PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

The ELT high level coordination and control

Gianluca Chiozzi, Nick Kornweibel, Ulrich Lampater, Babak Sedghi, Heiko Sommer

Gianluca Chiozzi, Nick Kornweibel, Ulrich Lampater, Babak Sedghi, Heiko Sommer, "The ELT high level coordination and control," Proc. SPIE 12189, Software and Cyberinfrastructure for Astronomy VII, 1218919 (29 August 2022); doi: 10.1117/12.2630435

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2022, Montréal, Québec, Canada

The ELT High Level Coordination and Control

Gianluca Chiozzi[1], Nick Kornweibel, Ulrich Lampater, Babak Sedghi, Heiko Sommer
European Southern Observatory, Garching bei München, Germany

ABSTRACT

The Extremely Large Telescope (ELT) is a 39-meter optical telescope under construction in the Chilean Atacama desert. The optical design is based on a five-mirror scheme and incorporates adaptive optics. The primary mirror consists of 798 segments. Scientific first light is planned by the end of 2027. The status of the project is described in [1].

The major challenges for the control of the telescope and the instruments are in the number of sensors (~25,000) and actuators (~15,000) to be controlled in a coordinated fashion, the computing performance and low latency requirements for the phasing of the primary mirror, performing adaptive optics and coordinating all sub-systems in the optical path.

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

In this paper we focus on architecture and design of the High-Level Coordination and Control (HLCC). It is the component of the control software responsible for coordination of all telescope subsystems to properly perform the activities required by scientific and technical operations.

We first identify the HLCC context by introducing the global architecture of the telescope control system and by discussing the role of HLCC and its interfaces with the other components of the control system.

We then analyze the internal architecture of the HLCC, and the primary design patterns adopted.

We also discuss how the features identified from the requirements and the use cases are mapped into the design.

Finally, the timeline and the current status of development activities are presented.

Keywords: ELT, telescope control software, architecture, design

1. INTRODUCTION

The ELT is a large segmented telescope. Significant wavefront perturbations are induced first by the atmosphere and then by the telescope itself (deformation through gravity, temperature, and wind loads). 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 passed over to the instruments. This means that the "spectrum of wavefront aberrations induced by the observatory is below that of the free atmosphere." [2].

The aim of the ELT is to provide a wavefront with an error in the range of tens 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 closed loop with a large temporal and spatial bandwidth. This is possible thanks to 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 [5]. The limited stroke (100um) of the M4 actuators and the limited capture range of the wavefront sensors already exclude that the wavefront can be controlled solely by M4. Even if this was feasible, it would not be possible to fix the field properties of the telescope by the M4 alone. M4 requires therefore the support of several additional control systems [4]:

- Feed-forward control is used when on-sky loops are open, to preset the telescope to a new target, and as an underlying first order system when tracking. The feed forward model takes into account astrometry and telescope deformations due to gravity and temperature to position the telescope within the acquisition range of on-sky sensors [6].

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

Software and Cyberinfrastructure for Astronomy VII, edited by Jorge Ibsen, Gianluca Chiozzi, Proc. of SPIE Vol. 12189, 1218919 · © 2022 SPIE
0277-786X · doi: 10.1117/12.2630435

- 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 798 hexagonal segments of the segmented M1 are moved 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 keep low order deformations (in the mm range due to gravity) at levels within the capture range of M4 [7].
- 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 and M5 eventually through a Main Axes guide correction. This process is referred to as Field Stabilization [8][9].
    - focus and coma are transferred to M2 occasionally, on a timescale of a few minutes [2].
    - 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.

The coordination of all the subsystems involved in these activities and the implementation of the high level strategies that define their interactions is the responsibility of the High Level Coordination and Control (HLCC) and of the Telescope Real-time Executor (TREx) components of the Central Control System (CCS)

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 [3] 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 describe just some key aspects to have a context and we summarize the most important developments since the publication of [3].

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*).

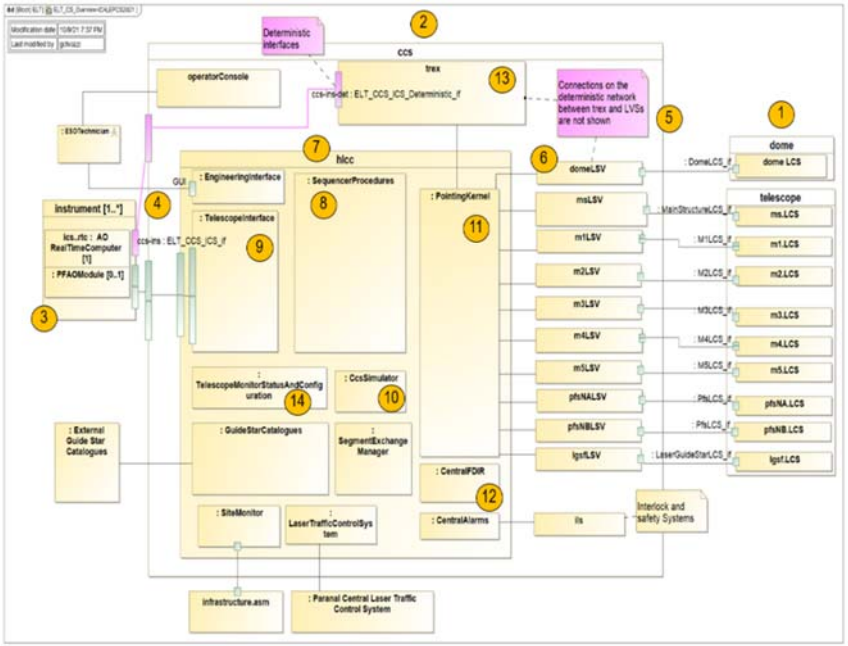

Figure 1: ELT Control System overview

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)), whose internal structure is described below.

A subsystem is responsible for autonomous control, monitoring and safety through the LCS. An LCS is integrated into the ELT Control System via its LCS-CCS interface, implemented via a suite of standard protocols running on switched Ethernet.

The LCS-CCS discrimination separates unit-level from telescope-level safety and control, but it also matches organizational boundaries in-line with the ELT procurement strategy: individual subsystems are designed, built and delivered by industrial partners; their integration is the responsibility of ESO.

A subsystem is presented to the 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). The CCS component integrating the LCS with the CCS is termed the Local Supervisor (LSV). It is within the LSV that the Subsystem Functions are implemented (either via interaction with the LCS, or within the LSV itself) and where the knowledge of the astronomical domain resides. For example, sky tracking algorithms are implemented inside the Main Structure LSV code, that will commands 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 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 (Figure 1-(4)).

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 tools belong to CCS.

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

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*).

All CCS software is built following a set of standards covering hardware, software, documentation, testing, coding and development process. Part of these standards is global use of the Core Integration Infrastructure (CII) [3] for interfacing with generic control system services such as: communication middleware, logging, configuration, online database and error handling. All applications are also based on the Rapid Application Development (RAD) framework [11]. 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. Anonymous publisher-subscriber communication and shared access to the online database are used to keep the coupling as loose as possible.

3. CCS INTERFACES AND OPERATION WORKFLOW

The diagram in Figure 2 takes key components from the ELT Control System (CS) and places them in an interface block diagram where the interfaces relevant to the telescope during observation are identified.

The LSVs publish the estimated state of their associated telescope subsystem as a set of State Variables.

The term State Variable (SV) refers to an element of the CS that represents a physical state of the SUC[14]. 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 subsystem states may be manipulated through a supervisory command interface exposed on the control network by the LSVs.

The Main Structure, Dome, Prefocal Station (PFS) and Laser Guide Star LSVs track sky coordinates and thus run pointing kernels publishing actuator position references at 20Hz. The Main Structure (Alt/Az axes), and Prefocal Station (guide probes) may receive deterministic input into the pointing kernel in the form of adjustments to the target position on the

focal plane (from the instrument) and the M4 and M5 offload to the Main Structure axes (from TREx as part of guiding and stroke management).

The M4 Adaptive Mirror and M5 Tip-Tilt stage are driven by TREx at rates up to 1kHz. To minimize latency, the commands from TREx are published directly to the M4 and M5 Local Control Systems (LCSs), bypassing the respective LSVs.

System-level conditions of the telescope are estimated from the set of subsystem State Variables. These conditions are summarized by the Telescope Monitor, for example: “presetting”, “guiding” “ready for INS hand-over”, “observing”. Such conditions are not mutually exclusive, and they do not inhibit the function of the telescope, rather they are used to inform the operators and coordinate observations with the instruments.

The telescope subsystems may be initialized and transitioned into night operations by the operators using interactive command sequences executed under the control of an interactive sequence execution engine termed the Sequencer (Figure 1-(8)).

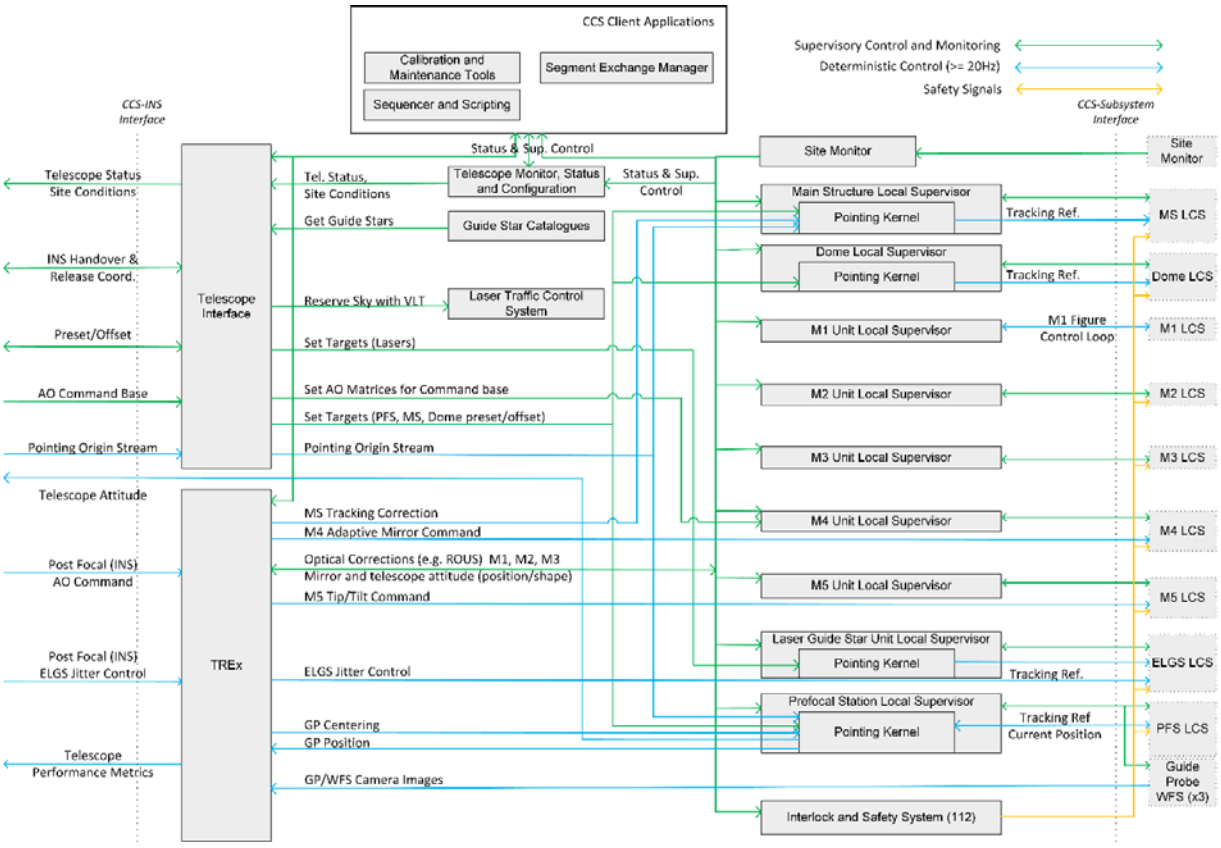

Figure 2: CCS interfaces

ESO technical staff (i.e operators, AIV and commissioning scientists and engineers, maintenance engineers or the software development team) can interact with the telescope using the operator consoles and engineering GUIs or through Python scrips run through the Sequencer or Jupyter-notebooks.

With the observing doors open and necessary subsystem functions available and enabled, the telescope is ready to observe.

The primary users of the telescope are the instruments and, through them, the astronomers preparing the observations to be run.

An instrument commences using the telescope by sending the observation specific parameters to the CCS: target, guide stars, observing wavelength, AO command base, etc, after which it commands the telescope to Preset. The guide probe

(GP) cameras are set up according to the expected guide star characteristics. TREx is configured as to which GP will be used for guiding, the AO command base, and other observation parameters. Likewise, the M4 Adaptive Mirror is loaded with the AO command base.

The telescope interface, upon reception of preset, will forward the preset coordinates to the Dome and Main Structure pointing kernels, running in the respective LSVs, which commence publishing position and trajectory demands at 20Hz to the telescope and dome axes (driven by the corresponding LCSs). The PFS LSV will receive coordinates of up to three guide stars (queried from the catalogue, if not provided by the observation) and will likewise commence publishing 20Hz position and trajectory data to the guide probe arms (driven by the PFS LCS), and North Angle and Pupil Angle at 20Hz to the INS-CCS interface for consumption by instrument tracking functions such as rotators, pupil steering mirrors and pointing origin stream. At the time of preset, the instrument will commence publishing its pointing origin stream (i.e. the coordinates that define the projection of the instrument's derotator center onto the PFS straight through focal plane), which is received by the Main Structure and PFS pointing kernels.

When the guiding-designated GP (typically the GP on the brightest star) images the guide star, the exposure (made available to the operator via TREx) may be adjusted through usual camera controls such as integration time and windowing, to improve the image quality. In addition to image display, TREx is producing image quality metrics, tip, tilt, and rotation error estimates based on the three guide stars and publishing these to the CCS (and INS interface). After the operator-selected guide star is centered on the GP reference pixel, TREx begins computing and publishing tip-tilt commands to M4, M5 and MS (via a collaborative control scheme[9]) and guiding commences, with the loop closed on the brightest guide star.

Once guiding, TREx will use sampled images and telescope metrology (published by LSVs state variables) to calculate periodic corrections to the telescope optics. The primary goal of these corrections is to maximize stroke on the M4 Adaptive Mirror, while improving image quality to a level suitable for instrument AO to begin. The instrument is receiving light at all stages and therefore it may commence its setup and acquisition process, but the telescope is not committed to deliver and maintain a defined image quality or to accept requests from the instrument in this optimization phase. After one or more iterations, the CCS will indicate "ready for handover" to the instrument, at which time the instrument may conclude its acquisition and is ready for AO.

The instrument performs its acquisition procedures concluding in a deterministic signal exchange between the instrument AO RTC and TREx, whereupon TREx switches from guiding (based on PFS GP-derived errors), to processing of the instrument-published AO commands. The handover lasts a few milliseconds at which time no corrections are sent to M4 adaptive mirror, M5 tip-tilt and MS axes (MS is still tracking). After successful handover the instrument is performing guiding and higher-order AO corrections, and TREx is responsible for mirror stroke optimization, primarily through its collaborative control scheme. The PFS guide probes continue tracking the guide stars, the guiding corrections derived from the PFS guide star may or may not be incorporated into the collaborative control scheme by TREx depending on the mode selected by the instrument: in "Cascade" mode the GP guiding corrections are incorporated with guiding corrections from the instrument; in "Sequential" mode, then PFS-derived guiding errors are ignored with tip-tilt and higher order AO modes fully driven by the instrument.

At any time after handover, whether requested by the instrument, the telescope operator, or as a result of an internal alarm (e.g. optical unit saturation), the telescope may ignore (not apply) instrument AO corrections and switch back to the PFS-derived guiding corrections. This event and condition are communicated to the instrument both over the deterministic network (to the instrument AO RTC) and the control network (to the instrument supervisory software), where instrument acquisition may be repeated if required.

The transition through acquisition to CCS-driven wavefront control onto INS-driven wavefront control and back is shown in the activity diagram in Figure 3.

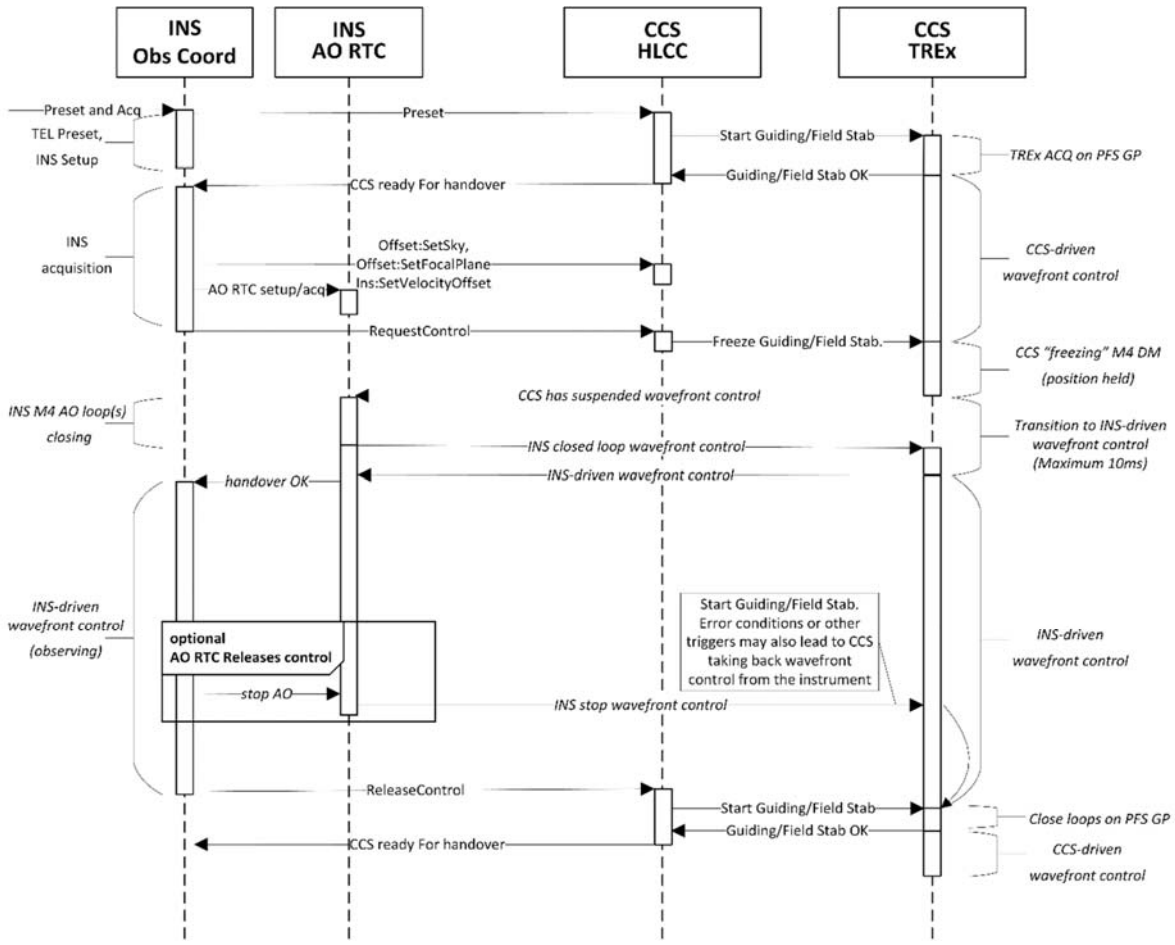

Figure 3: Transition through acquisition to CCS-driven wavefront control onto INS-driven wavefront control

4. HIGH LEVEL COORDINATION AND CONTROL (HLCC)

The HLCC offers a single interface of the whole telescope toward operators and the instrument control software, except for deterministic interfaces that are provided by TREx. 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. We foresee a long period of integration and commissioning, during which we will discover how to optimize the operation of our machine and how the elementary functions provided by the LSVs will be better composed together.

4.1 HLCC Deployment

The HLCC is C++, Python, and Qt software for Linux (currently CentOS but moving to Fedora) which will be executed on a mix of physical and virtual machines in the Armazones Computer Room. The software has only low requirements on determinism and synchronization, allowing hosting virtual machine instances on ESX servers with NTP time synchronization. Where greater determinism is required (for example for pointing kernels transmitting UDP data at 20Hz), physical machines with network connections to the Deterministic Network, and Time Reference Network (PTP) will be

used. UIs will be displayed on terminals in the Armazones Local Control Room, remotely in Paranal Control Room, or on other terminals in the telescope area if required.

Most of the HLCC interfaces to Subsystem Control Systems are present in the Computer Room over 10GbE. Where data is exchanged with subsystems in the telescope area (e.g. PLCs of an LCS), an infrastructure of single and multimode fibers is available.

4.2 HLCC architecture

The HLCC is architecturally composed of several functional building blocks, as can be seen in Figure 1. Among these:

- The interface toward the instruments, defined in the CCS-INS ICD [17], is allocated to the single component in Figure 1-(9), whose responsibility is to filter, validate and re-dispatch all commands received. This gives the flexibility to modify the internal structure without impacting the clients. All interfaces in the ELT are defined using standard structuring conventions [10].
- In order to provide maximum flexibility during AIV and Commissioning, *features* of the system (see also section 7) 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 managed by the SequencerProcedures module (Figure 1-(8)) and are written and executed using the Sequencer tool and are decoupled from the rest of the system. In this way also members of the AIV and commissioning teams can directly modify them.
- The Telescope Monitor Status and Configuration component (Figure 1-(14)) is responsible for monitoring all components of CCS and consolidating this information to build global status indicators. It manages requests for high level changes in the configuration, propagating information to individual subsystems. It makes high level status information available to human users as well to other systems, instruments in particular.
- The PointingKernel (Figure 1-(11)) component coordinates and interacts with the pointing kernel components in dome, main structure, PFS and laser systems. It interacts with TREx for most of the pointing logic, but also commands the LSVs directly.
- The CentralFDIR (Figure 1-(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.

As a general architecture concept at ELT CCS level,

- Instructions are given to components using a command/reply pattern, but the reply is only used to acknowledge proper reception of a command.
- A client shall infer the proper execution and completion of a request from status information available through a publish/subscribe pattern.

For example, an instrument will send a Preset command to the HLCC Telescope Interface to request pointing and tracking to a new target in the sky and receive an OK if the target is valid. It will know that the telescope is on target and ready for the instrument to start exposures when the "*ready for handover*" status information is published.

The HLCC architecture relies on the *Estimator-Controller-Adapter* patterns [14], where the estimation of the status of the system that we are controlling (the system under control) is kept clearly separated from the control tasks.

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

4.3 HLCC design and patterns

Each HLCC component is implemented as one or more software processes, developed in C++ around a state machine engine provided by the RAD framework [11] in combination with CII.

We describe below some characterizing elements of the design (used not only by HLCC but also by other CCS components.

The software stack of an HLCC process is shown in Figure 4.

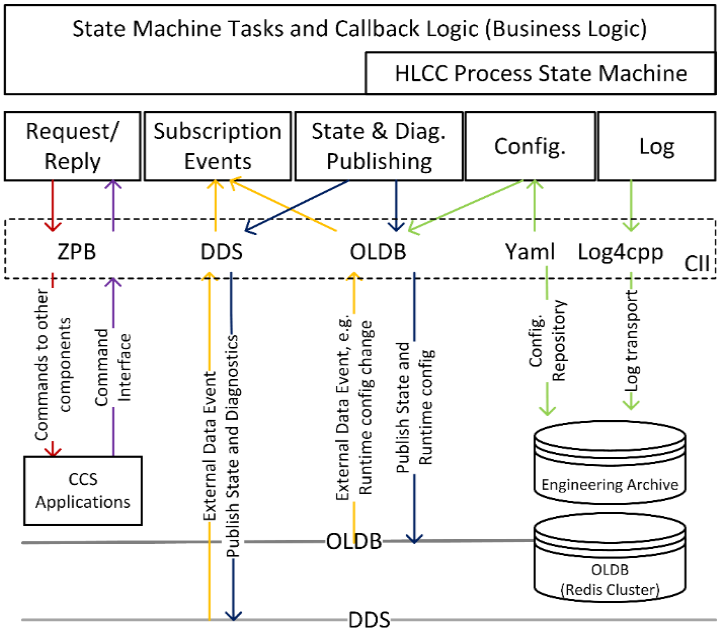

Figure 4: HLCC process software stack

State machine

All RAD applications in the ELT implement the same basic state machine [12] and a common set of standard commands defined in the ICD. The state machine is then extended to the functionality of the specific application (events and transitions). The challenge here is to ensure that the extended state machine remains compatible with the standard state machine; in simple cases this can be done by adding sub states or parallel regions to the existing ones, but little exists in literature and as tools to handle more complex situations: see [13] for an analysis and to see how we are addressing this issue.

When a command or any other external event is received by the application, it is translated into an event for the state machine, that in turn will execute the functions associated with transitions, state enter/exit actions and do activities. Figure 5 shows an example of standard state machine extended to cover the specific needs of one of the processes in the CCS Simulator, which provides a simulation of the telescope ICD toward instruments and is described with more details in section 4.4.

From the model of the state machine in SysML, an SCXML file is generated and it is loaded and interpreted at run time by the RAD engine [18].

The code is structured in separate small classes for Actions, Activities, Estimators, guard Conditions, making is easy to modify and extend the behavior of the application (even at run time, since the state machine is interpreted). We have seen that designing the logic of state transitions using a graphical and formal modeling tool and keeping this logic separate from the code if very useful to discuss the behavior of the system and to keep the code simple. One clear example is the handling of commands that can be executed only in certain conditions: when these conditions are mapped to states and the commands are defined by events only associated to the allowed states, the implementation is transparent and clear to anyone looking at the model.

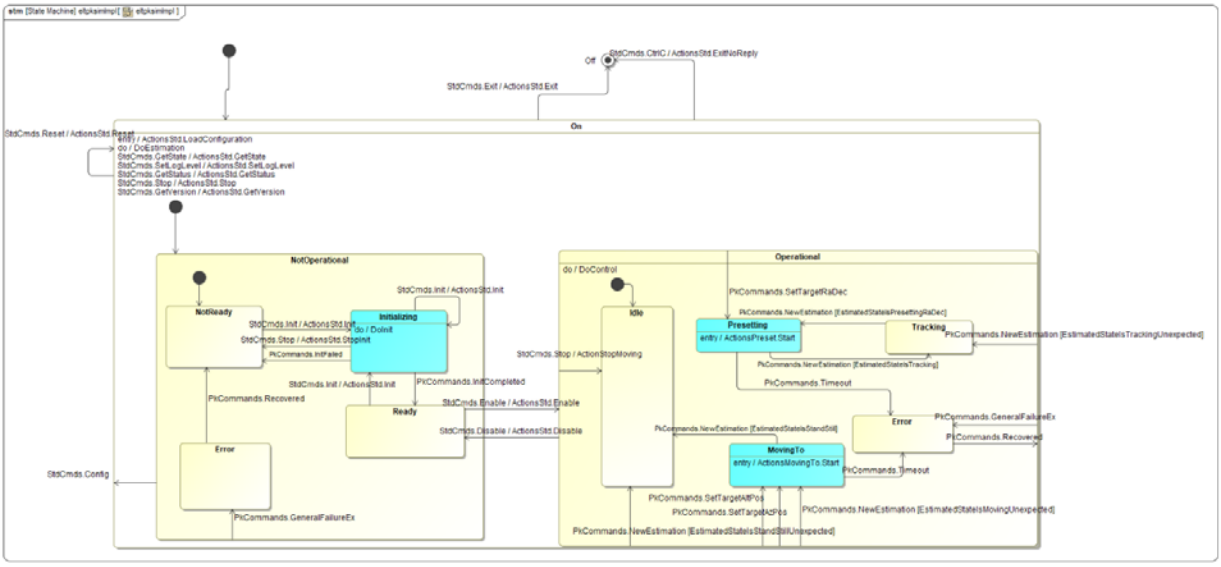

Figure 5: State machine for the CCS Simulator pointing kernel process

Actions and Activities

While Actions are supposed to be *instantaneous* and can therefore be executed in the main thread of the RAD engine (it is the responsibility of the developer to ensure that Actions are really considered *instantaneous* on the timescale of the application), Activities last instead for the whole time the system is in the corresponding state.

Therefore, Actions are spawned in RAD in separate threads whose lifecycle is bounded by the owning state.

Typical examples of Activities are Estimator and Controller Activities, or initialization that takes an extended amount of time, as described below.

Estimator and Controller

We want to be able to get information on the status of the system at any time, while we want to be able to control the system (i.e. move actuators) only when we are sure that every component is properly initialized and available.

Therefore Estimators are implemented as Do Activities of the On state and are therefore started as soon as the process is started up, and are always running until it is switched off.

On the other hand, Controllers shall be available only when the application is in the Operational state. When the set value for an actuator has to be kept under control by providing updated target positions (like for example when tracking a telescope to follow a target in the sky) there shall be a controller thread taking care of calculating and setting the new position. In our application we have therefore a DoControl thread that implements the position controller and that is running when the application is in the Operational state. A Preset command provides a new target in sky coordinates and the Controller converts them in a loop into actuator coordinates. If no sky target is present, the Controller simply ensures that the actuator remains at a fixed location under position control.

Other control tasks might be bound to specific sub-states of Operational or might be even implemented as Actions if they have a very short execution time.

Estimation events

The state machine (SM) of an application is driven by events.

External commands issued through the interface are mapped into events.

In order to also react to changes in State Variables (SVs), i.e. to changes in physical state of the system under control, Estimators push internal events to the state machine. Guards can then be used to filter these events and trigger transitions of state, matching event data against the guard rules defined for the SM.

In the simple example of the state machine of the CCS Simulator in Figure 5, at every iteration of the pointing kernel loop the estimator publishes a NewEstimation event. If, for example, the application is in the MovingTo state, but the estimator realizes that the telescope is now standing still, the transition to the Idle state is triggered.

This allows to use effectively the state machine to react to changes in the field, without having to implement loops and if/then/else switches that check the different conditions, but just implementing much simpler individual guard conditions.

Publishing estimated State Variables
The HLCC makes available the results of the estimation tasks to clients in two ways:

- Writing the individual values in the Online Database (OLDB)
- Publishing the values, grouped in topics of logically related SVs, in the publisher/subscriber service.

The OLDB is a hierarchical database built on top of Redis[2], while the publisher/subscriber is based on the Data Distribution Services (DDS)[3].

The use of two different publishing technologies covers a range of use cases. Redis suits itself to scripts, GUIs and sequencers, requiring no type-specific knowledge to browse and read, and a slower update rate (max 5Hz), nonetheless suitable for UIs. It is likewise centralized in the Redis server removing the need for a peer discovery/match phase. DDS on the other hand is more suitable for control software to control software communication, being decentralized, strongly typed, with high performance and a rich set of quality of service parameters.

Via subscription to OLDB data points and DDS topics, an HLCC application will be notified of data changes and may react. The event could relate to parameters or measurements computed and published by other processes, or a change in the runtime configuration. An HLCC application uses subscription to learn of and adapt to external changes.

Command interface
The data publishing pattern is not intended as the command interface to the process.

A command interface to the software combines a set of standard commands applicable to all HLCC software (state query and change commands, and other software diagnostics) and application-specific commands applicable to the target application. The command interface uses a Request/Reply communication pattern implemented by CII over "ZPB", a combination of ZMQ[4] with packet serialization by ProtocolBuffers[5]. ZPB is also used by the HLCC software to send commands to other HLCC processes.

Initialization and configuration
When an HLCC Control Software process starts, it reads its configuration from the Configuration Repository. This is a set of YAML files available at a URL, typically a set of files stored locally or queried from the Engineering Archive.

These data form the bases of the Runtime Configuration, which the process mirrors to the Online Database (OLDB) for observability and for clients that are interested in knowing the configuration of the system.

It is possible to ask the system to load a new configuration by forcing a new initiallization or by using specific configuration commands, if it makes sense to change some configuration parameters dynamically.

Subsystem Functions
An LSV hosts its Subsystem Functions within the ELT CCS, presenting the Subsystem Function as a set of published State Variables, a command interface and configuration data.

Figure 6 shows UI applications interfacing with the CCS via Request/Reply over ZPB and querying of current data from the OLDB. The technique is equally applicable to HLCC software or LSV software. HLCC software communicates with Subsystem Functions (or other HLCC software) via the same patterns, in addition it may subscribe to DDS topics of other control processes.

[2] Redis, https://redis.com/
[3] eProsima FastDDS, https://www.eprosima.com/index.php/products-all/eprosima-fast-dds
[4] ZeroMQ, https://zeromq.org/
[5] Google Protocol Buffers, https://developers.google.com/protocol-buffers

The goal is to have the same technologies and patterns applicable to all levels of CCS software, with the interface to the subsystem control systems represented as a set of Subsystem Functions. Coordinated supervisory control of the telescope is provided by the HLCC by manipulating and monitoring the Subsystem Functions via their State Variables, Command Interfaces and Configuration.

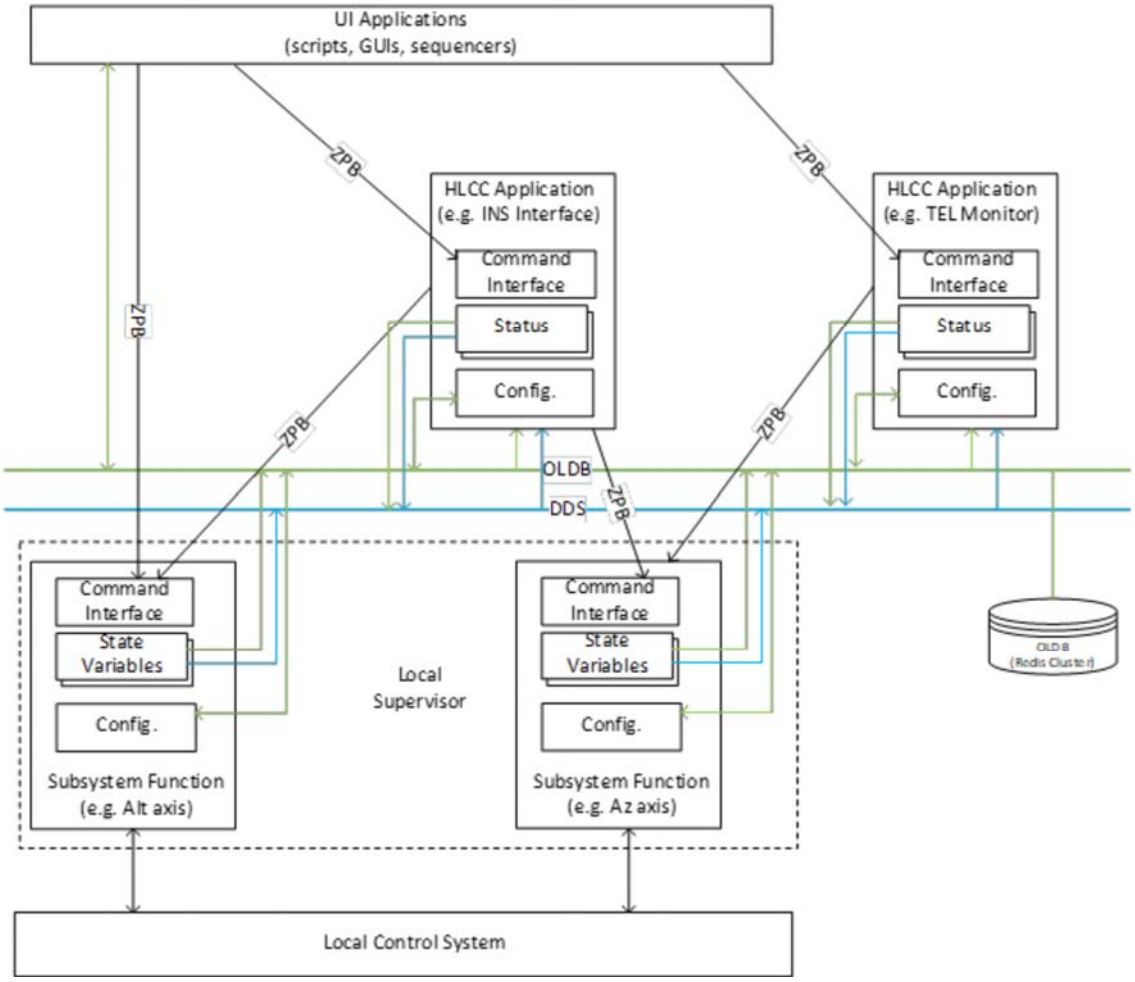

Figure 6: HLLC-LSV interfaces

Time synchronization

The ELT measures elapsed time from a monotonic clock, and manages time instances, or timestamps from a wall-clock.

The monotonic clock selected for the ELT Control System is the International Atomic Time (TAI) timescale. The wall-clock time standard is UTC, with UNIX epoch (Midnight, 1 January 1970).

The Networking Infrastructure is responsible for the communication of absolute time in a service termed the Time Reference System, which includes the Observatory Clock.

The Time Reference System network protocol is NTP (Network Time Protocol) for accuracy in the tens of milliseconds, and PTP (Precision Time Protocol) for accuracy in the sub-microsecond range.

Where commands require synchronization across distributed systems (Synchronous Commands), they are frequency locked to absolute time with zero offset. A Synchronous Command may be received at any time within the time window (from TE_i to TE_{i+1}) determined by the command rate and latency and is expected to be applied at the next time event following the reception of the command (TE_{i+1}).

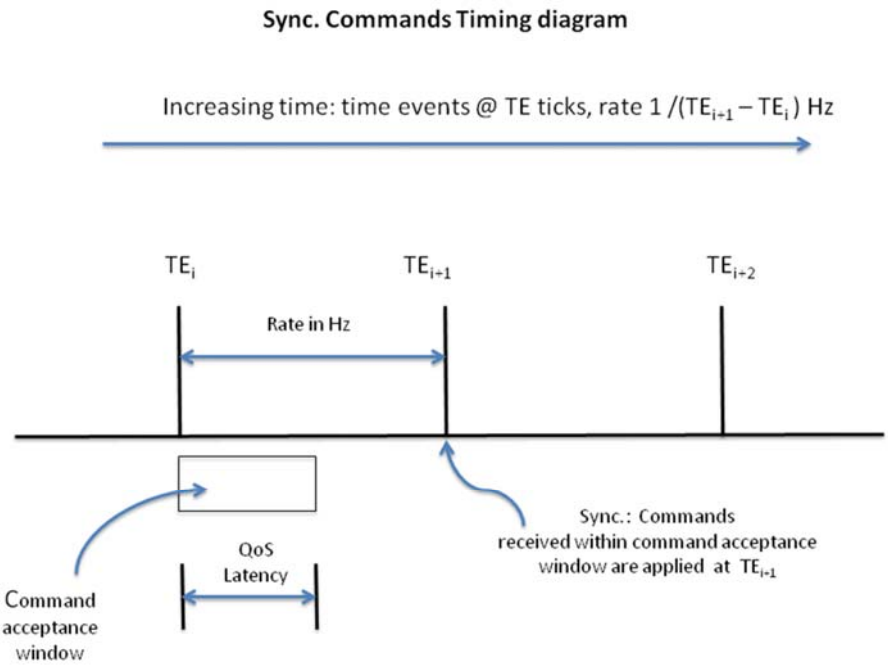

Figure 7: Synchronous Command timing diagram

For example, when tracking, the new target ALT/AZ position gets computed and commanded at a time that is aligned to the full second, and again at the next global time event 50 ms later.

For loops running at 20 Hz, our deterministic ethernet connections can be trusted to deliver the command with a latency that does not spill over to the next time event. When using commands that are agreed to apply to a future time, we avoid problems with the concept of "now", which would be plagued by varying latencies.

For tracking, we calculate (right after a time event) the expected position for the time 50 ms in the future, and send it. The MS LCS may receive it, say, 10 ms later, and knows that the commanded position applies for the upcoming time event 40 ms ahead. The same implicit timestamping applies to measurement data published at global time events. Our ICDs define for every command and measurement if it synchronizes with global time events in this way. CCS does not only calculate commanded positions in the future, but also "measurements" that are used by the instruments. For example, the image rotation ("north angle") can only be measured fully accurately once the telescope has moved to a new position. To help the instrument deal with rotations, the telescope nonetheless sends its best effort estimate for the image rotation that will be seen 50 ms later.

No backward calculation of sky positions

With the complex opto-mechanical system of the ELT, it is not clear if it would be possible to back-calculate the actual current position on the celestial sphere where the telescope is looking, based on internal telemetry, i.e. on current telescope and mirror encoder readings and pointing models.

The telescope can only really know where it is pointing by looking at the sky, i.e. by looking at the guide stars in the images acquired by the guide probes (the telescope has no access to the center field and therefore cannot see the actual

target position requested by the instruments, but can calculate with high precision its relative position with respect to the guide stars in the focal plane of the pre-focal station).

The telescope blind pointing system shall be just good enough to bring the guide stars in the acquisition field of the guide probes. At this point the system can measure the positions of the guide stars from the images and perform all optical and mechanical corrections needed to place the requested sky coordinates in the pointing origin defined by the instrument.

Therefore, only when there are known guide stars on the detectors of the guide probes the telescope knows where it is pointing and can publish the "true" measured (and not backward-calculated) position.

Otherwise no position gets published.

4.4 CCS Simulator

In the first iterations of the development of HLCC, we have concentrated on the development of the Telescope Interface of the CCS Simulator and of a simulation Pointing Kernel, in order to be able to provide our customers (mainly telescope integration team and instrumentation development teams) with a usable system.

To detect interface or implementation issues early (with frameworks and common design choices), the simulator is designed to be similar to a real application, rather than using all available shortcuts. The processes that implement the simulator are therefore all applying the design described in the previous sections.

We use independent threads for (a) control and simulation, and (b) estimation and publishing of measurements. In a real system, the estimator would likely be a separate process, which is why we keep it rather separated already now.

Both threads run a loop at 20Hz. Both loops are synchronized to global time events, but not directly between each other.

The simulator publishes both the non-deterministic and the deterministic data defined in the ICD between CCS and instruments, where in the final system the deterministic data will be published by TREx.

Working on the Telescope Interface and on the simulation Pointing Kernel has, as an additional benefit, given us a practical tool to discuss with the users the requirements for the interface, identifying and solving ambiguities at an early stage.

5. TELESCOPE REAL-TIME EXECUTOR (TREX)

Both HLCC and TREx 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.

TREx receives sensor data from the PFS guide probe and WFS cameras, as well as from, optionally, the Phasing and Diagnostic Station (PDS, an optical sensing subsystem that will be used during optical alignment and commissioning) 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.

TREx has several key modes:

- Perform field stabilization or guide probe AO (in the order of 50 modes) based on PFS GP Imager or WFS data. (the PDS SH WFS may also be used as a guiding sensor).
- Support instrument-driven AO, where tip-tilt control is shared between instrument and PFS sensor data, while the instrument controls the higher order AO modes (low temporal frequencies) (Cascade mode).
- Support instrument-driven AO, where tip-tilt and higher order modes are fully driven by the instrument AO module (Sequential mode).

In all cases TREx performs saturation management of the M4 Adaptive Mirror and M5 Tip-Tilt stage through transfer of accumulated stroke to the Main Structure and M1.

TREx is a multicore datacentre-class server running Linux. The software task affinities, interrupts and network stack and hardware ensure low-latency, deterministic reception and transmission of UDP/IP data packets over 10GbE. The UDP/IP data packets follow the ELT "MUDPI" data format which is used for all distributed control loops in the ELT (camera pixel streams, mirror and mount commands and measurements). MUDPI is a very basic protocol, as it contains no transaction

or session requirements, it is little more than a standardized wrapper for UDP payloads with some additional constraints. The simplicity of MUDPI, a key requirements, makes it suitable for use in Ethernet-based distributed control loops and high-performance interfaces of the Control System across languages and architectures [15].

TREx, where possible while meeting latency requirements, will use a configurable software framework and a suite of building blocks to abstract domain logic from underlying real-time computing infrastructure. After trials of several products, *GNU Radio*[6] has been preliminarily selected and successful deeper prototyping was carried out on this framework. GNU Radio operates on a flow graph composed of interconnected blocks with streams of items flowing along the connections. Blocks encapsulate the processing of items. They can be hierarchical, i.e. contain more interconnected blocks themselves, which enables nested flow graphs.

An example flow graph for TREx is shown below.

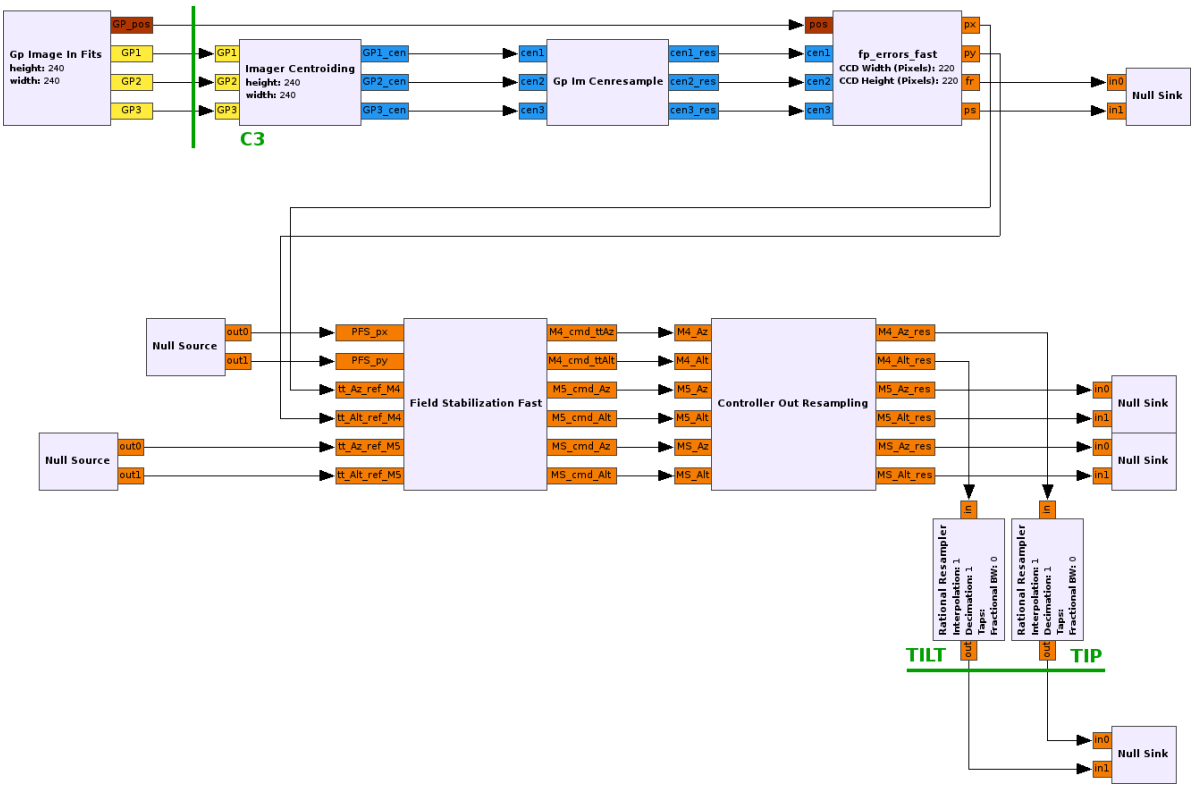

Figure 8: TREx GNU Radio flow graph

6. ELT CONTROL MODEL

Testing the high-level coordination of the system (integration of all components, coordination algorithms, operational models, performance) while the complete subsystems to be coordinated are not yet fully implemented, and before moving on the real telescope in Armazones, requires the availability of an integrated and realistic platform.

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

The ELT Control Model (ECM) contains hardware, software and networking infrastructure reflecting the designed final control system to be deployed at Armazones.

The ECM is situated at ESO headquarters in Garching, in the Computer Room (representing the Armazones Computer Room) and a neighbouring lab (representing the telescope area, or field). Single-mode, multi-mode and Cat5 cables join the two rooms reflecting the infrastructure between the computer room and field at Armazones. A time reference system (including Precision Time Protocol and Grandmaster clock) is likewise part of the ECM.

Subsystem suppliers are required to build a simulation mode in the LCS they deliver. The simulation mode allows the use of the LCS without requiring the presence of field devices and actuators. While the richness and completeness of the LCS simulations to be delivered will vary between suppliers, there are categories of simulation which are expected. These are:

- No simulation available
- Communication: the communication layer is covered, i.e. ports, tcp, multicast, content data structure, data-size and types.
- Reaction: if a command is sent, something happens, i.e. when sending the "Close Loop" command the loop closed status will change to "true".
- Behaviour: the behaviour of the system is simulated, for instance a command might be clipped, or a temperature sensor might change over time.
- Interface performance: for certain commands the interface performance is fulfilled in terms of timings (i.e. deterministic communication often requiring systems to be synchronized to the Grandmaster clock).

The simulations enable the LCS to be run in the ECM, where it may be used for testing and development, offline from the production system, and as part of full system test and integration activities. Note that no safety-critical applications (such as the safety hardware of an LCS) are to be provided in simulation, and the safety system is not a part of the ECM.

As each LCS design matures and stabilizes, a copy of the LCS hardware (needed to run the simulation) is procured and installed in the ECM. Periodic deliveries of LCS software received from the supplier are then deployed (in simulation mode) to this hardware, thereby maintaining the latest LCS software in the ECM.

As with LCSs, all Central Control System products are likewise deployed to the ECM for test and integration. This includes the integration infrastructure (communication middleware, configuration management, logging, online database), HLCC and TREx. As new versions of each of these products are released by the respective software teams, the release is installed (by the test and integration team) in the ECM and tested in conjunction with the other products. Issues identified in the integration tests are reported back to the respective software development team or subsystem supplier.

The ECM is intended to grow and be maintained to reflect the control system deployment foreseen for Armazones. In addition, the test and integration team will build a solid working knowledge of the full control system which can be leveraged during AIV.

7. DEVELOPMENT PROCESS

The HLCC development follows development standards and a high level development process common to the whole ELT project [16], tailored (as foreseen by the process itself) to the specific characteristics of HLCC and supported by tools made available by the ELT Development Environment team.

The adopted process is based on agile/iterative principles (supported by Jira[7] for task tracking and planning, with agile plugins).

Requirements and Use Cases developed with the contribution of the different stakeholders of the system (including not just scientific and operation personnel, but also development and integration and testing teams) are kept as the basis for the identification of the *features* to be developed. Features are mapped at the higher level into *epics* and at a smaller granularity into *stories*, to be implemented at each iteration.

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

Requirements and Use Cases are traceable from the top-level requirements of the associated subsystem (supported by DOORS[8] as the ELT project level requirements management tool). They are then traced to the implementation components through the system model (implemented using the MagicDraw[9] modeling tool, but only to the extent that is necessary to document the design and to trace it to requirements).

At each monthly iteration a set of *stories* to be implemented for the users is selected from the backlog, together with technical activities and findings from previous iterations (i.e. general software infrastructure, implementation of common libraries, refactoring, bugs to be fixed, performance to be improved).

Each iteration shall result in a "potentially shippable product" with new functionality of direct value for the final users. This implies that a *story* is a vertical development, from the external interface to the instruments or operators down to the interfaces to the subsystems' LSVs or TREx (or their simulation). During an iteration, continuous integration and testing take place through the integration of Jenkins[10] automation server pipelines with our GitLab repository and Jira. Jenkins is also running code quality control and coverage testing pipelines.

In this way the stakeholders see a tangible progress in the development of their product and an increase of value for them. They can test new functionality and get acquainted with the system. This fosters a productive discussion and produces very useful feedback for the development team to improve the product and adapt to the changing requirements of the final users. Indeed, requirements written up front are often generic or ambiguous and get clarified only once a first, even partial, implementation is available.

We are experiencing this very clearly now with the CCS Simulator, that provides a simulation of the ICD toward instruments: the ICD document is still very high-level and we have now the chance of clarifying and testing a lot of details and identify inconsistencies.

The "art" of a good iteration is to

- identify and define *stories* so that they can be reasonably implemented in the time span of an iteration.
- Find the right balance between stories and technical tasks, so that users perceive that each iteration has an added value, but the development team can at the same time keep under control the code base and extend it without having to resort to "spaghetti" code and dirty design solutions.

For example, an *epic* might cover the implementation of the preset to a new target on the sky. This shall be split into several *stories*, each of them adding one parameter to the interface or an extension to an underlying algorithm or a new interface toward a controlled subsystem. The full preset will be implemented in several iterations, but at each iteration the user will have something more available for testing.

The first things to be implemented are therefore the interfaces toward the user, typically dummy at the beginning. This is in contrast, for example, with a process where all the structure of an application is implemented first, but interfaces and functionality of interest for the users are provided only much later. The latter allegedly allows to develop a cleaner application, able to handle the full complexity of the features to be provided instead of requiring sometimes heavy refactoring as the complexity increases. But we think that the risk of delivering an application that does not satisfy the needs of the final users is much higher.

8. TIMELINE AND CURRENT STATUS

The status of the ELT Control System was presented at this conference in 2018 [3].

Since then,

- Common infrastructure (CII) and development environment have been consolidated and are routinely used. First real size applications based on CII have been implemented, providing valuable feedback, in particular for what concerns performance and reliability, which are being addressed.

[8] Doors, https://www.ibm.com/us-en/marketplace/rational-doors
[9] MagicDraw, https://www.nomagic.com/products/magicdraw
[10] Jenkins, https://www.jenkins.io/

- Work on most LCSs from the respective contractors is well advanced. The M1 LCS, developed in house and in a well-advanced state, is used now for integration and testing on real hardware and as a testbed for the technologies.
- Requirements for the LSVs have been collected and a common design has been drawn and tested on prototypes. Development of the first LSVs will start in the coming months.
- Architecture and high-level design for the HLCC has been defined, a prototype has been implemented to verify and compare technologies, and a first version of the CCS Simulator has been released for internal usage in the ECM. A CCS Simulator for use by instrument builders will be released alongside the next instrumentation software framework release.
- Requirements specifications for TREx are being reviewed and design and development is commencing.
- First versions of the instrumentation common software have been released to instrument consortia allowing them to get acquainted with the technologies and to start prototyping and development.

For what concerns more specifically HLCC, a first prototype was developed using Java and demonstrated very good results and sound design. A project-level decision was then made to move the HLCC programming language to C++ for greater uniformity with the LSV, RTC, and the instrumentation teams. It is hoped this change increases synergy across the teams and reduces maintenance effort in infrastructure products that currently have to support both Java and C++.

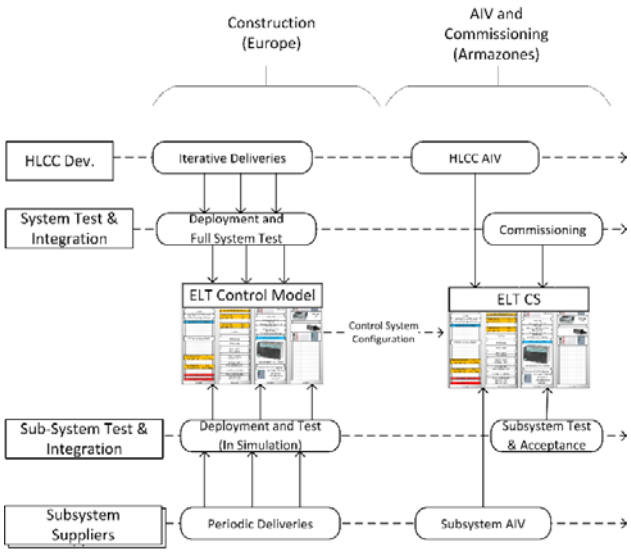

Figure 9: HLCC development timeline

The timeline of our work is sketched in Figure 9.

In the last year we have concentrated the HLCC development effort on the Telescope Simulator, on the telescope interface, and on what is needed by instrumentation and ECM.

We will continue to add features, to reach a complete implementation of the interface in the next year.

In the meantime, we will design interfaces to TREx and the LSVs, based on the analysis of operational use cases.

As TREx and LSV components will become available, we will in parallel implement the corresponding interfaces, that will be tested on the ECM.

Features will be developed and released based on the priorities defined by the AIV and commissioning teams.

The initial stages of AIV will focus on telescope subsystems, involving then LCSs and LSVs, but not the HLCC. One exception to this is the Segment Exchange Tool, which is required early as part of integrating the segment exchange manipulator and the local coherencer. This tool may be required already in 2024, prior to installation at Armazones.

HLCC will be required once the complete light path will be available and full telescope technical commissioning will begin, around 2027. At this point, the commissioning team will need the sequencing capabilities and the subsystem coordination functions to characterize the telescope and put in practice and refine the operational procedures defined until then mostly on paper.

9. CONCLUSION

While the control strategies outlined in section 1 are being simulated and tested 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

The HLCC in its initial versions must be an adaptable, flexible tool set. While the final goal of the HLCC is to present the telescope operator with a uniform and powerful interface to optimize telescope observation and maintenance, the goal during this phase (the Construction phase) is a set of tools supporting telescope AIV and capable of capturing and implementing new telescope operation and wave front control requirements as they appear in AIV. Along with changes in requirements, the HLCC must adapt to changes emerging in the as-built subsystems, delivered be external suppliers, and in evolving technologies. As always, we depend on the use of industrial standards and large, open source products to ease maintenance during software and hardware upgrades and follow a development process resilient to change.

REFERENCES

[1] Tamai, R. et al., "Status of the ESO's ELT construction", these proc. SPIE 12182, paper 12182-43 (2022)
[2] H.Bonnet *et al*., "Adaptive optics at the ESO ELT," Proc. SPIE 10703, Paper 10703-10 (2018).
[3] Chiozzi, G. et al., "The ELT control System", Proc. SPIE 10707, paper 10707-31 (2018)
[4] Chiozzi, G. et al., "The ELT Control System: Recent Developments", Proc. ICALEPCS2021, Shanghai, China (2021)
[5] R.Biasi et al., "E-ELT M4 adaptive unit final design and construction: a progress report", Proc. SPIE 9909, Paper 9909-7Y (2016)
[6] P.T.Wallace, "A rigorous algorithm for telescope pointing," Proc. SPIE 4848 (2002).
[7] B.Sedghi, M.Müller, "Dynamical aspects in control of E-ELT segmented primary mirror (M1)," Proc. SPIE 7733, Paper 7733-2E (2010)
[8] B.Sedghi et al., "Field stabilization (tip/tilt control) of E-ELT", Proc. SPIE 7733, Paper 7733-40 (2010)
[9] B.Sedghi, "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/
[10] Chiozzi, G. et al., "Designing and managing software interfaces for the ELT", These proc. SPIE 10707, paper 10707-78 (2018)
[11] ELT ICS Rapid Application Development (RAD), http://www.eso.org/~eltmgr/ICS/documents-latest/RAD/sphinx_doc/html/
[12] L.Andolfato et al.,"Behavioural models for device control", Proc. ICALEPCS2017, Barcelona, Spain (2017)
[13] J.Hofer, "Behavioural Customization of State Machine Models", Master's thesis, TU Kaiserslautern, Germany (2022)
[14] M.Ingham et al., "Engineering Complex Embedded Systems with State Analysis and the Mission Data System," Proceedings of First AIAA Intelligent Systems Technical Technical Conference (2004), https://trs.jpl.nasa.gov/handle/2014/38225
[15] ESO-302020 - MUDPI Standard, internal ESO ELT project document
[16] ESO-265043 - ELT Control System Software Development Process, internal ESO ELT Project document
[17] ESO-311982 - ICD between the instruments and the Central Control System, internal ESO ELT Project document
[18] L. Andolfato et al., "Experiences in Applying Model Driven Engineering to the Telescope and Instrument Control System Domain", Proc. MODELS2014, LNCS 8767, Springer, pp. 403-419 (2014).