ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH1BCO04

ASYNCHRONOUS EXECUTION OF TANGO COMMANDS IN THE SKA TELESCOPE CONTROL SYSTEM: AN ALTERNATIVE TO THE TANGO ASYNC DEVICE

B.A. Ojur*, [1], D. Devereux[2, 3], A. J. Venter[1], S. N. Twum[3], S. Vrcic[3]
[1]SARAO, Cape Town, South Africa
[2]CSIRO, Clayton, UK
[3]SKAO, Macclesfield, UK

Abstract

Equipment controlled by the Square Kilometre Array (SKA) Control System will have a TANGO interface for control and monitoring. Commands on TANGO device servers have a 3000 milliseconds window to complete their execution and return to the client. This timeout places a limitation on some commands used on SKA TANGO devices which take longer than the 3000 milliseconds window to complete; the threshold is more stricter in the SKA Control System (CS) Guidelines. Such a command, identified as a Long Running Command (LRC), needs to be executed asynchronously to circumvent the timeout. TANGO has support for an asynchronous device which allows commands to be executed slower than 3000 milliseconds by using a coroutine to put the task on an event loop. During the exploration of this, a decision was made to implement a custom approach in our base repository which all devices depend on. In this approach, every command annotated as "long running" is handed over to a thread to complete the task and its progress is tracked through attributes. These attributes report the queued commands along with their progress, status and results. The client is provided with a unique identifier which can be used to track the execution of the LRC and take further action based on the outcome of that command. LRCs can be aborted safely using a custom TANGO command. We present the reference design and implementation of the Long Running Commands for the SKA Controls System.

INTRODUCTION

A long running action, within the SKA Control System (CS) Guidelines [1], is attributed to a command that exceeds the execution time threshold of 10 milliseconds [2]. The Telescope Control System is composed of a number of subsystems which form an intricate network of communicating components. Due to the hierarchical interaction of these components, coupled with some network and I/O bound actions, response delays are symptomatic within this distributed system. TANGO [3] device servers used to control and monitor the equipment of these components timeout on commands which run tasks longer than 3000 milliseconds. Within the SKA network, commands which execute long running tasks should be executed by clients asynchronously to avoid timeouts [4]. Additionally, device servers should delegate tasks to threads to make them responsive to other client requests. The TANGO API enables both the implementation of an asynchronous device server and TANGO client. In addition to this solution provided by TANGO, the SKA Control System implements its own asynchronous execution of LRCs with a queue manager and a reporting mechanism to inform clients of the status of their submitted request [5, 6]. This implementation satisfies the four non-functional requirements (NFRs) relevant for handling LRCs, viz.: performance, dependability, interoperability and usability. Table 1 outlines the NFRs. The objective of this research paper is to describe the design and implementation of LRCs in the SKA Control System.

LRC IN SKA TANGO DEVICES: DESIGN

An Overview of the SKA Telescope Control System

Figure 1 depicts the normal hierarchical nature of TANGO nodes regardless of commands being LRCs.

Node A, found at the top of the hierarchy of nodes, has a process, Process X, to execute. Based on the complexity of the task, Node A divides Process X into 4 subsections namely, X1, X2, X3 and X4. Node A then delegates the four sub-processes to downstream nodes called Node A1, Node A2, Node A3 and Node A4 respectively as can be seen in Fig. 1. The input of all these sub-processes is a command together with its arguments; variables; configuration settings and state of the downstream node it is assigned to from Node A's perspective. This is indicated through the downward directed arrows connecting each downstream node with Node A. The upward directed arrows, from the perspective of Node A, indicate the values returned from the downstream nodes as well as any external and observable state of the downstream nodes. Just as Node A is responsible for dividing Process X into smaller processes it is also responsible for the aggregation of the responses received back from all the downstream nodes running the processes as well as any overhead that may come with that. After responses are aggregated, Node A can then report back to an upstream node denoted as vertical dots above Node A. As mentioned before, this architecture of disseminating and aggregating tasks is command independent and it therefore still applies within the solution of the LRCs spoken about in this paper.

The NFRs described in Table 1 together with key considerations enumerated in the CS guidelines, namely synchrony, asynchrony and concurrency were the building blocks used to design and methodically implement LRCs in

* bojur@sarao.ac.za

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH1BCO04

Table 1: The NFRs Related to Objectively Handling LRCs

Criterion	Explanation
Performance	Responsiveness of TANGO devices need to be improved because in the current synchronous implementation while the device is executing a long running command it does not service other requests (attribute reads and writes; other commands, including another high priority, urgent command). TANGO devices should support more than one client.
Dependability	Error and fault reporting shall not be paused while the TANGO device is executing a command. Similarly, a device that waits on another device should still be able to report its state and status.
Interoperability	The telescope control system shall be able to interact with various components, including ones that have long response times.
Usability	The complexity of a client due to interaction and coupling with a device, and assumptions about the future state of the device should not be too high.

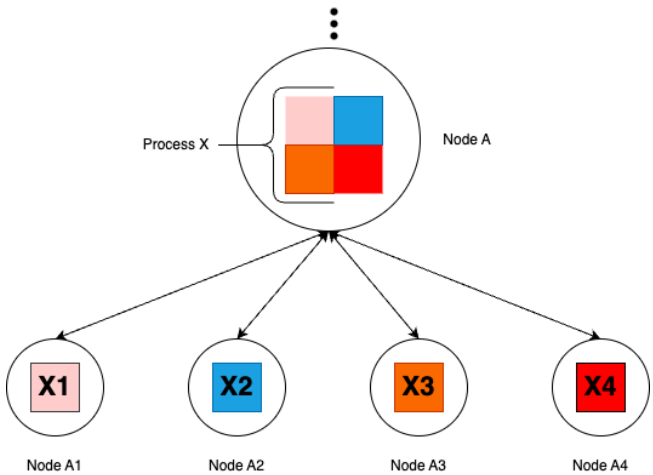

Figure 1: Diagrammatic illustration of the tango-example multi-level device and interaction among those devices.

SKA TANGO devices [2]. The next sections illustrate the design concept including client and server responsibilities.

Synchrony and Asynchrony

SKA TANGO devices use a hybrid approach to deal with LRCs. The design stipulates that synchronous interactions with a device should be available for commands that fall within the predicted response time of less than 10 milliseconds. Asynchronous interactions with a device should be available for commands that have the potential to be long running for a number of reasons [2]. This decision was based on Section 5.9.2 and 5.9.4.1 of the SKA CS Guidelines shown in Table 2.

Concurrency

The SKA CS Guidelines stipulate that an asynchronous interaction is required between a client and device to execute a single command on the device at a time, with the option of queuing commands on the device. [2] Concurrency is therefore enforced within the solution in the following ways:

- All LRCs sent towards a device are to be executed sequentially in a FIFO manner, not concurrently with each other.
- LRCs can be executed while concurrently servicing synchronous ”short running” requests such as other commands, attribute reads and writes and subscriptions to device and attribute events.

Queuing of Long Running Commands

The solution also stipulates that an SKA TANGO device should use a queue manager to manage an input queue for all LRCs. Commands issued are immediately enqueued and assessed against the state of the device for every dequeue operation to determine if the command is allowed to run. Table 3 explains the rules associated with the input queue.

Figure 2 illustrates the queuing mechanism and the flow of command execution for LRCs. Commands A and B are slow commands(LRCs) which will execute a task in a thread.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH1BCO04

Table 2: Extracts from the SKA CS Guidelines Informing the Hybrid Approach [2]

Section	Explanation
5.9.2	Use asynchronous commands for control that are known to be long running (due to a large number of components involved in the execution) and for control of operations that may have potential to be long running due to uncertainty of latency (network or I/O bound operations including operations that require the device that received the command from the client to invoke commands in other devices that may fail).
5.9.4.1	Use asynchronous commands sparingly because their implementation on both the client and device sides is more complicated. It can be added that a system with asynchronous communication may be less deterministic than one with synchronous communication.

Table 3: Input Queue Rules

Criterion	Explanation
Client Command Types	Only LRCs are queued. Commands that keep the client blocked until completed are not inserted in the queue.
Client Command Rejection	If an LRC, X, is being executed then another LRC, Y, will be rejected if a client sends it while X is being executed.
Execution	Items within a queue are executed in a FIFO manner. When not executing a command, the executor thread periodically checks the queue, removes the command at the head of the queue, and checks if the command can be accepted and executed in the current state. If not, the command is rejected. If the command can be accepted and executed in the current state, the Executor thread begins the command execution. A TANGO device will provide an unique identifier for each command invocation. This is necessary for reporting purposes to allow activity to be tracked without ambiguity.
Interrupt	The queue makes use of an abort command, “AbortCommands”, that clears the queue and aborts the currently running command
Client(s)	Clients can send multiple commands to a device before receiving back a response from the device. Since queues exist within a TANGO device’s domain a client does not need to have an internal queue to manage its requests to the TANGO device in question. More than one client can send commands to the same TANGO device. Clients can monitor the status of LRCs by using the “CheckLongRunningCommandStatus” command
Memory	On startup a TANGO device will have an empty queue. LRCs within a queue are not persisted.

Starting on the left hand side of the diagram in Fig. 2, when a command is received from a client it goes through a series validity tests. Not depicted within this diagram but evident in the processing of LRCs is the fact that a TANGO device will uniquely identify an LRC upon invocation so that the client can subscribe to change events on that device. Going back to Fig. 2, if all tests pass then the LRC is queued. Conversely, if any test fails the server receiver thread discards the command. In both cases the client will be able to keep track of the change events. After processing and queuing an LRC, the receiver thread can receive and process other incoming commands.

In the middle section of the diagram, when the server executor thread is ready to execute the next LRC command in the queue, it removes the command from the head of the queue. It then verifies that the command can be accepted in the current state. If not, then the command is discarded and communicated to the client through a change event. At that point the client then has the ability to react to this change event in their preferred manner. In the case where the LRC is accepted, the server executor thread will execute the command and its execution can result in a success or failure. Regardless of the outcome the client will be updated and can process the feedback from the change events. Upon completing the execution of an LRC the server executor will once

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH1BCO04

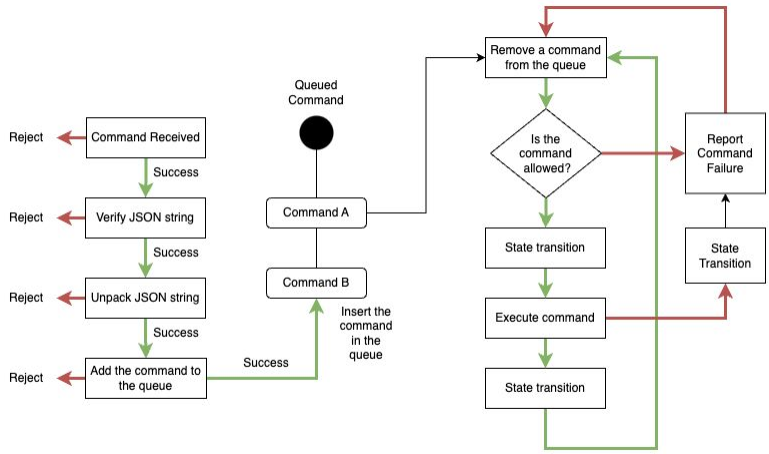

Figure 2: Queuing mechanism and the flow of command execution for long running commands.

again be in a state to process another queued LRC. The progress reporting delivered through change events is enabled in specific attributes: details follow in the next section.

LRC Progress Reporting

The solution also encompasses reporting, to monitor LRCs within the control system. SKA TANGO devices are programmed to report the progress status of an LRC to clients when any of these events occur.

- When an LRC is received.
- When an LRC begins its execution.
- When an LRC has completed execution regardless of the execution being successful, unsuccessful or when an abort command is executed.
- When an LRC is removed from a queue. This would happen when an abort command is executed.

In addition to the progress state of an LRC, it also covers the reporting of a return value of an LRC when it has completed its execution. As mentioned before in Table 3, a unique identifier is assigned to LRCs - this aids a client to successfully monitor their specific LRC since the identifier is globally unique across the network of SKA Tango devices.

LRC Attributes and Commands Clients can monitor and react to statuses of LRCs being executed through the use of progress attributes. In order to track status responses, clients need to subscribe to change events to receive the respective attributes. A subscription will inform a client of all attribute changes on an SKA TANGO device that they have subscribed to. For this reason, a client has the responsibility to ignor feedback related to identifiers they are uninterested in and pay attention to the ones that are of worth to them. To date there are 5 attributes used to track LRCs within this solution. The list that follows shows what attributes are available to clients:

- longRunningCommandIDsInQueue
  - Returns LRC IDs available in a queue.
- longRunningCommandsInQueue
  - Returns all the different types of LRCs present in the queue.
- longRunningCommandStatus
  - Returns the status of an LRC
- longRunningCommandProgress
  - Returns the progress of an LRC
- longRunningCommandResult
  - Returns the result of an LRC

A more indepth look at the available attributes can be found within the official Long Running Command documentation site [7].

Table 3 briefly introduced two commands, “AbortCommands” and “CheckLongRunningCommandStatus” that the solution uses. These commands aid the client in the management of LRCs. Their sequential execution descriptions are outlined next.

Interrupts are initiated by, “AbortCommands”. When invoked this is what happens:

- The current LRC is aborted.
- All enqueued LRCs are cleared out.
- The operation is communicated to the client irrespective of the command’s success or failure.

Statues of LRCs are checked through “CheckLongRunningCommandStatus”. In order to execute this command, a command identifier needs to be supplied as an input parameter. When it is executed one of the following state values are returned in response. Figure 3 shows LRC enumerated statuses.

Client Responsibilities The solution also imposes responsibilities on clients wanting to interact with SKA TANGO devices. Clients are responsible for monitoring the progress status of commands issued, as outlined in the LRC Progress Reporting section of this paper. In order to do this, clients need to subscribe to change events to receive command statuses and related attribute updates. Clients also need to implement a retry strategy to handle rejected commands that may appear because a queue is full or if an AbortCommands interrupt is invoked for instance. In the case when a client is interacting with different devices the onus is placed on the client to monitor the multiple executions. The implementation of how they should set up

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH1BCO04

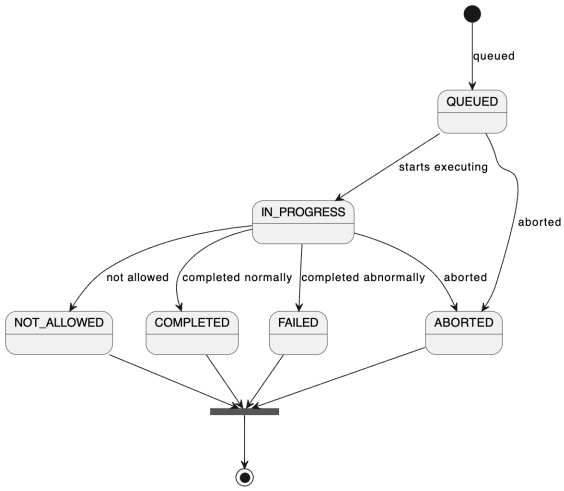

Figure 3: State diagram for LRC enumerated statuses

monitoring these devices internally is client specific and non-standard.

Implementation in ska-tango-base

This section outlines the solution with reference to this project's maintained and shared repository, ska-tango-base [5], which provides base classes to aid in the development of the control system. The ideas presented here echo what has been covered in the sections before but provides a more technical approach into the solution from a code perspective.

The base classes implement high level concepts, common approaches and design patterns outlined in the CS Guideline. This provides the wheels to harmonise the implementation of the devices and their interfaces. The repository exposes specialised classes which define commands as slow (long running) or fast and uses callbacks to update the long running command attributes to inform clients of the progress of their commands.

Component Managers The ska-tango-base device model envisages each TANGO device as comprising of two layers:

- A component manager is a pure python object that implements the monitoring and control functionality of the device.
- The TANGO layer wraps the component manager and provides it with a TANGO interface. Ideally this layer is as thin and as simple as possible, and simply maps
  - TANGO commands to component manager methods,
  - TANGO attributes to component manager properties,
  - component manager callbacks calls to TANGO events, and so on.

Component Classes TANGO commands are built on top of two basic command classes:

- FastCommand: a command class for commands that are fast and synchronous.
- Command tasks are queued for execution.
- SlowCommand: a command class for commands that are slow and therefore implemented asynchronously.

When a FastCommand is executed, the TANGO layer calls the corresponding method in the component manager, waits for the method to complete, and returns the result. This is appropriate because the component manager method is known to be ”fast”.

When a SlowCommand is executed, the TANGO layer:

- Creates a command ID for the command.
- Prepares a callback function that is associated with that command ID. When called with the appropriate updates on command status, progress and results, this callback function updates the TANGO layer's records of command status, progress and result for that command idea, resulting in updates to the following TANGO progress attributes longRunningCommandStatus, longRunningCommandProgress, and longRunningCommandResult.
- Invokes the corresponding component manager method, passing it the prepared callback.

The component manager method that is invoked by a SlowCommand:

- Starts or enqueues the work to be done asynchronously, passing it the callback
- Returns immediately.

The asynchronous work to be done is implemented so that the callback is called from time to time with updates on status, progress and finally the result.

An important property of this design is that the SlowCommand, and indeed the entire TANGO layer, is independent of any particular concurrency mechanism. The TANGO layer knows only that the work will be done asynchronously, and that it will be kept up to date on that asynchronous work by calls to the callback. On the other hand, the concurrency mechanism does not need to know about command IDs, or indeed anything to do with the TANGO LRC interface. It only knows that it has to call its callback with status, progress and result updates. Two concurrency mechanisms have been implemented to date:

- A TaskExecutor, based on a python standard ThreadPoolExecutor. In this mechanism, the work to be done is placed into a task queue that is serviced by some number of worker threads. Once a task reaches the front of the queue, it is picked up and run by a worker thread. The worker thread executes the task, calling the callback with updates from time to time.
- A Poller, in this mechanism, the work to be done, and the callback, are stored in memory that is shared with a poller. As part of its polling loop, the poller continually checks that shared memory, to help it decide what to do on the next poll. When the poller decides to execute a task on its next poll, it also calls the corresponding callback with updates.

The ska-tango-base repository has a reference implementation [8] of the component manager which can be used for teaching and testing purposes. It illustrates how a compon-

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH1BCO04

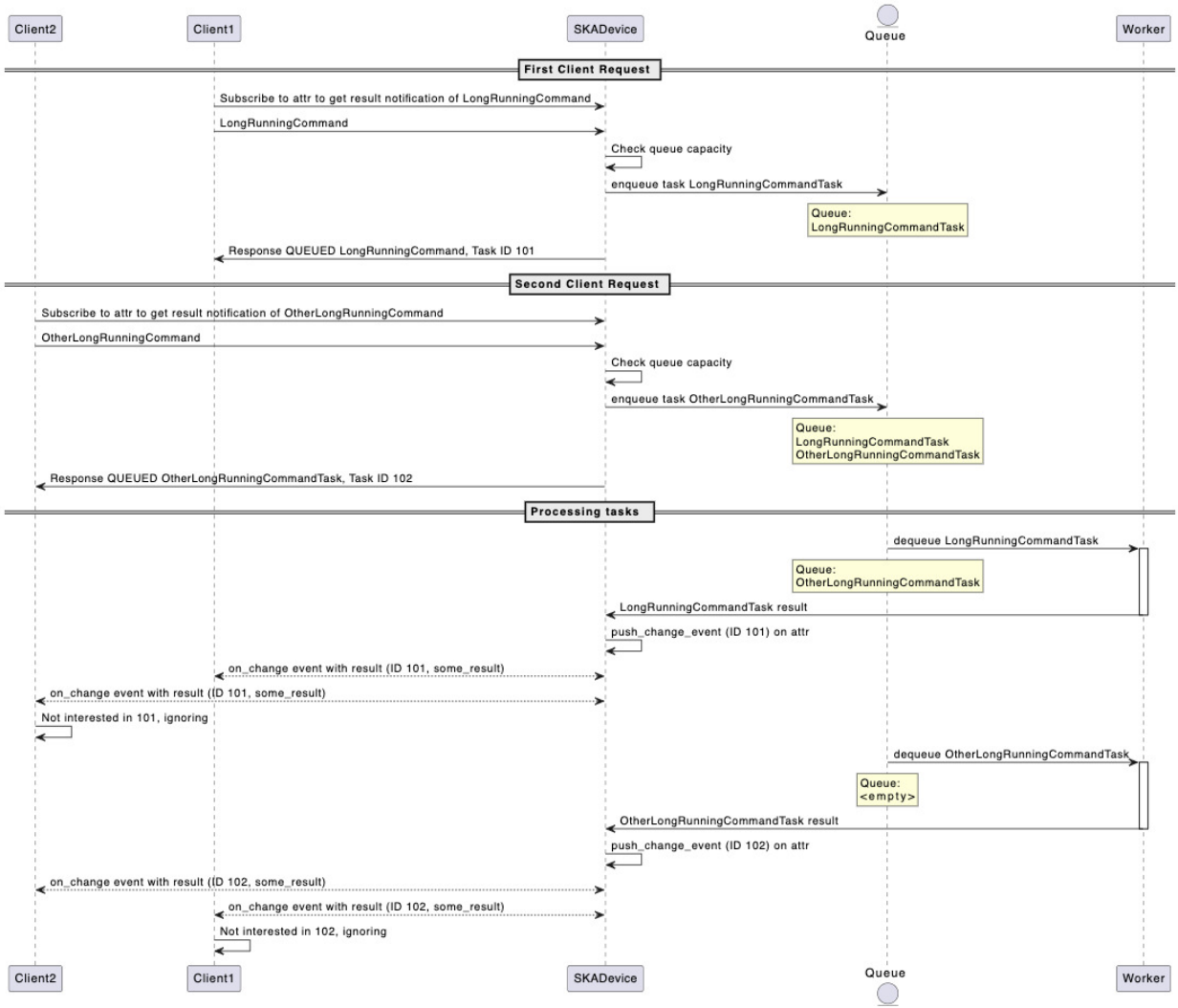

Figure 4: Multiple Clients Invoke Multiple Long Running Commands

ent manager is implemented and can be adapted for specific needs but not to be subclassed.

Example Scenario

This section provides an example of how an LRC within the SKA control system would be handled. The scenario presented is not exhaustive. Other scenarios are documented, but not shown within this paper, include:

- How multiple clients can invoke multiple long running commands while a non-long-running command is executed.
- How long running commands are aborted when multiple clients have invoked multiple long running commands.
- How a device restarts when there is a long running command in its queue.
- How the solution handles multi-level commands.
- How a sample client interaction looks like.

Figure 4 illustrates how SKA TANGO devices handle multiple clients invoking multiple LRCs. The basic flow of the sequence diagram is outline here [7]:

- Multiple clients invoke long running commands.
- Command tasks are queued for execution.
- Tasks are executed one at a time, first in, first out.
- Once completed (successfully/unsuccessfully) the client is notified via a change event on an attribute.

CONCLUSION

LRCs have a negative impact on the throughput of distributed systems if they are implemented as blocking requests. This necessitates the need to have asynchronous execution of such commands. The TANGO Controls library recognises this need and provides the avenue to write asynchronous devices. In the SKA control system, an alternative asynchronous solution has been implemented according to our CS guidelines. Top of the list for requirements comprises: performance, dependability, interoperability and usability. The solution presented makes use of an input queue, within all TANGO the devices, with a reporting mechanism to manage and track multiple commands sent towards each TANGO device. In order to differentiate the commands sent to a device, clients receive a unique identifiable code

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH1BCO04

which they can use to listen for progress updates. Lastly the solution also poses responsibilities on the client to subscribe to event changes to track the statuses of commands they have sent and also enforces that the client is responsible for aggregating the data received back from the TANGO devices.

ACKNOWLEDGEMENTS

Numerous contributions were used to develop this suitable LRC solution within the SKA network. Special thanks to those linked to Square Kilometer Array Observatory (SKAO), South African Radio Astronomy Observatory (SARAO) and Commonwealth Scientific and Industrial Research Organisation (CSIRO).

REFERENCES

[1] Square Kilometre Array (SKA), https://www.skao.int/

[2] L. Pivetta *et al.*, "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] TANGO, https://www.tango-controls.org/

[4] S. Vrcic, "Design Patterns for the SKA Control System", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 343–347. doi:10.18429/JACoWICALEPCS2021-TUBR02

[5] SKA TANGO Base, https://gitlab.com/ska-telescope/ska-tango-base

[6] https://developer.skatelescope.org/projects/ska-tango-base/en/latest/index.html

[7] https://developer.skao.int/projects/ska-tango-base/en/latest/guide/long_running_command.html

[8] https://gitlab.com/ska-telescope/ska-tango-base/-/tree/main/src/ska_tango_base/testing