ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL01

THE ELT CONTROL SYSTEM: RECENT DEVELOPMENTS

G. Chiozzi, L. Andolfato, J. Argomedo, N. Benes, C. Diaz Cano, A. Hoffstadt Urrutia,
N. Kornweibel, U. Lampater, F. Pellegrin, M. Schilling,
B. Sedghi, H. Sommer, M. Suarez Valles
European Southern Observatory, Garching bei Muenchen, Germany

Abstract

The Extremely Large Telescope (ELT) is a 39m optical telescope under construction in the Chilean Atacama desert. The design is based on a five-mirror scheme, incorporating Adaptive Optics (AO). The primary mirror consists of 798 segments with 1.4m diameter. The main control challenges can be identified in the number of sensors (~25000) and actuators (~15000) to be coordinated, the computing performance and small latency required for phasing of the primary mirror and the AO. We focus on the design and implementation of the supervisory systems and control strategies. This includes a real time computing (RTC) toolkit to support the implementation of the AO for telescope and instruments. We will also report on the progress done in the implementation of the control software infrastructure necessary for development, testing and integration. We identify a few lessons learned in the past years of development and major challenges for the coming phases of the project.

INTRODUCTION

The ELT is a large segmented telescope, where significant wavefront perturbations are induced by the telescope itself (deformation through gravity, temperature, and wind loads), in addition to perturbations added by the atmosphere. The goal is to control the telescope enabling the delivery of a diffraction limitable beam at each of the ELT Nasmyth foci, i.e. where the light beam is transferred to the instruments. This means the "spectrum of wavefront aberrations induced by the observatory is below that of the free atmosphere." [1].

The ELT Control System implements the overall control of the telescope (and dome), including the computers, communication and software infrastructure. It defines standards for control and electronics hardware and software and data communication. It includes the high-level coordination software, wave front control computer and engineering data archive.

In a system the size of the ELT Control System, decisions on algorithms and computational performance are not the only major design problems. The organization of the overall system, its behavior and interactions represent a significant organizational complexity which must be addressed.

An important factor influencing the architecture of the control software is the procurement strategy, that foresees the outsourcing of all components and services which can be efficiently delivered by industrial partners, while maintaining in-house those tasks for which ESO has a particular domain expertise. Based on this principle, also the overall control system shall be composed of components designed, built and delivered by many industrial partners or in-house. Distributed development and integration of the subsystems demand clear interfaces which should match not only a functional breakdown of the control system, but reflect the organizational boundaries of the many development locations.

CONTROL STRATEGY

The main challenge of ELT is to provide a wavefront with an error in the range of 10^{th} of nm in the presence of perturbations that can be in the range of mm (in the case of gravity deformation when changing the telescope pointing from zenith to horizon). The most important role in the associated control strategy is played by the deformable quaternary mirror (M4). It is controlled in an on-sky loop using stellar light with a large temporal and spatial bandwidth. This requires a deformable mirror of unprecedented size. M4 has 5352 degrees of freedom, with the on-sky loop being closed at rates up to 1 kHz [2]. The limited stroke (100um) of the M4 actuators and the limited capture range of the wavefront sensors exclude that the wavefront can be controlled solely by M4, but require it to be supported by several additional control systems:

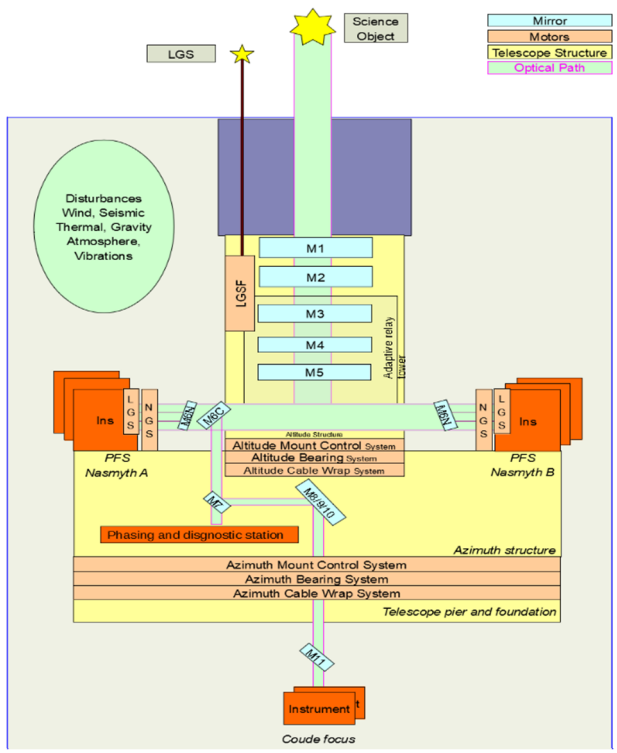

Figure 1: Telescope subsystems following the light path.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL01

- Feed-forward control is used mainly during the "blind phases" of the Telescope when on-sky loops are open. E.g. a pointing system is used to preset the telescope to a new target, with the feed forward model taking into account astrometry and telescope deformations due to gravity and temperature to bring the telescope within the acquisition range of on-sky sensors [3].
- Feedback loops based on telescope internal metrology (as opposed to on-sky loops) are used to control the state of the telescope. E.g. the large segmented primary mirror consisting of 798 hexagonal segments, and the M1 control system moves each segment in piston, tip, and tilt (2394 degrees of freedom) to control the shape of M1 based on measurements of relative displacements between segments using 4524 Edge Sensors. The M1 Figure Loop can reduce relative edge displacements to several nm and can keep low order deformations (in the mm range due to gravity) at levels that are within the capture range of M4 [4].
- Stroke Management is a background task to redistribute the slowly building non-zero-mean components in low order modes of M4 to other degrees of freedom:
  - tip and tilt modes are controlled in a collaborative control scheme together with M5 and Telescope Main Axes, which desaturates M4 eventually through a Main Axes guide correction. This process is referred to as Field Stabilization [5,6].
  - focus and coma are transferred to M2 occasionally, at most every 5 minutes [7].
  - higher orders are continuously offloaded to M1, essentially by commanding the low pass filtered modal amplitudes accumulated on M4 to the M1 figure loop control system.

Moving optical surfaces changes the wavefront error measured by the Adaptive Optics system and causes the M4 to desaturate, i.e. saturation management requires a closed on-sky loop. Stroke management redistributes stroke between the optics degrees of freedom but does not aim on maintaining the telescope at its prescription [7].

While the control strategies outlined above are simulated [8] and tested [9,10,11] in detail, it is important to note that the unprecedented size of ELT is expected to lead to surprises during commissioning. Changes in the control strategy are hence expected, and therefore the control system is built such that well understood subsystem control is decoupled from less defined high-level (on-sky) control, and the latter one is developed in a way that allows for the flexible adjustment of algorithms during commissioning.

CONTROL SYSTEM ARCHITECTURE

The ELT Control System architecture has been described in [12] and since then we have consolidated and further developed it in the details, using the feedback from prototypes and the first applications. Here and in the next sections we will describe just some key aspects to have a context and we will describe the most important developments since the publication of [12].

The architecture enforces a distinction between a telescope device (*the System Under Control - SUC*, e.g. the M2 unit) and the component controlling that device (*the Control System - CS*). This terminology is adopted from the State Analysis (SA) methodology developed at JPL [13].

As a system of systems, the ELT contains layers of controllers. A lower level component comprising a CS and SUC appears as a SUC to a higher-level CS. For example, the primary mirror segment position actuators (PACT) with embedded position controller and course and fine stage actuators appear as components of the SUC to the M1 CS responsible for figure control.

Figure 2 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 Fig. 2-# in the text below).

The first breakdown of the ELT CS is into the many individual control systems associated with Telescope subsystems (called the Local Control Systems (LCS)) (Fig. 2-1), and the single system that integrates these, termed the Central Control System (CCS) (Fig. 2-2), whose internal structure is described in the corresponding section below.

The LCS-CCS discrimination not only separates unit-level from telescope-level safety and control, it also matches organizational boundaries in-line with the ELT procurement strategy: individual subsystems (mirror units, main structure, …) are designed, built and delivered by industrial partners; their integration is ESO responsibility.

On the other side of CCS are instruments (Fig. 2-3), developed by Consortia of ESO partner institutes. Each instrument includes an independent Instrument Control System (ICS) developed following the ELT standards, and interfacing with the telescope through the CCS interface, split over a control and a deterministic network (Fig. 2-4).

The interfaces between LCSs, CCS, and instruments are defined in a series of ICDs specifying the logical addresses, data types, formats, rates and characteristics of the data communication. The same applies to the interfaces between the internal components of CCS, even if that is fully developed inside ESO. This strict interface management is key, given the distributed nature of the ELT CS and the range of developers and suppliers.

From State Analysis we have also adopted the *State Variable* and *Estimator-Controller-Adapter* patterns[13].

The term *State Variable (SV)* refers to an element of the CS that represents a physical state of the SUC. For example, a limit switch in the SUC will physically be in an opened/closed state and the CS will use evidence such as sensor measurements to *estimate* the state of the switch as opened/closed. SVs are observable by clients.

The *Estimator-Controller-Adapter* pattern is based on:

- The Adapter that allows to communicate with the SUC to command and measure.
- The Estimator that receives measurements from the SUC via the Adapter and computes State Variables. Estimators do not send commands to the SUC.
- The Controller that is responsible to control the SUC via the Adapter. It can access State Variables if needed.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL01

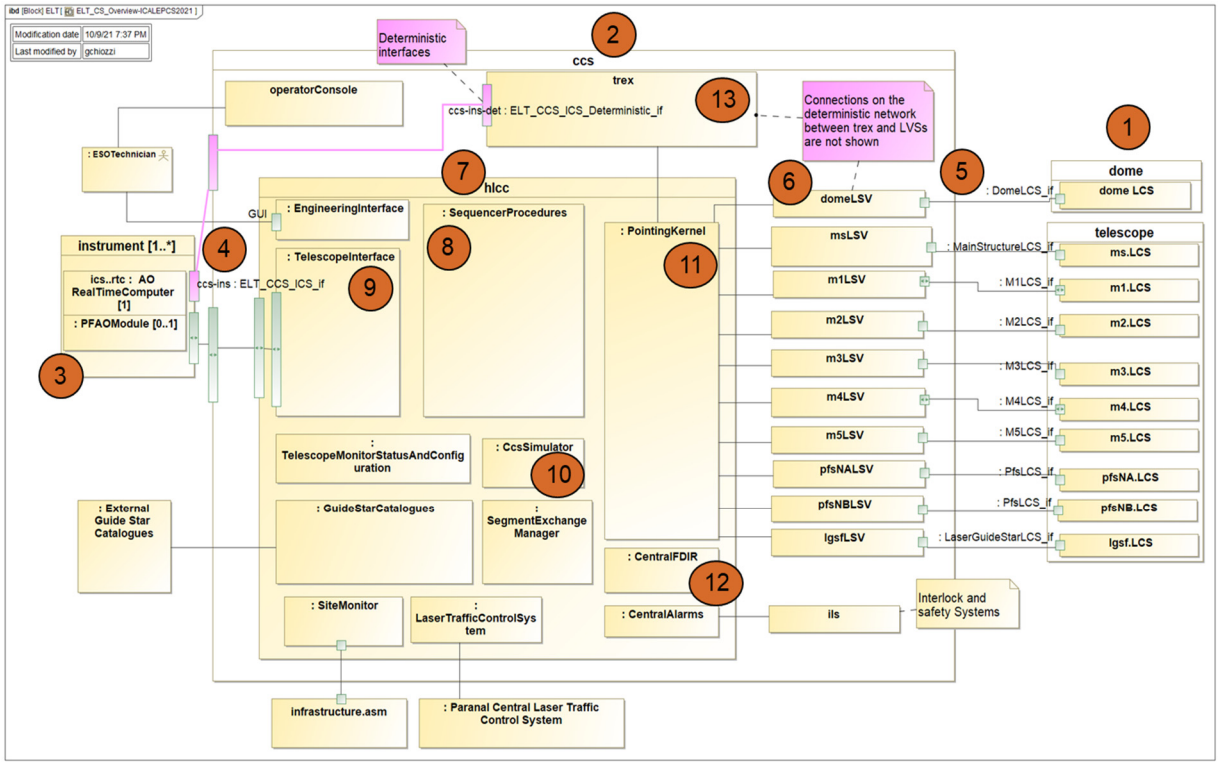

Figure 2: ELT Control System overview.

Commands sent by a Controller affect the CS and therefore the value of State Variables.

We have decided (after analysis and prototyping) not to adopt *SA goal-based operation*. This would have changed radically our way of viewing the system with respect to previous projects and the experience of the astronomers developing observations. Since we do not have the requirement to develop an autonomous system (where goals-based operation is a major advantage), we believe that the cost of a paradigm shift would have not been compensated by the benefits.

LOCAL CONTROL SYSTEMS

A Local Control System (LCS) contains two key functional groups: control and safety. Control functions enable standard operations, while safety functions preserve integrity and guarantee safety of personnel and equipment.

The control functions include Control Software, Local Control Unit(s), remote IO and a local communication.

The safety functions include Safety Logic, Local Safety Unit(s), Safety IO devices and a fail-safe communication.

The LCS enables safe control of the functions of the associated subsystem (e.g. M2 mirror cell). The functions provided by the LCS can make no assumptions as to the nature of the subsystem use in the context of the telescope control system (the operation and wave front control strategies being implemented by the CCS). For example, the M4 adaptive mirror has to be available and operable to full performance irrespective of whether or not the telescope is observing or calibrating or parked.

The LCS provides interfaces to CCS that enable individual and independent control of subsystem devices and functions (Fig. 2-5). For example, it must be possible to move the warping harness of an M1 segment irrespective of the status of edge sensors and position actuators.

The common characteristics of LCSs and the specific example of the M1 LCS, that is developed directly at ESO as well as the Laser Guide Star (LGS) LCS, have been described with more details in [14]. All other LCSs are being implemented by the contractor responsible for each individual subsystem; at present time, most telescope subsystems have passed final design review and are in different stages of implementation. Each contractor is followed-up by a member of the ESO CCS development team. Particularly important is the definition and consolidation of the ICDs between each LCS and CCS as the detailed design and implementation progress.

CENTRAL CONTROL SYSTEM

The Central Control System (CCS) integrates the many Local Control Systems 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 tools belong to CCS. CCS is therefore the primary user interface to operate the Telescope.

CCS (developed in-house, to leverage our specific astronomical expertise) is responsible for coordination and for all what concerns the astronomical domain.

CCS applications are organized in a shallow hierarchy of loosely coupled cooperating components as can be seen in (Fig. 2-2). All applications are based on a common software infrastructure (see [12] and the section below) and

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL01

in particular on the Rapid Application Development (RAD) framework [15]. RAD helps in the development of event driven applications by imposing a common design and providing tools to quickly produce application skeletons ready to use. RAD applications are built around a BOOST ASIO event loop [16] integrated with a SCXML state machine interpreter [17]. Anonymous publisher-subscriber communication and shared access to an Online Database are used to keep the coupling as loose as possible.

LSVs

Within CCS, a Local Supervisor (LSV) (Fig. 2-6) is a software component that provides access to specific Local Control System (LCS) functionalities. The LSV is responsible for implementing the telescope domain logic and translating the telescope concepts into the device domain handled by the corresponding LCS. For example, the M1 LSV is responsible for controlling and maintaining a certain optical quality of the whole telescope primary mirror surface while the M1 LCS is managing the actuators and sensors installed on each segment of the mirror.

There is one LSV for each of the following telescope LCSs: M1, M2, M3, M4, M5, Pre-Focal Stations, Main Structure, Dome, and Laser Guide Star. Each is independent from the other LSVs and can use only the services of the corresponding LCS. For instance, M1 LSV is not allowed to communicate directly to M4 LSV and can send commands only to M1 LCS.

The services provided by an LSV are grouped in "subsystem functions". Each subsystem function corresponds to a set of functionalities implemented by the LCS. For example, since the Main Structure LCS allows to position the altitude and azimuth axes independently, the Main Structure LSV provides two subsystem functions, one to deal with altitude and one for azimuth. For each subsystem function an operational state is estimated by the LSV using the published LCS measurements. The subsystem functions are supposed to be, as far as possible, independent (e.g. it should be possible to use altitude axis even if azimuth is not available and vice-versa).

Each subsystem function is made of an adapter library, one or more estimator applications, and zero or more controller applications following the Estimator-Controller-Adapter design pattern. Each estimator and each controller are implemented by a dedicated application. LSV estimator and controller applications share the same architecture to allow faster development and easier maintenance.

Common requirements across all LSVs and specific requirements have been identified, a set of common design patterns has been drawn and implemented in pathfinder LSV applications. Detailed design for the individual LSVs is being analyzed and the development has started and will be the main activity for the coming years until deployment in the ELT Control Model[12] and at the telescope.

HLCC

The High-Level Coordination and Control (HLCC) software layer lies above the LSVs (Fig. 2-7). It offers a single interface of the whole telescope toward operators and the instrument control software. Its main task is for supervisory applications to coordinate the various telescope subsystems.

The main challenge for HLCC is to implement a well-structured system that at the same time can be modified to a large extent during telescope commissioning.

To reach this objective we have identified the building blocks that can be seen in Fig. 2. An important role will be played by the SequencerProcedures (Fig. 2-8). These will be developed around *features*[12], independent supervisory applications designed to perform a complete operational function/use case of value to the users of the system.

We have planned for a long period of integration and commissioning[18], during which we will discover how to operate our machine and how the elementary functions provided by the LSVs will have to be composed together. Implementing *features* as independent components, using an interpreted language (Python) accessible to the commissioning team (not necessarily SW developers) allows us to evolve them in an easy way, with minimal impact on other *features*.

In the last two years we have developed a prototype to validate our architecture and design with respect to the software infrastructure and the application framework. We chose a vertical slice from a dashboard GUI down to several services representing a telescope tracking the sky. Here we give a few examples of design choices that differ from existing software at other ESO telescopes.

Measurement data was treated using estimators subscribing to the publishers that deliver input data to them, processing this data in Java code or in external Python scripts (Jep framework[19]), and pushing results out through StateVariables. Estimators form a hierarchy, e.g. with 2 estimators for incoming Alt and Az positions of the telescope, and downstream estimators that combine or convert this data. The use of pub-sub communication makes deployment of estimators in one or many processes a flexible choice. We got smooth data flow through these estimators when feeding them with simulated data at 20 Hz. One of the estimators for derived data produced the telescope's actual state, e.g. as "tracking" or "moving".

The first HLCC prototype of the RAD applications was implemented in Java, that we considered better suited than C++ for high-level coordination, without demanding performance requirements.

For the current implementation of the HLCC applications, we decided to change from using Java and Python to using C++ and Python. Java worked very well for the prototype, and likely would have worked well also for the final applications. But LSV and instrumentation applications are being developed in C++ to leverage the huge experience in the ELT software development team and to avoid potential performance problems coming from JVM garbage collection, which could introduce unwanted jitter to applications. We hope that the lower efficiency we experience working with C++ will be compensated by synergies and code reuse across subsystems as well as less maintenance in the lower level infrastructure software through decreased support for the Java language.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL01

We identified the HLCC interfaces and applications in a first design iteration and we have started now with the implementation. The Telescope Interface implements the ICD between all of CCS and the instruments. All requests from Telescope Interface will be served by delegating to other HLCC or LSV applications or to the Telescope Real-time Executor (TREx), described below.

As an alternative, especially during early development when these other applications do not yet exist or do not yet provide simulation capabilities of their own, the TelescopeInterface (Fig. 2-9) will delegate to the CcsSimulator (Fig. 2-10). The CcsSimulator will be also delivered to the consortia developing instruments, to have a frontend for testing their interactions with CCS.

In addition to the request-reply and publish-subscribe communication described in the ICD, instruments, as well as operators, will be able to access dedicated telescope data published in the Online Database.

The PointingKernel (Fig. 2-11) application controls all telescope subsystems. It interacts with TREx for most of the pointing logic, but also commands the LSVs directly.

The CentralFDIR (Fig. 2-12) application monitors those aspects of quality and failures that involve more than a single subsystem.

Other dedicated applications exist for star catalogues, monitoring and configuration, alarms, or specific tasks such as segment exchanges.

TREx

Control loops with demanding real-time requirements described in the Control Strategy section are not part of HLCC; instead, they are allocated to the Telescope Real-time Executor (TREx) (Fig. 2-13), which communicates directly with LSVs and LCSs using, when necessary, the deterministic network (not shown in the figure). HLCC commands and monitors TREx.

This component of the system is now in the requirements' collection phase. What is clear is that the WFC and stroke management strategy and algorithms will, for a big part, be developed as part of commissioning, while gaining experience on the as-built telescope.

For this purpose, a flexible software framework for real-time control applications is required. The framework must be suitable for use by control engineers and a visual programming environment, comprising a palette of data analysis and algorithmic building blocks, is foreseen. The framework should not require detailed knowledge of real-time systems when mapping the application to the underlying hardware resources.

A demonstration prototype was developed in 2019 based on GNU Radio[20], a free open-source development toolkit to implement signal processing tasks. Although GNU Radio is not built with low-latency and control systems in mind, it seems possible to overcome the limitations with minor adaptions and by tailoring our use to the pure algorithmic domain. CS specific functions (e.g. I/O, monitoring, error handling and recovery) should be handled in a separate entity that interfaces the real world with GNU Radio. GNU Radio comes with a GUI usable by non-software engineers and integrates well with C++ and Python.

Results with a computational load comparable to TREx show that 1kHz loop rates are reliably achievable. Measurements indicate that 2kHz loop rates may be possible.

RTC AND ADAPTIVE OPTICS

The core component of any AO system is the Real Time Computer (RTC) which measures the incoming wavefront aberrations by means of sensors and corrects for them by means of a deformable mirror.

During scientific observation, the ELT telescope performs only guide probe AO, i.e. measurements done with a wavefront sensor (WFS) installed on the pre-focal station (PFS) and limited to the first few modes are used to reject low and mid spatial/temporal frequency wavefront. This is required to provide an image quality sufficient for handover to an instrument. Guide probe AO is performed by TREx, which stops performing it after INS handover; at this point the instrument drives the M4 to achieve the desired image quality [7].

There will be therefore one RTC per each instrument to implement high order AO modes and one RTC used on the telescope for commissioning and diagnostic (the Phasing and Diagnostic Station).

The ELT RTC architecture [21] is defined by ESO not only for the telescope, but also for the instruments, with the aim of streamlining development and leveraging re-usability. This architecture identifies two distinct components (Fig. 3).

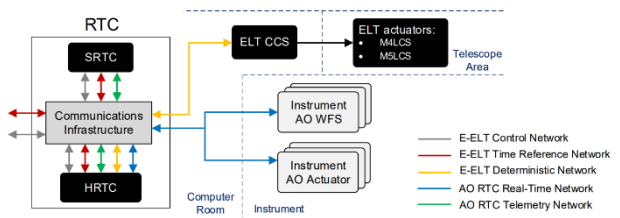

Figure 3: RTC architecture.

Each of them targets functions in a specific domain and timescale and follows its own technology roadmap.

The Hard Real-Time Core (HRTC) implements the main AO control loops, which perform demanding computations on incoming WFS measurements and command actuators within tight timing constraints. The HRTC is interfaced to the AO WFSs and actuators on the Instrument via a dedicated RTC Real-Time Network. The HRTC commands both the M4 and the M5 via the Deterministic Network.

The Soft Real-Time Cluster (SRTC) is a set of computing nodes in charge of the high-level supervision and optimization of the HRTC. It is driven by requests from Instrument Control Software (ICS), as well as by the reception and automatic processing of telemetry data (e.g. measurements, commands). Computations elapse from seconds to minutes and involve algorithms operating on large data sets. A dedicated AO RTC Telemetry Network

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL01

interconnects the HRTC and SRTC. ESO has defined standards for the SRTC technology and delivers the AO RTC Toolkit: common RTC functions are addressed by a suite of software tools, libraries and reference implementations.

The HRTC is on the cutting edge of technology, and therefore rather than standardizing technology, we have decided to specify only interfaces, to achieve full replaceability as mitigation of obsolescence.

Nonetheless, the HRTC prototype (HRTCp) has been developed to explore the feasibility of building an ELT-size, AO RTC based on mainstream CPU and Ethernet technology - i.e. without accelerators or specific-purpose hardware. Key aspects to be assessed are modularity, scalability, long-term maintainability and upgradeability.

The HRTCp addresses a Multi-Conjugate AO (MCAO) configuration with six LGS WFS and adopts a functional breakdown that enables scaling it down to Single Conjugate AO (SCAO), such as the ELT Wavefront RTC (WFRTC), i.e. the RTC to be used in combination with the PFS and during commissioning for verification of telescope SCAO capabilities. The HRTCp operates at 500 Hz and implements a pseudo open-loop control (POLC) algorithm dominated by two memory-bound, Matrix-Vector-Multiplication (MVM) operations of size 6,316x55,392 per loop cycle. All the external interfaces use 10 GbE signaling to ingest an aggregate 44.4 Gbps incoming pixel traffic and produce 87 Mbps actuator commands and 2.7 Gbps telemetry data towards the SRTC. Simultaneously, the system continuously receives disturbance data at 1 Gbps.

The *as built* HRTCp is comprised of 14 mainstream servers: two front-end nodes (namely A and B) per LGS WFS, one shared back-end node and one low-level supervisory node. Front-end node A implements the MVM in the direct control path using 48 CPU cores on a dual-socket AMD EPYC 7742 server. Front-end node B is responsible for the MVM in the offline control path (burst computation) and employs 56 cores on a dual-socket AMD EPYC 7501 server. Aggregate, each pair of front-end nodes delivers in excess of 600 GB/s memory bandwidth and 300 GFLOP/s only for these operations.

The internal communications employ 10 GbE, except for the collection of results from the front-end A nodes by the back-end node. Early tests showed that this is a major serialization point (contributing a 100 us delay over 10 GbE), thus dictating the use of single 100 GbE link. With this, the current HRTCp end-to-end latency (measured from last pixel image packet received to last command packet sent) is 244 us. The associated deviation is below 4 us over 2-minute intervals, with maximum excursions of 25 us.

The key challenge has been to achieve the required performance using only hardware platforms and software techniques with long-term maintainability and upgrade paths. A key mandate was to widen the profiles that could contribute to RTC development by targeting knowledge domains (i.e. CPU-based computation and Ethernet networking) available at ESO.

The development was started in early 2018 and the system is currently in operation at ESO premises. Important picks from this prototype development are:

- Recent, mainstream CPU architectures targeting High Performance Computing and providing fine NUMA granularity can be leveraged to fit our problem.
- Deterministic networking can be achieved with the native Linux network stack, using interrupt routing and controlling packet coalescence. Deterministic, overall latency performance is possible exploiting real-time techniques such as NUMA affinity, core isolation, thread pinning and inter-thread polling.
- It is possible to write maintainable C++ software that implements the above techniques, with little or no explicit vectorization (i.e. offloading this to the compiler). The current trend for increased CPU core count in CPU families helps mitigate the architectural constrains derived from core isolation and thread pinning.

The HRTCp will be used as a flexible platform where upcoming CPU families and networking devices can be benchmarked. In addition, several of the HRTCp design choices will be at the core of the ELT WFRTC.

A subset of the techniques developed within the HRTCp scope have been ported to the VLT domain. An upgrade of the VLT SPARTA systems will replace the obsolete, FPGA-based real-time core with a single server, while respecting the legacy (non-Ethernet) I/O interfaces.

THE MINUSCULE ELT (MELT)

The Miniscule ELT (MELT) [9] 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.

The optomechanical setup uses the Active Segmented Mirror (ASM) with 61 piezo-driven segments and a diameter of 15 cm. It was used on sky on a VLT telescope during the Active Phasing Experiment (APE)[22].

Several beam paths after the optical train on MELT are conditioned and guided to wavefront sensors and cameras, sensitive to wavelength bands in the visible and infrared to emulate wavefront commissioning and phasing tasks.

In MELT, the ELT main axis control is emulated with a moveable diffraction-limited source that emits white light from the visible up to the K band through a turbulence generator. A single conjugate adaptive optics Shack Hartmann (SH) WFS is used in closed loop with an ELT RTC and M4 to test and validate offloading scenarios to M5 and the main axis. In addition, it is used to deploy and

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL01

validate wavefront control algorithms and the influence of AO on M1 phasing using the baseline SH high order WFS, but also M4 phasing issues with its petals, and scalloping. The bench also allows to test different phasing concepts.

The MELT Control System applies the same architectural concepts as the ELT. Furthermore, the central services developed for the ELT, are deployed in MELT as soon as they are released, such that their usage provides early feedback. MELT is therefore an excellent testbed also for the whole control SW architecture and tools.

The bench will help us to be as much as possible prepared when the telescope will send the star light through the optical train to be able to tackle the unforeseeable problems and not be caught up with the foreseeable ones.

COMMON SW INFRASTRUCTURE

The control software of the ELT is built on top of a common software infrastructure, comprising libraries, frameworks and development tools for building, deploying, documenting and testing the applications.

Among these, particularly important is the Core Integration Infrastructure (CII). CII comprises communication libraries, configuration infrastructure, network value cache, and logging infrastructure for control applications. The communication part includes an interface definition language for defining the request/reply interfaces as well as the publish/subscribe contracts, independent from the actual protocol (ZMQ, DDS, OPC/UA, and others) used. CII provides APIs in three languages (C++, Python, and Java).

CII relies on a number of third-party products. For example, the Online Database (a central value cache on the control network) keeps its data split over three third-party databases: elasticsearch, redis, minIO. While we benefit from the stability and feature richness of these products, mastering them is a challenge. We also often find the need of creating additional tooling, in particular when we aim to allow our users to handle them on their development set-ups without direct support from the CII team.

Navigating in the problem space of performance versus usability versus maintainability, we are frequently discussing whether a given usage pattern should be supported in the CII infrastructure layer or not. A clear cut would be desirable between functionality implemented in ground-layer infrastructure, versus middle-layer frameworks, versus higher-layer subsystems. Defining this cut, or clear criteria for it, is an on-going challenge.

CII has been developed by Cosylab for ESO, from a specification of about 800 requirements in three years; the first version is being adopted by the control subsystems.

Different subsystems have different usage patterns for using the CII software, and currently the main activity consists in adjusting functionality according to user feedback and moving features between software layers.

We have recently transferred the user documentation from documents (word / pdf) to a collaborative platform (GitLab / reST / Sphinx / Jenkins). This enables users to prepare and propose improvements, while still being governed by a well-defined process. This has well decreased the turn-around time for documentation updates and makes our documentation more helpful.

While CII gets more and more used in the control subsystems, we are getting ready to deal with the resulting higher amount of user feedback and support requests.

The development of GUIs is addressed instead by the Control UI Toolkit (CUT). CUT is a set of libraries, widgets and graphical design patterns tailored to the ELT Control System requirements. Qt is used for graphical rendering; Taurus [23,24] as an abstraction layer that provides a powerful MVC pattern specifically designed with control systems in mind. Additional custom widgets, utilities, color schemes, and documentation based in Taurus and Qt complete the GUI development environment.

Taurus was selected due to similarities in requirements to our own specification, offering:

- Extension capabilities: plugins for Taurus can be developed to enhance its communication capabilities, model access, plotting and image rendering.
- MVC design: allows decoupling widget and views development from models. While we develop models to access CII services, we can still continue development of application views using the "eval" model plugin, or any other plugin.
- Multiple expertise levels: developers in ELT varies in GUI software development experience. Taurus offers three ways to develop GUIs, from a very simple no-code approach, to complete freedom [24].
- Declarative binding: supports declaration in UI specification of binding between datapoints and widgets.
- Subscription, polling, filtering, customization and declarative configuration of UI.

Taurus is extensible by design, a fact that is appreciated and allows us to provide support for CII communication libraries and infrastructure as they are integrated in the development environment. At this moment, ELT includes Taurus in its Software Development environment, and we contribute our bug fixes and development directly to the upstream project. It also includes the Taurus CII Online Database Plugin. The Control UI Toolkit has been used until now to implement prototype and engineering GUIs.

CONCLUSION

The ELT Control System faces major challenges that are expected to be overcome by the development of a System of Systems flexible up to the commissioning phase. After the requirements and design phase and the development of the technical infrastructure and of (large scale) prototypes, we are now moving to the serial development of the actual system components. The LCSs, primarily developed externally by outsourced contractors, LSVs and HLCC, shall be integrated into one single System user interface. Specific validation test benches (MELT) are operational and following the evolution of the project.

The Scientific First Light for the ELT is foreseen for the end of 2027 and our schedule is in line with this target.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL01

REFERENCES

[1] H.Bonnet *et al*., "Adaptive optics at the ESO ELT," Proc. SPIE 10703, Paper 10703-10, 2018.

[2] R.Biasi *et al*., "E-ELT M4 adaptive unit final design and construction: a progress report", Proc. SPIE 9909, Paper 9909-7Y, 2016.

[3] P.T.Wallace, "A rigorous algorithm for telescope pointing," Proc. SPIE 4848, 2002.

[4] B.Sedghi, M.Müller, "Dynamical aspects in control of E-ELT segmented primary mirror (M1)," Proc. SPIE 7733, Paper 7733-2E, 2010.

[5] B.Sedghi *et al*., "Field stabilization (tip/tilt control) of E-ELT", Proc. SPIE 7733, Paper 7733-40, 2010.

[6] B.Sedhi, "Tip/tilt control strategies at ELT" Wavefront sensing and control in the VLT/ELT era, 3rd edition, invited talk, Paris, 2018 https://indico.obspm.fr/event/56/contributions/145/

[7] H.Bonnet *et al*., "Adaptive optics at the ESO ELT", Proc. SPIE 10703, Paper 10703-10, 2018.

[8] B.Sedghi *et al*., "E-ELT modeling and simulation toolkits: philosophy and progress status", Proc. SPIE 8336, Paper 8336-06, 2011.

[9] T.Pfrommer *et al*., "MELT: an optomechanical emulation testbench for ELT wavefront control and phasing strategy", Proc. SPIE 10700, Paper 10700-3F, 2018.

[10] M. Dimmler *et al*., "E-ELT M1 test facility", Proc. SPIE 8444, Paper 8444-1Y, 2012.

[11] H.Bonnet *et al*., "Fast optical re-phasing of segmented primary mirrors", Proc. SPIE 9145, Paper 9145-1U 2014.

[12] G.Chiozzi *et al*., "The ELT Control System", Proc. SPIE 10707, Paper 10707-31, 2018.

[13] 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

[14] L. Andolfato *et al*., "The ELT M1 local control software: from requirements to implementation, Proc. ICALEPCS2019, New York, USA, 2019.

[15] ELT ICS Rapid Application Development (RAD), http://www.eso.org/~eltmgr/ICS/documents-latest/RAD/sphinx_doc/html/

[16] Boost asio, https://www.boost.org/doc/libs/1_77_0/doc/html/boost_asio.html

[17] State Chart XML, https://www.w3.org/TR/scxml/

[18] R.Tamai *et al*., "The ESO's ELT construction progress", Proc. SPIE 11445, Paper 114451E-16, 2020.

[19] JEP – Java Embedded Python, https://github.com/ninia/jep

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

[21] M. Suárez Valles *et al*., "Adaptive Optics Hard and Soft Real-Time Computing Developments at ESO," AO4ELT6 Adaptive Optics for Extremely Large Telescopes, Quebec, June 9-14, 2019.

[22] F.Gonte *et al*., "APE: the Active Phasing Experiment to test new control system and phasing technology for a European Extremely Large Optical Telescope," Proc. SPIE 5894, Paper 5894-0z, 2005.

[23] C.Pascual-Izarra *et al*. "Taurus big & small: from particle accelerators to desktop labs," in Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17), Barcelona, Spain, Oct. 2017, pp. 166-169. doi:10.18429/JACoW-ICALEPCS2017-TUBPL02

[24] C.Pascual-Izarra *et al*., "Effortless creation of control & data acquisition graphical user interfaces with taurus," in Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), Melbourne, Australia, Oct. 2015, pp. 1138-1142. doi:10.18429/JACoW-ICALEPCS2015-THHC3003