700-3F (2018).
[13] A.Reinhacher et al., “Advancements of the ELT M1 Figure Control Loop: edge sensor fault detection and management of actuator non-linearities in a large MIMO system”, proc. SPIE 13094-51 (2024)