DEVELOPMENT OF THE SKA CONTROL SYSTEM, PROGRESS AND CHALLENGES

S. Vrcic, T. Juerges, SKA Observatory, Macclesfield, UK

Abstract

The SKA Project is a science mega-project whose mission is to build an astronomical observatory that comprises two large radio-telescopes: the SKA LOW Telescope, located in the Inyarrimanha Ilgari Bundara, the CSIRO Murchison Radio-astronomy Observatory in Western Australia, with the observing range 50 – 350 MHz, and the SKA MID Telescope, located in the Karoo Region, South Africa, with the observing range 350 MHz – 15 GHz. The SKA Global Headquarters is in the Jodrell Bank Observatory, near Manchester, UK. When completed, the SKA Telescopes will surpass existing radio-astronomical facilities not only in scientific criteria such as sensitivity, angular resolution, and survey speed, but also when it comes to the number of receptors and the range of the observing and processing modes. The Observatory, and each of the Telescopes, will be delivered in stages, thus supporting incremental development of the collecting area, signal, and data processing capacity, and the observing and processing modes. Unlike the scientific capability, which in some cases may be delivered in the late releases, the control system is required from the very beginning to support integration and verification. Development of the control system to support the first delivery of the Telescopes (Array Assembly 0.5) is well under way. This paper describes the SKA approach to the development of the Telescope Control System, and discusses opportunities and challenges resulting from the distributed development and staged approach to the Telescope construction.

INTRODUCTION

The SKA Observatory [1] is an international organisation whose mandate is to build and operate two multi-purpose radio telescope arrays. The SKA Low Frequency Telescope array (in further text SKA LOW), located in the Murchison region, Western Australia, with the observing range 50 - 350 MHz, will consist of more than 131,072 log-periodic antennas organised as 512 stations; the maximum distance between two stations is 65 kilometres. The SKA Mid Frequency Telescope array (in further text SKA MID), located in the Karoo region, South Africa, with the observing range 350 MHz - 15 GHz, will comprise 197 offset-Gregorian dishes; the dishes are 15 metres in diameter, the maximum distance between two dishes is 150 kilometres. In both Telescope arrays, the receptors (stations in the SKA LOW and dishes in the SKA MID) are arranged in a dense core (with the diameter of ~1 km), and three spiral arms.

Work on the construction, which officially started in 2021, was preceded by the pre-construction activities, which ended with the successful completion of the Critical Design Review (CDR). The CDR was a requirement for the transition to the construction phase. As a result, when the Project entered the construction phase, a comprehensive set of documentation was available, including the requirements and architecture documents, not only for the Observatory and for both Telescopes, but also for each major sub-system, including the Telescope Control System [2, 3].

This paper provides an overview of the Telescopes and Telescope Control System, describes how the development team is organised, outlines the progress made so far, and discusses some of the challenges the development team encountered during the construction and the strategy to address those challenges.

TELESCOPE ARCHITECTURE

Figure 1 shows the major subsystems, flow of the observed data and flow of control. Figure 1 is an overview of an SKA Telescope; the receptors are different (Low Frequency Aperture Array (LFAA) and MID array of dishes) while the rest of the subsystems are very similar in the SKA MID and KA LOW.

The key Telescope subsystems that capture and process astronomical data are the receptors, Central Signal Processor and Science Data Processor. The key Telescope support systems are Synchronisation and Timing (SAT), Control System (CS), Networks and Computing Platform where software is deployed. The sub-system known as Observatory Science Operations (OSO) provides a set of tools for proposal submission and management, and for observation preparation, planning, scheduling, and execution.

Equipment and software for each SKA Telescope is deployed as follows:

- The core of the array is located at a radio-quiet site in a desert (Karoo Region, SA and Murchison region, AU)
- Receptors in the spiral arms are spread across a large area; some of the MID array dishes are located more than hundred kilometres from the core.
- The Central Processing Facility (CPF) is located near the core of the array.
- The Science Processing Centre (SPC, where Central Signal Processor and Science Data Processor are deployed, is located in a major urban centre (Cape Town and Perth).
- The Engineering Operations Centre (EOC) is located in a town relatively close to the Telescope sites (Carnavon, SA and Geraldton, AU).

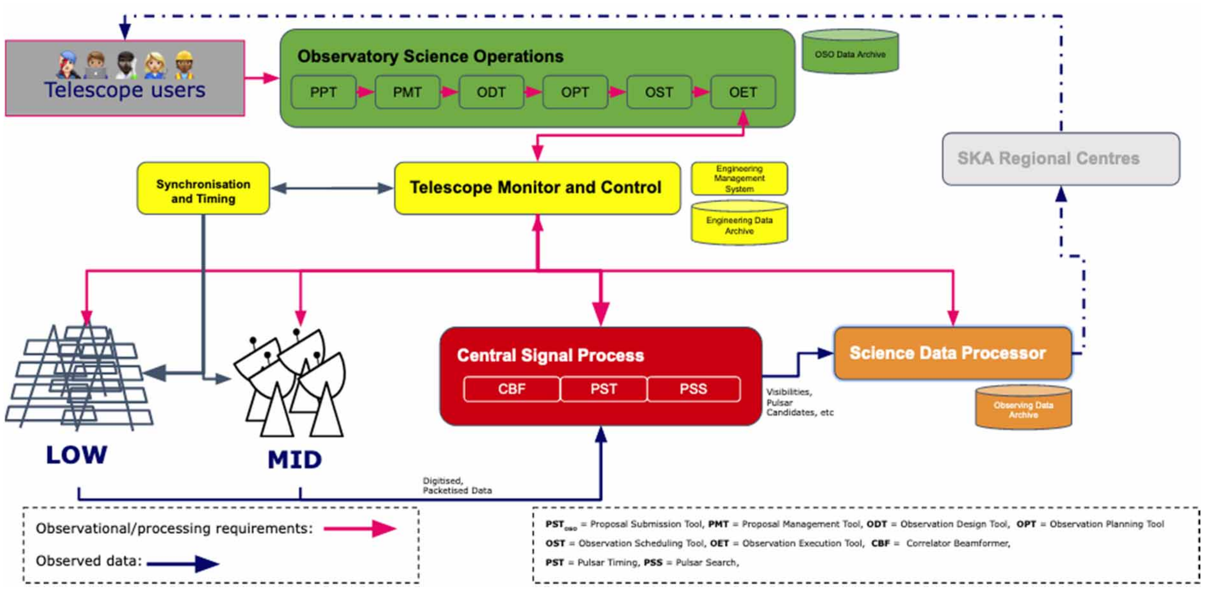
Telescope users
Observatory Science Operations
PPT
PMT
ODT
OPT
OST
OET
OSO Data Archive
SKA Regional Centres
Synchronisation and Timing
Telescope Monitor and Control
Engineering Management System
Engineering Data Archive
Central Signal Process
CBF
PST
PSS
Science Data Processor
Visibilities, Pulsar Candidates, etc
Observing Data Archive
LOW
MID
Digitised, Packetised Data
Observational/processing requirements:
Observed data:
PST_OSO = Proposal Submission Tool, PMT = Proposal Management Tool, ODT = Observation Design Tool, OPT = Observation Planning Tool
OST = Observation Scheduling Tool, OET = Observation Execution Tool, CBF = Correlator Beamformer,
PST = Pulsar Timing, PSS = Pulsar Search,

Figure 1: SKA Observatory - key subsystems.

Given the distributed deployment, some of the monitor and control software must be deployed locally (close to the equipment), to provide monitor and control capability in the absence of connectivity with the Central Processing Facility, Engineering Operations Centre, and/or Science Processing Centre.

CONTROL SYSTEM REQUIREMENTS

Physical distribution of equipment, described in the previous section, is one of the aspects to consider when designing the Control System. Other significant features of the Telescope design are implementation of various subsystems, and the number of components to be monitored and controlled. For example, the Low array will consist of 131,072 log-periodic antennas, organised as 512 stations. Signal captured by each antenna is digitised and inputs from different antennas aligned in time before they can be used as input for beamforming. Each Station is required to form up to 48 beams on the sky, each beam can be pointing to a different point in the sky. Hundreds of software, firmware and hardware components are required to perform signal processing for 512 stations. The Monitoring, Control and Calibration Subsystem (MCCS) for the Low Frequency Aperture Array (LFAA) will consist of hundreds of software components. In the SKA MID Telescope, the Correlator and Beamformer must be able to process up to 5 GHz of bandwidth from each of 197 receptors; more than 800 boards, each equipped with a System on Chip (SoC), comprising ARM processor and FPGA, are needed to perform required signal processing. These two examples illustrate the number of components to be monitored and controlled. Other Telescope subsystems are comparable in size and complexity, and in some cases even more demanding.

The Telescope Control System role and requirements may be grouped as follows:

- Requirements related to monitor and control of equipment and software.
- Requirements related to execution of observations.
- Non-functional requirements.

Summary of the Control System requirements related to equipment and software:

- Monitoring - periodically check the status of equipment and software, report status on request and change, report faults and errors, implement intelligent status aggregation, i.e. ability to interpret impact of faults and errors on the Telescope availability.
- Archiving - maintain historic record of the Telescope status. Enable Operations to select the attributes to be archived periodically and/or on change. Provide API and tools for access to the historic record.
- Provide Application Programming Interface (API) that can be used to trigger state/mode changes and configuration changes. Expose as attributes all information required to understand the state of the system, including information required for debugging and troubleshooting.
- Support for hardware / software / firmware updates.
- Support for debugging, testing and maintenance.
- Provide HMI for monitoring and control as described above.

The requirements listed above apply at all the levels in the system hierarchy: Telescope, subsystems, and individual components.

Summary of the Control System requirements related to observing:

- Each telescope supports multiple observing modes - provide an API that can be used to configure a telescope for the desired observing mode.
- Point the receptors and beams; track sources (as per observation configuration).
- Provide regular updates for pointing and delay tracking.

- Provide an API that can be used to start and stop observations (co-ordinated start/stop for pointing, capture and processing of the astronomical data, and generation of output products).
- Provide an API that can be used to subdivide the array (both receivers and processing resources), and to operate each sub-array independently, in terms of pointing, observing band & mode, and start/stop of observations. (Subarraying is an architecturally significant requirement which increases Control System complexity.)
- Provide API and HMI for monitoring observing related status per subarray (including the equipment used by the subarray).
- Provide a forensic tool for understanding of the state and behaviour of the system.
- Provide access to calibration correction parameters and algorithms.

The key non-functional requirements for the Control System are:

- Reliability - Ability to perform a required function under stated conditions for a stated period of time. Measured by: Mean time between failures. Mean time to repair.
- Resilience - Ability to function in the presence of errors and failures. Ability to recover from errors and failures. Containment (localization) of errors and failures. If some of the components fail, the rest of the telescope should continue to function.

Control System reliability and resilience have a direct impact on the Telescope availability; when the Control System is unreliable or not working the Telescope is not available.

Other significant non-functional CS requirements are performance, testability, usability, scalability, extendibility, modifiability, and upgradability.

CONTROL SYSTEM ARCHITECTURE

Telescope architecture was defined early in the design process, some aspects of the Control System architecture were a logical consequence of the Product Breakdown Structure (PBS) and allocation of functionality. General approach to Control System architecture is a layered architecture, where each subsystem, product and component is responsible to provide an interface that can be used to configure and control the operating mode, as applicable for each entity, and to evaluate and understand the status and behaviour. Layering is required to handle the system that consists of many diverse components. Another feature of the CS design is to push the monitor and control related functionality and intelligence to the subsystems and components; each sub-system and component then translates higher-level parameters in the detailed configuration.

Other choices made early in the development include:

- Fundamental SKA Software Standards [4] define the preference for open-source software and Python as the programming language of choice.
- Choice of TANGO Controls framework [5] as the base for the Control System implementation [2, 3].
- Control System Guidelines [2] define:
  - Hierarchical approach to Control System, identify the key components and outline flow of information.
  - A standard set of states and mode indicators, the so-called SKA Control Model.
  - A standard approach for definition of the product/component API.
  - Roles and responsibilities for individual components.

The Telescope Monitor and Control (TMC) subsystem which provides the overall Telescope control and monitoring functions, will be deployed in the Central Processing Facility (CPF) near the array core (although some of the components may be deployed in the Science Processing Centre (SPC), details are still to be defined). The Local Monitor and Control (LMC) for each sub-system will be deployed close to the subsystem it controls; the same applies for the smaller subsystems and individual components. The Dish LMC is deployed in the pedestal of each MID Dish. The Monitor, Control and Calibration System (MCCS) for the Low Frequency Aperture Array (LFAA) is deployed in the Central Processing Facility, and the remote stations.

Other technological choices made so far:

- Software deployment - The Control System components will be deployed on the Common Cluster of Servers. Computing infrastructure is managed using COTS tools, not part of the Telescope Control System.
- Technologies for the Common Software Service, including virtualisation, have been selected.
- TANGO Controls framework provides built-in support for logging; the SKA approach is to use TANGO API for setting logging levels, while the infrastructure for collection of logs, logging repository and access to the logging repository is provided by the Common Software Services.
- TimescaleDB has been selected and deployed as the backend for the archiving.
- Taranta, a browser-based tool, which is part of the TANGO Controls ecosystem, has been selected as a technology of choice for the CS provided User Interfaces (UIs).

Choice of the Alarm Handling tools is still outstanding, the Elettra Alarm Handler and IC@MS by S2Innovations are being considered.

DESIGN AND IMPLEMENTATION

At the beginning of construction, a set of the SKA Tango Base Classes were developed that implements the common API and state machines defined in the CS Control Guidelines [3]. Most sub-systems use the SKA Tango Base Classes as the base for development, some have chosen to develop their own implementation that provides the same functionality and API. Main reasons for the development of the own implementation are:

- Development of most systems started in parallel, as a result, the quality of the initial versions of the base classes was not satisfactory. The teams that were not willing to wait for implementation to stabilise developed their own implementation.
- The concept of the base classes is somewhat controversial, some of the team members believe that when a large number of components base their implementation of the same set of classes, the system is hard to evolve. Evolution of interfaces is always hard and has to be synchronised; if a server changes API, the client must be modified to use the new API. In practice, our system has shown that it is not necessary to upgrade all components at the same time. Quite to the contrary, it is hard to motivate teams to keep pace with the improvements and upgrades. At any time, the integrated Control System comprises components that use different versions of the base classes. However, this is not to say that our approach is perfect.

During the development, the control model was refined, definition of components and commands improved. The commands used to allocate resources and configure subarray observing mode need to provide a rather large set of parameters; in Tango Controls framework a command can pass a single parameter. In SKA, the Tango API is used, not only to control individual devices, but also to control the Telescope and large sub-systems; to avoid the hard limit on the number of parameters, some of the commands pass a JSON script as a parameter. This allows for automated syntactic checks (against the JSON schema) and supports weak typing; both of these techniques are used to implement loose coupling of components [4].

As already mentioned, the SKA CS implements layering to handle the complex system that consists of many diverse components. Passing commands & requests through many layers may result in a brittle system, if not carefully implemented, a single misbehaving component may disrupt operations. Each component must be implemented defensively (containment, ability to operate in presence of errors). Implementation of asynchronous, non-blocking commands is described in “Asynchronous Execution of the TANGO Commands in the SKA Control System: An Alternative to the TANGO Asynch Device” [6].

So far, at least a rudimentary implementation of the TMC and virtually all LMCs have been developed. This includes the commands that control operational state and commands used to configure subarrays. All components implement a standard set of state and mode indicators. Integration of the CS components both for the MID and LOW Telescopes is in progress, and that’s where the team encounters challenges (see the section on challenges for further discussion).

The logging repository and related infrastructure, as well as the Engineering Data Archive (EDA), which provides historical record (archiving), have been deployed in the System Integration Facilities (ITFs) and are available for use by the integration teams.

Integration of the Control System components with the products delivered by the contractors has started, and this has accelerated pace of development, as the feedback from integration contributes to the backlogs of the development teams.

The development is currently transition from the phase where the development was mostly driven by the software team priorities, to the phase where priorities for the software development are mostly driven by the availability of the equipment and the system integration.

This is an exciting time for the SKA Observatory and the Control System team.

STAGED DELIVERY

The SKA Telescopes will be developed, deployed, and integrated in several stages. The Control System must provide enough functionality to support integration and verification in each stage of development.

The first milestone, known as Array Assembly 0.5 (AA0.5), will be used to prove that the system can function as an interferometer; the functionality provided by AA0.5 will not be sufficient to support scientific observations. The AA0.5 version of the MID Telescope consists of 4 dishes, the Correlator and Beamformer, the PST Engine able to receive and record a single PST beam, and the SDP able to receive and store visibilities for an array of 4 dishes, perform calibration, and other required processing. The AA0.5 version of the LOW Telescope will consist of 6 stations; other subsystems are required to provide the same functionality as for the MID Telescope.

The Control System must provide the APIs and HMIs to enable integration commissioning teams to understand and evaluate the state of the system, visibility for the parameters used in calibration, and ability to configure a subarray for the supported processing modes, start/stop observation, configure archiving and access the archived data and more. In short, virtually all CS functionality has to be provided but for a small system. So far only a subset of the functionality required for AA0.5 has been implemented, work on status aggregation and alarm reporting is still outstanding.

When it comes to non-functional requirements, in the early stages, resilience is extremely important; it is to expect that other components and sub-system will not be stable, the CS will have to detect and report issues, function in the presence of failures, and return to normal operations when the failed components recover. In other words, the Control System must be able to contain failures and compensate for deficiencies in other components.

Testability and modifiability will be verified over time, as the system expands and evolves. Some aspects of the Control System performance will be tested in early Array Assemblies; but most will be verified only when almost all equipment becomes available; the number of components to monitor and control, to a great extent, impacts the performance. In the same manner, scalability will be verified late in the project construction.

CHALLENGES

Time Zones

The software teams working on implementation of the Control System software are distributed over many time zones, from eastern Australia to western Canada, and especially for the teams in Australia and Canada, participation in the meetings and events is a challenge. The teams are distributed, and the number of developers is growing, it is expensive and difficult to organise face-to-face meetings. The SKA team has been adapting to this situation from the beginning, and, so far, team cohesion remains strong.

Evolution of Interfaces

All subsystems are being developed in parallel, for the Control System that means that detailed requirements are not available, or, if defined in pre-construction, are not applicable anymore. As the Telescope subsystems and components are designed and developed, the APIs get refined and modified. This is to expect in an Agile development environment but can cause delays and rework.

Rapid Growth

The guidelines for implementation of the Control System were developed early in the development, and have been widely adopted, but, as the pace of the development accelerates and the number of teams and developers increases, the tendency to diverge from the project standards seems to gain momentum. This may seem counterintuitive, as refusal to adopt project standards usually increases the workload, but under the pressure to deliver the teams sometimes opt for short term gains (or perceived gains).

Entropy

Given the size of the project and diversity of components and subsystems, maintaining the documentation up to date is a challenge. Documents developed in pre-construction define requirements, architecture, and interfaces for all sub-systems, they are huge in volume and scope, keeping them up to date is a challenge. In construction, the software documentation is available in the Solution Intent [7] space in the SKA Confluence [8] and the documentation generated from the code is available on Read the Docs [9, 10]. Confluence is suitable tool for rapidly changing and evolving projects, but maintaining documentation requires time and dedication, and it seems that new material is being added at ever increasing pace. Keeping it consistent is a challenge.

Lack of Experience with the Real System

Testing and integration are performed in a virtualized environment, it is easy to loose perspective of what is important, in particular when it comes to response time and resilience.

Continuous System Integration

The biggest challenge for our team is system integration. The software development team has set a goal to deliver an integrated system at the end of each Program Increment, this requires elaborate planning and synchronisation of development and integration activities, which is not easy, as different components do not always have the same priorities. For example, for the signal processing subsystems, implementation of the signal chain is the highest priority, the vertical integration with the upper layers of the control system is a relatively low risk functionality that would rather be performed after the key functionality is developed and tested. For some components, development, testing and integration of all interfaces in parallel is a challenge, in part due to limited resources.

We are getting better in synchronising the development goals and activities, but for the software team, continuous integration of the full system remains a challenge. But we are learning how to efficiently make coordinated upgrades. The team is currently re-organising the approach to integration testing.

Integration testing would be easier if the system is integrated from the bottom up, i.e. if the devices lowest in the monitor and control hierarchy are developed and integrated first. However, that's not possible, to shorten the development time and reduce the risk all the layers of monitor and control are developed in parallel.

Traditionally, similar systems were integrated from the bottom up, the components and subsystems at the bottom of the monitor and control hierarchy were tested first, then gradually integrated in more and more complex sub-systems, each tested in isolation, to verify the functionality and adherence to agreed interface specifications, and finally the whole system was integrated and tested.

The SKA does not intend to abandon the goal related to continuous integration, the plan is to provide better support for integration of two or more subsystems, so that issues can be detected early and resolved in time to support system integration.

CONCLUSIONS

Summary of what we learned over the last two years:

- Define and document standards and guidelines as early as possible.
- Document rationale for the requirements, design choices and technology choices. Technical decisions can be questioned and re-considered at any point during the project; a written record of rationale behind technical decisions helps to avoid repeating the same discussions over and over.
- It is good to be flexible, but changes are costly. Carefully consider the benefits and the cost of change.
- Delivery of the Control System is often driven by availability of other components. The Control System team has to adapt to support delivery of other components.

REFERENCES

[1] SKA Observatory, `https://www.skao.int/`

[2] L. Pivetta, A. DeMarco, S. Riggi, L. Van den Heever, and S. Vrcic, “The SKA Telescope Control System Guidelines

and Architecture", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 34-38. doi:10.18429/JACoW-ICALEPCS2017-MOBPL03

[3] S. Vrcic, "Design Patterns for the SKA Control System", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 343-347. doi:10.18429/JACoW-ICALEPCS2021-TUBR02

[4] M. Bartolini, N. Rees, "Fundamental SKA Software Standards" SKA, SKA-TEL-SKO-0000661.

[5] TANGO-Controls, https://www.tango-controls.org/

[6] B. A. Ojurt, D. Devereux, A. J. Venter, S. N. Twum, S. Vrcic, "Asynchronous Execution of the TANGO Commands in the SKA Control System: An Alternative to the TANGO Asynch Device", presented at ICALEPCS'23, Cape Town, South Africa, Oct. 2023, paper TH1BCO04, this conference.

[7] P Wortmann, L Christelis, U Badenhorst, B Mort, P Swart, F Graser, G le Roux, V Allan, "Solution Intent Definition Document", SKA, SKA-TEL-SKO-0001065.

[8] Atlassian tools: Confluence and JIRA, https://www.atlassian.com/

[9] SKA Developer Portal, https://developer.skao.int/

[10] Read the Docs, https://about.readthedocs.com