ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEMG002

SKA CONTROL SYSTEM IN 2025

S. Vrcic, T. Juerges, SKA Observatory, Jodrell Bank, Macclesfield, UK
J. Engelbrecht, Vivo Technical, Cape Town, South Africa

Abstract

It is 2025 and the SKA Telescope Control System has come a long way since the start of construction. The outline of the software architecture and some key technology decisions (including the choice of Tango) were made early. To keep the geographically distributed teams engaged, and avoid creating silos and fragmentation, development of virtually all the software components started in parallel; often while the detailed designs for the custom hardware was still evolving and before the COTS equipment was selected. The deployment strategy was adjusted to align with the industry trends. From designing a software system for hardware that does not exist we arrived at the point where we can prove that the software can work with the hardware. However, the software design and implementation meeting reality uncovered some issues, forcing us to make changes (ska-tango-base) and learn hard lessons (naive implementation of event callbacks). Are we ready to deliver a large, distributed control system? We realize that scalability will be a challenge. This paper provides an honest overview of what works and what did not work so well, and how we address issues.

SKA TELESCOPES - AN OVERVIEW

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 LOW Telescope), 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 MID Telescope), 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 LOW and dishes in the MID) are arranged in a dense core (with the diameter of ~1 km), and three spiral arms.

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), is used to prove that the system can function as an interferometer; the functionality provided by AA0.5 is not sufficient to support scientific observations. The AA0.5 version of the LOW Telescope consists of 6 stations, the Correlator and Beamformer (CBF), and the Science Data Processor (SDP) able to receive and store visibilities for an array of 6 stations, perform calibration, and other required processing. The AA0.5 version of the MID Telescope consists of 4 dishes; other subsystems are required to provide the same functionality as for the LOW 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 must be able to configure a subarray for the supported processing modes, start/stop observation, configure archiving, provide HMIs for access the archived data, and more. In short, virtually all key monitor and control functionality has to be provided but for a small system.

WHAT HAS BEEN ACHIEVED SO FAR

Amazingly, it works! The first minimalistic version of the SKA LOW Telescope passed Assembly, Integration and Verification (AIV) tests, and was handed to the Science Commissioning team for further testing with the goal to verify that the telescope design is fit for purpose (i.e. can work as an interferometer and matches required performance). Control and Monitor software is one of the components that enabled the success of this first phase. This success is the result of years of work on design, implementation and integration.

The schedule for the MID Telescope integration is somewhat behind the LOW, work on the Dish Structure controller and integration with other components installed in the dish indexer and pedestal is in progress, we expect to have a fully integrated dish by the end of 2025. This should be closely followed by the delivery of three more dishes and other telescope subsystems. Work on the assembly, integration and verification of the first release of the MID Telescope will continue in 2026.

Control and monitoring software for both MID and LOW Telescope provides functionality required to perform tasks required for verification of these scaled down versions of the LOW and MID Telescopes. We implemented a set of Tango Devices that provide API to enable Observation Execution Tool (or a script) to execute imaging scans, capture acquired astronomical data, and archive monitoring data. A set of basic User Interfaces (Taranta [2] dashboards) provide overview of the Telescope status and allow users to issue commands to configure and execute scans. Commands to abort execution of commands and restart some of the sub-systems and devices are also provided. However, most of the time testing is controlled via scripts.

In the meantime, development continues to add more functionality, improve stability and performance and provide user friendly interfaces.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEMG002

WHAT WORKED WELL

Scaled Agile Framework

Sclaed Agile Framework (SAFe) [3] provides a solid organisational structure for a word wide collaboration. Software teams are organised in 5 Agile Release Trains; each train comprises people located in at least two continents. World-wide collaboration can be productive when well organised. We all learned to consider and adjust for the time zone differences and different holiday seasons.

Tango Controls

Despite many challenges we had to overcome, the Tango Controls [4-6] framework is working well. Tango provides an infrastructure that allows any Tango Device to communicate with any other Tango Device within the system; to reduce complexity, we defined a hierarchical system [7, 8], and we find that our approach is well supported by Tango. Each subsystem implements a Tango Device that represents the subsystem and provides a single point of access for monitoring and control.

Advantage of this approach is simplified communication flow. Details of implementation are internal to each subsystem, it is key that the monitor and control interface is well defined and agreed on.

How it works:

- Works well for configuration of complex systems (allows separations of concerns, provides hi-level view).
- Does not work well for the cases where high-volume of monitoring and/or control data has to be exchanged between the devices in different sub-systems. Had to make exceptions and introduce other communication methods (Kafka [9]). We may need to make more adjustments as the system scales to its full size. This was expected because Tango Controls, by design, is a low frequency monitor and control framework (~10s of Hz) that does not support frequent high volume data transfers.

Base Classes, State Machines

Early in the project we defined a set of state and mode indicators and related state machines and started development of the base classes that implement a common set of states and modes [7, 8]. This has proven very useful as it encourages harmonisation of implementation of monitor and control software.

WHAT TOOK TIME

Getting the Foundation in Place

Although initiated at the beginning of construction, the base classes (ska-tango-base) are still evolving [8, 10-11]. It took us many iterations to get the foundation into a state that the users are happy with. We believe that we are now close to finally having a set of base classes that are fit for purpose and software we are proud of (proper practices).

We learned that an unregulated approach to the evolution of software infrastructure does not work (too many cooks). Everyone with a 'good idea' made changes that did not necessarily integrate well with the intended and existing design.

Getting the foundation right includes refactoring ska-tango-base without breaking existing software. We learned how to properly use deprecation of features, which is key for software stability, but requires additional effort and skill. In the long run the investment pays off, in a project of this size, it is inefficient and impossible to keep all the sub-systems in lockstep.

Getting the CI/CD Integration Environment Right

A lot of time and effort was invested as we experimented with various approaches to integration testing in a virtual environment. We are constantly evolving and improving our ability to integrate as many products as possible before deploying to test facilities.

Delays in Delivery of Components

Delays in delivery of some components forced us to change the plans, as the lack of components required for testing and verification forced us to invest more time in the development of simulators, thus increasing workload for the software development teams. However, some issues can be uncovered only when real components are available.

WHAT WE NEED TO IMPROVE

Handling Failures

We need to improve how we handle failures and improve ability to function in the presence of failures. If one component fails, the rest of the system should continue to function. The telescope architecture allows observations to be executed even when most of the receptors and signal processing resources are not available. It is up to the user to define the minimum requirements for each observation. The Telescope Control System should enable observations to continue as long as the user specified minimum requirements are met. For a long time, software verification and integration was conducted in a virtual environment, meaning that software developers were not exposed to the real world issues and scenarios. Initiatives are on the way to improve ability to detect and report failures, to continue to function in the presence of failures, and to be able to recover from failures.

Status Reporting

Status reporting should be improved to provide the operators with timely information regarding availability of the resources (receptors and signal processing resources) and performance. This requires software to assess and report impact of failures on the ongoing observations, and availability of unused resources.

Development Practices and Process

Need to streamline development practices and processes. There is a sense that some of the development practices and processes are overengineered and are preventing developers to quickly iterate the development cycle of

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEMG002

coding, committing, testing. For example, in the initial CI setup, three linters were used to check Python code, resulting in a situation where one of the linters suggests a modification which gets reported as an issue by one of the other two linters. This led to frustration among developers.

Recently, some of the teams adopted a single, more modern linter (ruff [12]) that provides most features in a single tool, this should be universally adopted.

Developers sometimes wait up to 20 minutes for the linter to complete during a pipeline run in CI. Suggestion is to perform linting as a single final step before merging, this would save time.

Logging

Logging is not uniformly implemented, several ways are used to produce logs: printing to stdout, logging with Python loggers, logging with the Tango logger, logging with the SKA Tango logger (implemented in ska-tango-base). In some cases, using an alternative approach is justified (e.g. an API of the SKA Tango logging makes it unsuitable to be used in certain situations), but in others the choice is based on the preferences of developers, or due to missing functionality in the SKA Tango logging system. As a result of the recent refactoring of the SKA Tango base classes it is now much simpler for developers to use the SKA Tango logger. We hope that this will provide sufficient incentive to harmonise implementation of logging.

However, not all is bad when it comes to logging, virtually all software products adhere to the standard log format, defined on the SKA Developers Portal [13].

Integration Testing

Testing of the integrated system as early as possible helps uncover issues not detected when testing in a virtual environment using simulators; this is where issues related to timing and synchronisation get detected, as well as issues related to inadequate handling of errors and failures. And it is during the integration that we sometimes find out that not all development teams fully commit to the application of the common practices and rules. It is hard to find the best approach to integration, products should be integrated as early as possible to detect inconsistencies, misunderstandings and errors, however, using early releases often requires additional time and effort from the integration team, as they end up investigating issues that should be detected during the product verification. It is sometimes hard to identify the ideal time to deploy in the integration and test facility. To increase the quality of the software products that we deliver to integration and test facilities, we need more advanced simulators to help uncover issues while testing in a virtual environment.

Database Management

We need to develop a detailed plan for deployment and management of data bases, including:

- Tango Database (TangoDB) – The TangoDB is the core of every Tango Controls system [14]
- Engineering Data Archive (EDA) [15]
- Alarm Management System (AMS) [16]

WHAT TANGO IS MISSING

Web-based User Interfaces

The Tango ecosystem does not provide professional web-based graphical user interfaces. Most of the Tango GUIs are desktop applications because most facilities in the Tango community did not make the switch to web-based utilities yet. The SKAO is at the forefront when it comes to the need for all the user facing tooling to be web-based. Among others, the SKA needs web-based tools that provide an overview of the telescope status and allow for some configuration and control. We also need tools to provide a more detailed view and some control for each of the sub-systems, and, in some cases, for individual devices. SKAO also needs:

- Alarm displays and UIs for alarm management including managing alarm configuration.
- UIs for displaying the content of the Engineering Data Archive (time series engineering data) and for managing configuration of archiving.
- UIs for displaying and filtering logs.

Key requirements for User Interfaces can be summarized as follows:

- Web-based.
- User friendly.
- Able to handle a large number of attributes, monitor a large number of devices, across several Tango facilities.
- Performance to match need for monitoring in real-time, allowing telescope operators and maintainers to closely monitor overall status of the telescope and, when needed, status of individual devices.

SKA invested in the development of Taranta [2], but although greatly improved, Taranta still does not meet all the requirements.

Documentation and Examples

Tango Controls documentation [17] has been improved a lot but still lacks good examples of Tango devices that demonstrate certain aspects of the capabilities, for example: events handling and subscription. As a consequence, in some cases in the SKA software, bad implementation of callbacks impacted the ability to handle events and the whole event handling machinery came to a perceived standstill, leading to raising concerns regarding Tango ability to handle events. As soon as the callback implementation was corrected, the concerns about event system performance and reliability proved unfounded.

WHAT WE DON'T LIKE TO TALK ABOUT

Impact of Containerisation

The way we use Kubernetes and containerisation makes debugging very hard. To avoid potential security issues and unintended breaking of the containerised system, no debugging tools are installed in any container. A sidecar

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEMG002

approach is now considered and would benefit developers a lot when they need to debug issues in running containers.

Kubernetes + firewalls + VPNs are not made for Tango: One has to jump through many hoops to make real hardware work with a Tango system that runs in one of the clusters. Single Pixel Feed Controller (SPFC) & Single Pixel Feed Receiver (SPFRx) are good examples; both are hardware components with Tango software that cannot run in Kubernetes due to the hardware's interface for software.

Kubernetes is impacting how we deploy our system. Due to the limited number of pods per node (Kubernetes allows up 250 pods per node), we are forced to run more than one Tango device per Tango device server. This leads to scalability issues that are imposed on the Tango device servers: more devices per device server result in more Tango events per device server, this in turn puts the single threaded event handling in the device servers under more pressure. This can lead to severe delays in event delivery to the event callbacks which in turn has a direct impact on the operation and performance of the respective Tango devices.

More than once, we encountered intermittent issues with event handling which are hard to diagnose, a problem exacerbated by the way we deploy software (containerisation), plus restricted access to the test facilities: firewalls, VPNs, etc.

Impact of Authentication, Authorization and Auditing (AAA)

AAA implementation was harder than expected. The choice of authentication services (Entra ID) imposed unforeseen complexity on the implementation. In one case the lack of being able to register "*.service.somedomain" to support multiple subdomains with yet undefined names required a rewrite of parts of an existing software package (Engineering Data Base – EDA [14]). It is understandable why an authentication service would not allow this, but such restriction was not anticipated by the designers of the software package.

Also, the registration process is not yet scriptable. As a result, each service has to be manually registered, and the number of services / devices even in the current, minimalistic version of the SKA Telescopes is significant.

CHALLENGES AHEAD

Scaling to the planned SKA size. One of the main requirements for the SKA Control System is scalability, we planned and designed for scalability, and we implemented the software to scale up. But in order to really know "how much of what" we need facts and numbers. Therefore, an agile approach is taken to deploy software subsystems and components first, then identify the bottlenecks and scale up the number of software components to alleviate the bottleneck. We already encountered issues with the number of attributes and events and solved those issues. We will continue to solve the issues as they get identified.

In addition to the performance issues related to handling events and commands in real-time, scaling to the planned SKA size means adding thousands of devices to the Tango DB and managing the Tango DB effectively, managing the network configuration, and other infrastructure. We still have to develop procedures and tools to efficiently handle configuration and deployment.

WHAT WE WOULD DO DIFFERENTLY NEXT TIME

Avoid focusing on the delivery of 'user value' too early in the development, instead, at the beginning of the construction, focus on the delivery of good design and good foundation.

Invest more resources in the development of the base software at the beginning of the project.

Test how the system scales early in the development process. Deploy a production size system using wireframe components. This can be used to test both the design concept and deployment strategy. Wireframe components do not need to implement functionality; it is sufficient to implement few basic functions (e.g. initialize, start/stop), deploy and instantiate all software components, preferably scaled to the full size of the system, and execute a select set of test cases. This can help identify weaknesses and flaws in the design and in the deployment strategy, before fixing them becomes costly.

Define User Interface (UI) concepts and select UI technologies early in the project; develop UI in parallel with the Control System. Development of User Interfaces focuses developers on the user needs and allows users (testers, operators, maintainers) to provide input early in the development process.

REFERENCES

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

[2] Taranta, https://tango-controls.readthedocs.io/projects/Taranta/en/latest/.

[3] Scaled Agile Framework – SAFe, https://framework.scaledagile.com/.

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

[5] Chaize J.-M., G?Âtz A., Klotz W.-D., Meyer J., Perez M., Taurel E., "TANGO - An Object Oriented Control System Based on CORBA", in Proc. ICALEPCS'99, Trieste, Italy, Oct. 1999, paper WA2I01, pp. 475-479.

[6] Tango Controls project on Gitlab, https://gitlab.com/tango-controls/.

[7] 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

[8] 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

[9] Apache Kafka, https://kafka.apache.org/.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEMG002

[10] B.A. Ojur, D. Devereux, S.N. Twum, A.J. Venter, and S. Vrcic, "Asynchronous Execution of Tango Commands in the SKA Telescope Control System: An Alternative to the Tango Async Device", in *Proc. 19th Int. Conf. Accel. Large Exp. Phys. Control Syst. (ICALEPCS'23)*, Cape Town, South Africa, Oct. 2023, pp.~1108-1114. doi:10.18429/JACoW-ICALEPCS2023-TH1BCO04

[11] SKA Tango Base Classes, https://ska-telescope.gitlab.io/ska-tango-base/.

[12] Ruff linter, https://docs.astral.sh/ruff/linter/.

[13] SKA Developers Portal, https://developer.skao.int/.

[14] The Tango Database, https://gitlab.com/tango-controls/TangoDatabase

[15] T. Juerges and A. Dange, "The SKAO Engineering Data Archive: From Basic Design to Prototype Deployments in Kubernetes", in *Proc. 19th Int. Conf. Accel. Large Exp. Phys. Control Syst. (ICALEPCS'23)*, Cape Town, South Africa, Oct. 2023, pp.~1590-1593. doi:10.18429/JACoW-ICALEPCS2023-THSDSC05

[16] T. Juerges, S. Vrcic, and A. Dange, "Design, implementation, and deployment of SKAO's AlarmHandler, the software system to automatically handle Tango controls attribute alarms in the SKA control system of SKA Low and SKA Mid," in *Software and Cyberinfrastructure for Astronomy VIII*, G. Chiozzi and J. Ibsen, Eds., Sep. 2024, p. 45. doi:10.1117/12.3018419

[17] Tango Controls documentation, https://tango-controls.readthedocs.io/en/latest/