ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUBL02

IMPLEMENTING AN EVENT TRACING SOLUTION WITH CONSISTENTLY FORMATTED LOGS FOR THE SKA TELESCOPE CONTROL SYSTEM

S.N. Twum*, W. Bode, A. F. Joubert, K. Madisa, P.S. Swart, A. J. Venter
SARAO, Cape Town, South Africa
A. Bridger, UKATC, Edinburgh; SKAO, Macclesfield

Abstract

The SKA telescope control system comprises several devices working on different hierarchies on different sites to provide a running observatory. The importance of logs, whether in its simplest form or correlated, in this system as well as any other distributed system is critical to fault finding and bug tracing. The SKA logging system will collect logs produced by numerous networked kubernetes deployments of devices and processes running a combination off-the-shelf, derived and bespoke software. The many moving parts of this complex system are delivered and maintained by different agile teams on multiple SKA Agile Release Trains. To facilitate an orderly and correlated generation of events in the running telescope, we implement a logging architecture which enforces consistently formatted logs with event tracing capability. We discuss the details of the architecture design and implementation, ending off with the limitations of the tracing solution in the context of a multiprocessing environment

PREVIEW TO THEORY AND SKA SYSTEM ARCHITECTURE

Observability and Monitoring in a Distributed System

Logs have long been the de facto approach used to sample parts and peek into the internal state of a running program. Coupled with metrics, monitoring can be done across a system to understand its health at any given time. Inasmuch as this age old approach has been very beneficial to developers, especially for debugging purposes, it is limited in its diagnostic ability in distributed environments. Monolithic applications are easily observable using only logs and metrics. But in the dawn of the era of microservice architecture, logging alone is not adequate to debug and probe the internal state of such a system. Distributed systems with different services and multiple instances of these services need a correlated view of events to troubleshoot errors. It now requires the use of logs, metrics and traces, all together producing the emergent quality of this new buzz word, "observability". J. Heather explains observability as inferring the internal state of a system from its external outputs. But it is not just the ability to see what is going on in your systems. It's the ability to make sense of it all, to gather and analyze the information you need to prevent incidents from happening, and to trace their path when they do happen, despite every safeguard, to make sure they don't happen again[1]. Full observability of a distributed system is a function of three pillars [2], namely:

- Logs: a snapshot of an event in a running system.
- Metrics: measurement of activities on a running system, e.g. CPU load.
- Tracing: A trace is a representation of a series of causally related distributed events that encode the end-to-end request flow through a distributed system [2].

An Overview of the SKA Telescope Control System Architecture

The SKA telescope control system will be a collection of software and services running from two sites which will be controlled from HQ in Jodrell Bank. The software consists of Tango Devices managing specific telescope hardware, and processes running all manner of software which are maintained by 17 or more teams on our Agile Release Trains (ARTs) [3]. The control system has a hierarchical structure to reduce complexity. The TANGO device is an abstraction hierarchy that has functional purpose at the top, and that goes down to physical form at the bottom. The frequency of intervention required at the different levels are different, increasing as you go downward the hierarchy (the Telescope Operator will exercise low frequency supervisory control, while at the lowest level you find real-time, closed loop feedback control) [4]. Control propagates downward (with fan-out) through this hierarchical structure. This is one source of causal path for events to propagate through the system.A specialisation of this hierarchical structure is that of the sub-array, an aggregation of telescope resources that are engaged for use in an observation [5]. Figure 1 illustrates the usage of sub-array nodes to control of the Mid telescopes.

During the lifetime of a running telescope, commands are triggered, events are fired, threads are spawned, several things are happening at the same time and it is not a trivial task to have end to end observability of the running system. This complex network exudes all the characteristics and challenges that are inherent in distributed systems. Though the various moving parts of the system are well tested, they are susceptible to faults and the ability to pin point a glitch to an exact root cause, following it through the various parts can only be provided in a distributed tracing system.

* stwum@sarao.ac.za

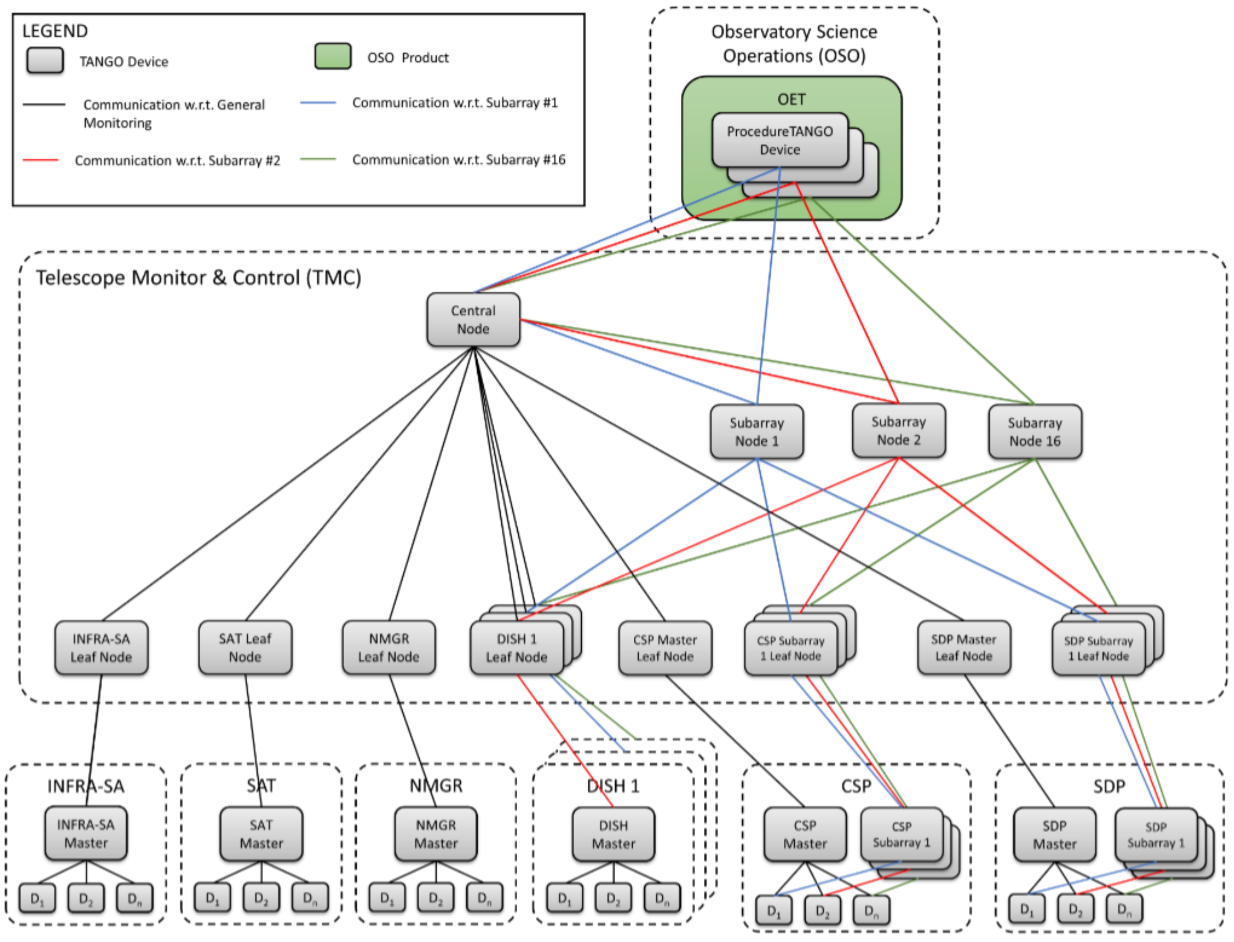


Figure 1: Hierarchy of tango devices demonstrating the control of the Mid telescopes using sub-array nodes [5].

In order to glean correlated debugging information for the SKA distributed services/devices architecture, we first adopted a standardised approach to logging across all packages maintained by the different teams and then, implemented a tracing solution in the form of transaction IDs to be used by Tango commands. The remaining sections discuss in detail the design decisions on our logging standard and the transaction IDs used to provision our tracing solution.

HARMONISING APPROACHES TO LOGGING IN THE SKA SERVICES/DEVICES SYSTEM

The SKA logging system collects, stores and retrieves logs from derived, off-the-shelf and bespoke software. All these different logs sources are maintained in different packages by different teams on the SKA Agile Release Train. Prior to the harmonised logs, each team had their own logging standard. After a review of the different logging approaches used by the teams, we provided a reference logging implementation which implements a standard log format. The details about the standard log format follow in the next section.

SKA Standard Log Format

The SKA log format ensure logs are emitted in a uniform manner to aid in troubleshooting and parsing. All processes executed inside a container log to `stdout`. For the emitted logs to be ingested into the logging system, they have to conform to the format below [6]:

```
VERSION"|"TIMESTAMP"|"SEVERITY"|"[THREAD-ID]"|
"[FUNCTION]"|"[LINE-LOC]"|"[TAGS]"|"MESSAGE LF
```

Example log in the ska log format is below:

```
1|2019-12-31T23:12:37.526Z|INFO||testpackage.
testmodule.TestDevice.test_fn|test.py#1|
tango-device:my/dev/name| Regular information
should be logged like this FYI
```

The log message format is not an extension of syslog/RFC5234 format. More on the SKA log message format can be found on the SKA Telescope Developer Portal [1].

To aid in filtering, Table 1 below describes tags that have been adopted as standard tags which may be found in a log message.

[1] https://developer.skatelescope.org/en/latest/tools/logging-format.html

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUBL02

Table 1: SKA Standard Tags for Logs [6]

Tag Name	Description	Example
tango-device	An identifier string in the form: `<facility>/<family>/<device>`. This corresponds with a Tango device name. • facility: The TANGO facility encodes the specific telescope (LOW or MID) and the telescope sub-system (SaDT, TM, SDP, CSP, Dish, LFAA, INFRA) • family: Family within facility • device: TANGO device name	`MID-D0125/rx/controller` • MID-D0125: where 0125 is the serial number of the Dish instance • rx: The Single Pixel Feed Receiver of the Dish • controller: The controller of the Single Pixel Feed Receiver
subsystem	For software that are not TANGO devices, the name of the telescope sub-system.	SDP
transaction_id	Transaction ID to associate logs from different systems relating to a distributed activity	`txn-t0001-20200928-000000010`

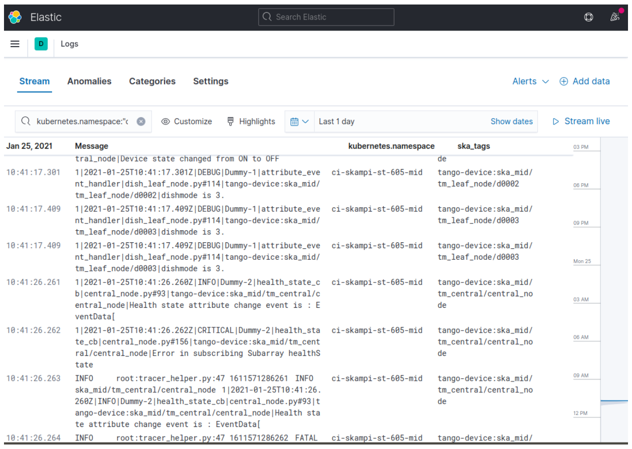
Design Motivation

The preliminary results gathered from analysing the code base of existing projects at the time drove the design of our current standard. We prioritised:

- Readability
- A sensible size of log length to provide useful information. The minimum required are:
  - timestamp
  - log level
  - extensible tags - a mechanism to specify arbitrary tags
  - the function in the source code where the log emanates from
  - the filename where the log call originates
  - the line number in the file
- Ability to parse the log message.

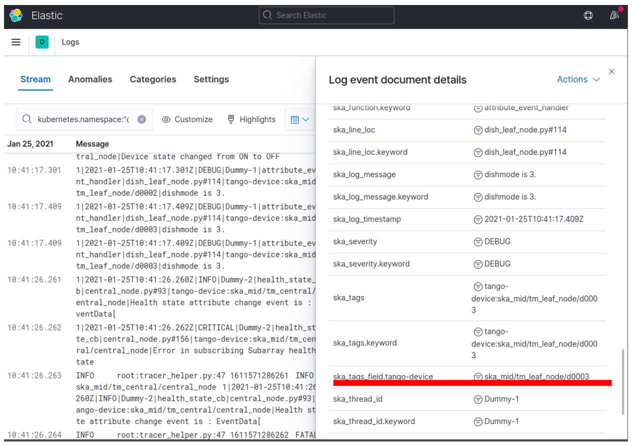
The SKA Logging Configuration Library

The SKA logging format is hosted in the ska-telescope gitlab repository as a python package. The package allows developers to configure logging for any module to have all applications log in a consistent format [7]. The SKA tags defined in Table 1 above can be added to a tags filter available in the library among other configurations. All new ska projects in both the Observation and Management Control and Data Processing ARTs use this logging library as do the existing ones before the format and its library were implemented.



```
Jan 25, 2021  Message                                              kubernetes.namespace   ska_tags
              tral_node|Device state changed from ON to OFF                              de
10:41:17.301  1|2021-01-25T10:41:17.301Z|DEBUG|Dummy-1|attribute_eve  ci-skampi-st-605-mid  tango-device:ska_mid/
              nt_handler|dish_leaf_node.py#114|tango-device:ska_mid/                          tm_leaf_node/d0002
              tm_leaf_node/d0002|dishmode is 3.
10:41:17.409  1|2021-01-25T10:41:17.409Z|DEBUG|Dummy-1|attribute_eve  ci-skampi-st-605-mid  tango-device:ska_mid/
              nt_handler|dish_leaf_node.py#114|tango-device:ska_mid/                          tm_leaf_node/d0003
              tm_leaf_node/d0003|dishmode is 3.
10:41:17.409  1|2021-01-25T10:41:17.409Z|DEBUG|Dummy-1|attribute_eve  ci-skampi-st-605-mid  tango-device:ska_mid/
              nt_handler|dish_leaf_node.py#114|tango-device:ska_mid/                          tm_leaf_node/d0003
              tm_leaf_node/d0003|dishmode is 3.
10:41:26.261  1|2021-01-25T10:41:26.260Z|INFO|Dummy-2|health_state_c  ci-skampi-st-605-mid  tango-device:ska_mid/
              b|central_node.py#93|tango-device:ska_mid/tm_central/c                          tm_central/central_no
              entral_node|Health state attribute change event is : E                          de
              ventData[
10:41:26.262  1|2021-01-25T10:41:26.262Z|CRITICAL|Dummy-2|health_sta  ci-skampi-st-605-mid  tango-device:ska_mid/
              te_cb|central_node.py#156|tango-device:ska_mid/tm_cent                          tm_central/central_no
              ral/central_node|Error in subscribing Subarray healthS                          de
              tate
10:41:26.263  INFO     root:tracer_helper.py:47 1611571286261  INFO    ci-skampi-st-605-mid  tango-device:ska_mid/
              ska_mid/tm_central/central_node 1|2021-01-25T10:41:26.                          tm_central/central_no
              260Z|INFO|Dummy-2|health_state_cb|central_node.py#93|t                          de
              ango-device:ska_mid/tm_central/central_node|Health sta
              te attribute change event is : EventData[
10:41:26.264  INFO     root:tracer_helper.py:47 1611571286262  FATAL   ci-skampi-st-605-mid  tango-device:ska_mid/
```

Figure 2: Kibana logs filtered using the value of the `ska_tags_field.tango-device` field [6].



```
Jan 25, 2021  Message
              tral_node|Device state changed from ON to OFF
10:41:17.301  1|2021-01-25T10:41:17.301Z|DEBUG|Dummy-1|attribute_ev
              nt_handler|dish_leaf_node.py#114|tango-device:ska_mid
              tm_leaf_node/d0002|dishmode is 3.
10:41:17.409  1|2021-01-25T10:41:17.409Z|DEBUG|Dummy-1|attribute_ev
              nt_handler|dish_leaf_node.py#114|tango-device:ska_mid
              tm_leaf_node/d0003|dishmode is 3.
10:41:17.409  1|2021-01-25T10:41:17.409Z|DEBUG|Dummy-1|attribute_ev
              nt_handler|dish_leaf_node.py#114|tango-device:ska_mid
              tm_leaf_node/d0003|dishmode is 3.
10:41:26.261  1|2021-01-25T10:41:26.260Z|INFO|Dummy-2|health_state_
              b|central_node.py#93|tango-device:ska_mid/tm_central/
              entral_node|Health state attribute change event is :
              ventData[
10:41:26.262  1|2021-01-25T10:41:26.262Z|CRITICAL|Dummy-2|health_st
              te_cb|central_node.py#156|tango-device:ska_mid/tm_cen
              ral/central_node|Error in subscribing Subarray health
              tate
10:41:26.263  INFO     root:tracer_helper.py:47 1611571286261  INFO
              ska_mid/tm_central/central_node 1|2021-01-25T10:41:26
              260Z|INFO|Dummy-2|health_state_cb|central_node.py#93|
              ango-device:ska_mid/tm_central/central_node|Health st
              te attribute change event is : EventData[
10:41:26.264  INFO     root:tracer_helper.py:47 1611571286262  FATAL
```

Log event document details

ska_function.keyword	attribute_event_handler
ska_line_loc	dish_leaf_node.py#114
ska_line_loc.keyword	dish_leaf_node.py#114
ska_log_message	dishmode is 3.
ska_log_message.keyword	dishmode is 3.
ska_log_timestamp	2021-01-25T10:41:17.409Z
ska_severity	DEBUG
ska_severity.keyword	DEBUG
ska_tags	tango-device:ska_mid/tm_leaf_node/d0003
ska_tags.keyword	tango-device:ska_mid/tm_leaf_node/d0003
ska_tags_field.tango-device	ska_mid/tm_leaf_node/d0003
ska_thread_id	Dummy-1
ska_thread_id.keyword	Dummy-1

Figure 3: Kibana logs with only messages with the value `ska_mid/tm_leaf_node/d0003` for the `ska_tags_field.tango-device` field [6].

TRACING EVENTS IN THE SKA CONTROL SYSTEM USING TRANSACTION IDS

The SKA control system churns out logs and aggregates metrics for system health monitoring; the missing piece to the full observability puzzle was the ability to obtain correlated log trails (tracing) of events. Tracing is the the only signal which explains the relationship between the different parts of the system which are managed by the different teams in our ARTs. It is nearly impossible to debug and discover patterns or correlations in the SKA control system searching through the gigantic logs being aggregated from the many services running in the kubernetes orchestrated environment. With the SKA formatted logs complete, it served as a scaffold to build the tracing solution by injecting a transaction ID in the logs. This design was driven by the snippet below:

```
with transaction('My Command') as txn_id:
    # do stuff
    ...
```

The idea in the snippet above is to ensure that any action performed within the context of the transaction will have record of an id for tracing. The transaction ID is cascaded down further requests until a trace is complete. In the SKA control system, the transaction context handler is implemented in a transactions python library [2] which relies on the SKA Logging [3] and SKA Unique Identifier (SKUID) [4] libraries.

SKA Unique Identifier Library

The SKUID library does more than just generate unique ids for transactions. It generates scan ids, entity ids and transaction IDs for telescope operations [8]. The scan and entity ids are out of this discussion's context. The transaction ID is generated every time a transaction context block is initialised. It is a unique value served by a remote or local unique id generator (which can possibly generate a duplicate). The remote URL has a `generator-id` following right after the "`txn`" prefix (e.g. `txn-t0001-20200914-123456789`) while the local generator has "`local`" right after the "`txn`" prefix (e.g. `txn-local-20200921-516590971`).

SKA Log Transactions Library

This library glues together the SKUID and logging packages to provide a transaction ID in a context handler which is injected into logs as a tag [9]. Within the context handler there is an enter and exit log entry which log the beginning and the end of a transaction using the generated transaction ID or a specified custom ID. In the event an exception occurs within the transaction, an exception log will be emitted with the transaction ID also. An example log on entry and exit of the transaction context in the ska log format looks like this:

[2] https://gitlab.com/ska-telescope/ska-ser-ska-ser-log-transactions
[3] https://gitlab.com/ska-telescope/ska-ser-logging
[4] https://gitlab.com/ska-telescope/ska-ser-skuid

```
1|2020-10-01T12:49:31.119Z|INFO|Thread-210|
log_entry|transactions.py#154||Transaction
[txn-local-20201001-981667980]: Enter[Command]
with parameters [{}] marker[52764]

1|2020-10-01T12:49:31.129Z|INFO|Thread-210
|log_exit|transactions.py#154||
Transaction[txn-local-20201001-981667980]:
Exit[Command] marker[52764]
```

To demonstrate the usage of the log-transaction library, a multi-level device has been added as an example in the ska-tango-examples [5] library. This multi-level device compromises a top-level device, four mid-level devices and one low-level device as depicted in Fig. 4.

The mid-level devices are triggered from commands in the top-level device and they in turn call the low-level device. During the interaction between these devices, the transaction ID generated at the top-level is propagated all the way to the low-level device and appears as tags in the logs as shown in Fig. 4. The rest of the logs are a combination of logs showing entry and exit into mid-level and low-level devices. When all mid-level devices with their associated low-level device triggers are finished, the trace ends with a log indicating an exit from the top-level device [10].

Tracing in a Multithreaded Environment

Though the log transactions library has support for async code it does not support a multithreaded case at the moment. Consider the scenarios in the code snippets below:

```
# SCENARIO 1
class Device(Device):

    def function_that_logs(self):
        do_something()
        log.info("logging something") # The
          ↪ same thread, so this should log
          ↪ a transaction ID

    @command
    def some_command(self):
        with transaction("name", parameters,
          ↪ logger=ska_logger) as
          ↪ transaction_id:
            function_that_logs()
```

[5] A project that demonstrates how to structure an SKA project that provides some simple Tango devices coded in PyTango. See https://gitlab.com/ska-telescope/ska-tango-examples

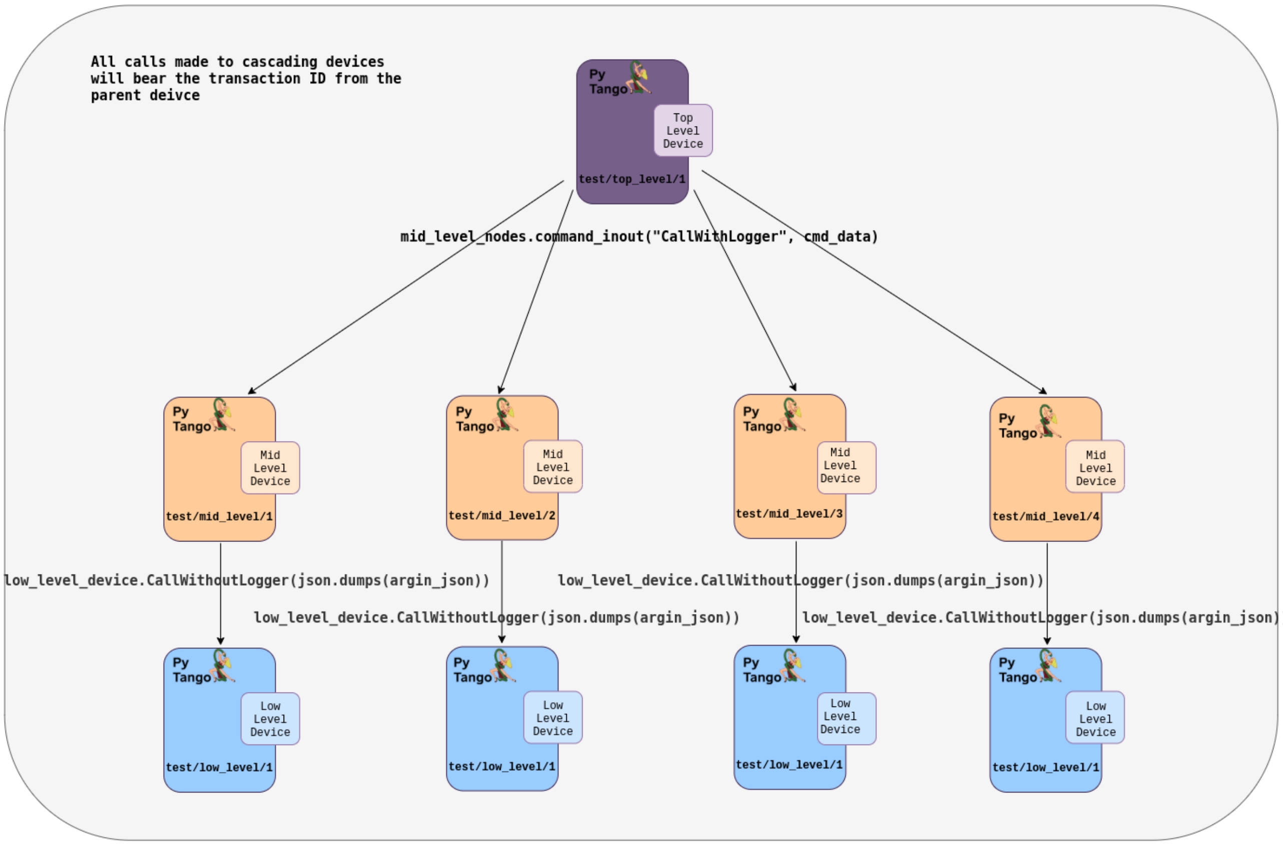

All calls made to cascading devices will bear the transaction ID from the parent deivce
Py Tango
Top Level Device
test/top_level/1
mid_level_nodes.command_inout("CallWithLogger", cmd_data)
Py Tango
Mid Level Device
test/mid_level/1
test/mid_level/2
test/mid_level/3
test/mid_level/4
low_level_device.CallWithoutLogger(json.dumps(argin_json))
low_level_device.CallWithoutLogger(json.dumps(argin_json))
low_level_device.CallWithoutLogger(json.dumps(argin_json))
low_level_device.CallWithoutLogger(json.dumps(argin_json))
Low Level Device
test/low_level/1
test/low_level/1
test/low_level/1
test/low_level/1

Figure 4: Diagrammatic illustration of the tango-example multi-level device and interaction among those devices.

```
 # SCENARIO 2
class Device(Device):

    def function_that_logs(self):
        do_something()
        log.info("logging something") #
          ↪ This will not have a
          ↪ transaction ID

    @command
    def some_command(self):
        with transaction("name", parameters,
          ↪ logger=ska_logger) as
          ↪ transaction_id:
            t = threading.Thread(
                target=function_that_logs,
                  ↪ args=(1,))
            t.start()
            t.join()
```

The code in SCENARIO 1 will produce a log just like in the example log above with the transaction ID in it but the log from SCENARIO 2 will not. The same is true for work passed off from a transaction to a thread managed by a thread pool. The thread lifetimes would exceed the transaction. This applies to code with or without asyncio or gevent.

FUTURE WORK

The enablers for the functioning of the SKA log transaction, namely SKA logging and SKA unique ID generator, do not have any left over features to be added except the local ID generation of the SKUID service to guarantee absolute uniqueness. This is trivial at the moment as the SKUID service URL is served up all the time in the integration environment. Having had all three pillars of distributed system implemented, the logical next step is to build the tooling we need to facilitate system diagnosis. The tooling will be available in all ART and will display to a user the flow of information and state across the MVP. At the time of implementing the log transaction, the SKA Telescope Control System is currently been demonstrated in an MVP and there is no clear evidence of that our troubleshooting would benefit from this feature. The need for it will be assessed and implemented as the project progresses, especially with AA0.5 release at hand.

CONCLUSION

To fully understand a distributed system requires us to have distributed tracing. Tracing compares to a database join with the contributing tables being the different paths the event travels through and each event having an ID on which the join is performed [11]. The SKA Control System like every other distributed system poses a big observability challenge. To be able to infer the internal state of both low and mid telescope control systems, we have developed libraries to ensure running applications have consistently formatted logs with tracing. All the work done are available in public repositories under the ska-telescope organisation in: ska-ser-logging, ska-ser-skuid and ska-ser-log-transactions.

ACKNOWLEDGEMENTS

We would like to acknowledge the National Research Foundation (NRF), the South African Radio Astronomy Observatory and the SKA Observatory for their support in carrying out this work under the Bridging Program and the SKA SAFe program.

REFERENCES

[1] J. Heather, *The New Stack, Cloud Native Observability for DevOps Teams*. Alex Williams, 2021.
[2] C. Sridharan, *Distributed Systems Observability: A Guide to Building Robust Systems*. O'Reilly Media, Inc., 2018.
[3] M. Bartolini, L. Brederode, M. Deegan, M. Miccolis, N. Rees and J. Santander-Vela, "Scaling Agile for the Square Kilometre Array", in *Proc. ICALEPCS'19*, (New York, NY, USA), ser. International Conference on Accelerator and Large Experimental Physics Control Systems, https://doi.org/10.18429/JACoW-ICALEPCS2019-WEPHA011, JACoW Publishing, Geneva, Switzerland, Aug. 2020, pp. 1079–1083, ISBN: 978-3-95450-209-7. DOI: 10.18429/JACoW-ICALEPCS2019-WEPHA011. https://jacow.org/icalepcs2019/papers/wepha011.pdf
[4] W. Findeisen, "Hierarchical control structures", *Control and Cybernetics*, 2000.
[5] P. Dewdney, "SKA1 design baseline description", Rep. SKA-TEL-SKO-0001075, Internal SKA document, 2000.
[6] SKA log message format, https://developer.skatelescope.org/en/latest/tools/logging-format.html
[7] SKA logging, https://developer.skao.int/projects/ska-ser-logging/en/latest/?badge=latest
[8] SKA unique identifiers, https://developer.skatelescope.org/projects/ska-ser-skuid/en/latest/?badge=latest
[9] SKA transaction logging, https://developer.skao.int/projects/ska-ser-log-transactions/en/latest/?badge=latest
[10] Multi level device in tango example, https://gitlab.com/ska-telescope/ska-tango-examples/-/merge_requests/28
[11] Why distributed tracing will replace (most) logging, https://www.youtube.com/watch?v=Hv98hU3nj0U