# MeerKAT CONTROL AND MONITORING SYSTEM ARCHITECTURE

Neilen Marais*, SKA South Africa, Cape Town

*Abstract*

The 64-dish MeerKAT radio telescope, currently under construction, comprises several loosely coupled independent subsystems, requiring a higher level Control and Monitoring (CAM) system to operate as a coherent instrument. Many control-system architectures are bus-like, clients directly receiving monitoring points from Input/Output Controllers; instead a multi-layer architecture based on point-to-point Karoo Array Telescope Control Protocol (KATCP) connections is used for MeerKAT. Clients (e.g. operators or scientists) only communicate directly with the outer layer of the telescope; only telescope interactions required for the given role are exposed to the user. The layers, interconnections, and how this architecture is used to meet telescope system requirements are described. Requirements include: Independently controllable telescope subsets; dynamically allocating telescope resources to individual users or observations, preventing the control of resources not allocated to them; commensal observations sharing resources; automatic detection of, and responses to, system-level alarm events; high level operator controls and health displays; automatic execution of scheduled observations.

## INTRODUCTION

The MeerKAT Control and Monitoring (CAM) subsystem is the high level control system that enables all the other subsytems to operate cohesively as a single telescope. Most of the CAM design principles were presented at ICALEPCS 2013 [1]. While much detail design and implementation has been done since then, the only major architectural addition to CAM is the concept of telescope subarrays and the subarray-manager components that implement them. An unusual feature for the control architecture of a large scientific instrument is the lack of any messaging middleware or message bus.

A brief MeerKAT status update is given, and a high level overview of the MeerKAT CAM architecture is presented with a description of how it is used to meet CAM requirements in practice. The scalability and reliability of this architecture are discussed.

## MeerKAT DESCRIPTION AND STATUS

MeerKAT will be a 64-receptor[1] aperture-synthesis inteferometric radio telescope array. Receptor antennas are offset Gregorian antennas with a 13.5m main reflector in a feed-low configuration. An offset optical configuration has been chosen because its unblocked aperture provides optimal optical performance and sensitivity with good rejection of unwanted radio frequency interference from satellites and terrestrial transmitters. It also enables the installation of multiple receiver systems in the primary and secondary focal areas and provides a number of operational advantages.

The antenna platform supports up to four receiver systems mounted on a carousel, allowing the telescope to switch frequency bands within minutes. The initial design committed to cryo-cooled L-band receivers. Some experimental uncooled K-band receivers are installed during commissioning. Since then, the MeerKAT project has committed to installing UHF-band receivers and the Max-Planck-Institute for Radio Astronomy has committed to funding and building S-band receivers for MeerKAT.

MeerKAT receptors directly digitise the received signal, without transmitting analogue signals to a central facility – a first for radio telescopes. A central Time and Frequency Reference (TFR) subsystem using a GPS-disciplined Rubidium clock provides a highly accurate absolute time reference and a sub-picosecond accurate clock signal for phase-coherent digitiser operation. The clock and timing pulses are distributed to the receptors via dedicated fibre links. The TFR will later be upgraded to a MASER timing source.

Signals are transmitted from from the receptor digitisers via multiple bonded 40 Gigabit Ethernet (40GbE) fibre links to the Correlator-Beamformer (CBF) subsystem in the Karoo Array Processor Building (KAPB) using the packetised SPEAD [2] format. A separate physical Ethernet network is used for CAM traffic from the receptors.

The CBF subsystem performs the first level of signal processing and data reduction using the SKARAB open FPGA platform [3], and multicast capable commodity 10/40GbE switches as a data fabric. It produces correlated visibilities for imaging, and beam-former voltage data that is consumed by the Science Processing (SP) subsystem and optionally by user supplied equipment provided by third parties. SP is a software subsystem running on general purpose computers in the KAPB, utilising GPU- and/or other forms of acceleration. SP processes the CBF output into images and other relevant science outputs, and also archives observation data.

Ancillary subsystems include: the MeerKAT site Camera subsystem; the weather monitoring subsystem; the KAPB building management subsystem that monitors ambient conditions and RFI-door intrusion detection where appropriate; the Power Distribution Units (PDUs) of several subsystems to facilitate emergency power shutdowns. These subsystems are not directly involved in observations but are monitored to ensure safe and reliable operation of the telescope.

All subsystems are coordinated and monitored by CAM using a logically separate CAM network, using Ethernet as a fieldbus. Communication with all other subsystems is via the Karoo Array Telescope communications protocol (KATCP) [4]. CAM Device Translators are used to provide a KATCP interface where this is not available natively.

* nmarais@ska.ac.za

[1] A receptor logically groups the subsystems of a single antenna dish.

ISBN 978-3-95450-148-9

MeerKAT's full functionality is divided into six Array Release (AR1-AR6) milestones with the goal of deploying AR1 in H1 2016. During 2014 all the earthworks, power and other civil infrastructure were completed, including installation of fibre to all the receptor sites and construction of the underground, RF-shielded KAPB that hosts the MeerKAT data centre. First light was achieved with the first of the 64 receptors, confirming that the system as designed will significantly exceed the 220 $m^2$/K system sensitivity requirement once all 64 receptors are installed. This would make MeerKAT the most sensitive L-band radio telescope in the world. During 2014 the CAM subsystem team completed feature development for the Receiver Test System (RTS), and commenced detail design and development of the MeerKAT AR2 features.

During 2015 the RTS was deployed to site. RTS is a standalone mini-telescope system that can work with up to four receptors for receptor acceptance testing and commissioning. RTS implements the minimum telescope functionality needed to commission receptors. Antennas under test are separated from the rest of the MeerKAT telescope data network by patching their physical data fibres into the RTS data-network switch, while the physical CAM network is shared between RTS and MeerKAT through configuration procedures.

Eight receptors are on site as of October 2015 and are at various stages of integration, commissioning and acceptance testing. Rolling construction and deployment of the remaining receptors will take place through 2016 and into 2017. New receptors are first added to RTS before graduating to the operational MeerKAT system for science use.

During 2015 the implementation of the AR2 milestone CAM functionality was completed and verified against simulators pending the completion of other subsystems' functionality. AR2 requires 16 science-ready receptors and a CBF that produces a limited selection of correlator data products, and an SP subsystem that ingests said products in real time. Furthermore, basic subarray functionality is required.

The early phase of MeerKAT development is mostly about constructing and commissioning the 64 Receptors and procuring a sufficient number of SKARAB boards to provide proposed CBF functionality. The later phases, while less visible, encompass the majority of the "construction" effort. Telescope functionality is added through software (CAM, SP and CBF local control) and FPGA firmware (CBF).

## COMMUNICATION LAYERS

The CAM architecture is divided into three layers of components as shown in Figure 1 with connections strictly from higher to lower levels, and a fourth category of CAM controller components that collaborate as models and controllers in the Model-View-Controller software model. Apart from **katportal** clients that use http and websockets, all connections are made using KATCP over TCP/IP; each directed arrow starts at a client making a TCP connection to a KATCP server running at a well known IP and TCP port. For subsystems external to CAM the IPs and ports are assigned by System Engineering (SE), while CAM components have IPs and ports assigned internally by CAM within the CAM IP range[2]. These IPs and ports are maintained in the CAM config (**katconfig**). All these connections are made via the CAM network[3].

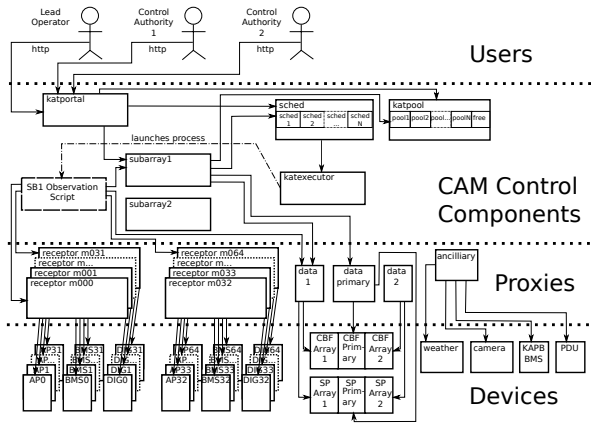


Figure 1: CAM Layers and Observation Control.

Members of the Devices layer each expose a KATCP interface, that is the interface between CAM and other subsystems; they are described using Interface Control Documents (ICDs) that are under SE configuration control. A single subsystem may have multiple instances, e.g. Antenna Positioner (AP) has an instance per receptor; each instance runs a separate KATCP server at a separate IP. These KATCP interfaces represent the controller of a specific piece of hardware, while the CBF and SP subsystems have their own internal control systems and present virtual devices. Devices make no KATCP connections via the CAM network, and act strictly as KATCP servers.

One level up are the CAM Proxies that make KATCP connections to devices, and act as KATCP servers to CAM controller components and to observation scripts. Each proxy exposes a single KATCP interface, but may connect to multiple logically related devices. Each proxy is managed by **katpool** (see next section) as a telescope resource. Devices are interrogated for KATCP sensors and requests, and these are proxied to the proxy's KATCP interface. Multiple instances of a proxy may exist, e.g. one receptor proxy per physical receptor, one data proxy per logical subarray. A proxy may implement special configuration/control for a device; e.g. the receptor proxies implement pointing corrections for the antennas.

Next come the CAM controller components that collectively implement the "intelligence" of the CAM subsystem. They may make KATCP connections to proxies and among

[2] The telescope network is treated as an infrastructure subsystem. IP ranges are assigned to various subsystems by SE.

[3] The CAM network is further segmented to avoid inappropriate communication between layers; the **katportal** component interfaces with the rest of the telescope LAN.





ISBN 978-3-95450-148-9

themselves. At the top are the external telescope users. They connect to the **katportal** component using web browsers over authenticated http connections. They are allowed telescope control on the basis of their assigned roles. The two roles considered here are: Lead Operator (LO); and Control Authority (CA). The LO manages the assignment of telescope resources to subarrays, and the assignment of logged-in users as CAs of a particular subarray. The CA manages observations on a specific subarray. See [5] for more details of the operator interface.

## OBSERVATION CONTROL

Figure 1 shows the main component interactions during observations. MeerKAT will initially have four[4] subarrays, but in the interest of space and clarity only two subarrays and the interactions of only one subarray are shown.

**katpool** is purely an accounting component; it keeps track of telescope resource allocations to subarrays and assignment to observations of resources within a subarray. Commensal observations are facilitated by only assigning controlling access of a resource to a single Observation Scheduling Block (SB)[5], while assigning non-controlling access to commensal observation SBs. **sched** manages the observation schedule of all the subarrays. **subarray1** is the subarray controller for "array1", managing its lifecycle.

The LO signs in to **katportal** and puts **subarray1** in configuration mode. She assigns (via the web GUI [5]) the SBs of the observations planned for her shift to **subarray1**, and assigns a subset of receptors and CBF product resources to **subarray1** as required by the scheduled observations. **katportal** requests **sched** (the CAM scheduler component) to assign the requested SBs to **subarray1** and requests **katpool** to allocate the requested resources to "pool1" (**subarray1's** resources). **katpool** moves the resources from the "free" pool to "pool1", making them unavailable to other subarrays. Satisfied with the configuration, the LO requests subarray activation. **katportal** requests **subarray1** to activate.

**subarray1** observes **pool** to get its resource assignment. Next it requests the CBF primary interface[6] to set up "array1" with the list of receptors to subscribe to. The CBF primary interface is used for high level configuration and lifecycle management of CBF arrays. It creates a secondary KATCP interface for "array1". The **data1** proxy connects to this interface and exposes it the higher layers. **subarray1** then requests **data1** to activate the requested CBF products. CBF now assigns and programs FPGA resources to produce the requested products for "array1". **subarray1** configures SP to receive the CBF data product in a similar way.

**subarray1** enters the active state, and the LO is notified. She delegates control to another user, making him the CA for **subarray1**; **katportal** will now relay all CA requests to **subarray1**. The CA puts the **subarray1** schedule in queue mode. **subarray1** relays this to **sched**. **sched** selects SB1 from the **subarray1** schedule and begins to activate it. It requests **katpool** to assign resources from "pool1" to SB1. **katpool** verifies SB1's resource requirements against those available in "pool1". If insufficient resources are available, the CA is notified, otherwise they are assigned to SB1.

Once resources are assigned, **sched** invites **katexecutor** to execute SB1, and informs **subarray1** that SB1 is now active. **katexecutor** examines SB1's instruction set to determine which observation script to run and the appropriate script parameters. It then launches the SB1 observation script in a subprocess. The observation script uses standard CAM Python library *katcorelib* to connect to **subarray1**; at startup the observation script has no knowledge of the telescope besides the address of **subarray1** and its SB ID code. Once connected, the script identifies itself as SB1. **subarray1** verifies that SB1 is indeed active, and replies with a datastructure describing the telescope resources assigned to SB1. This is passed to *katcorelib* which creates KATCP connections to the resources as required. The script is given a standard telescope object that exposes each KATCP resource (i.e. proxy connection) as a separate client object which exposes KATCP requests as function calls[7] and KATCP sensors as objects that can be polled, subscribed to, or monitored for a specific trigger value. Resource group objects allow all assigned receptors to be pointed to the same target with a single call, or the same sensor to be monitored across all the receptors. The observation script can now control its assigned resources until SB1 completes. Data products produced by the CBF are captured and processed by SP. SB1's metadata is also sent to SP so that the captured data can be accurately classified and archived.

Once SB1 completes, **sched** ensures via **katexecutor** that the script has terminated (ensuring that all SB1's KATCP connections are also closed), updates SB1's status to completed, and requests **katpool** to free the resources that were assigned to SB1. **sched** selects the next SB from the **subarray1** schedule, and repeats the process with the new SB. At the end of the observations, the CA or LO can request **subarray1** to be freed. The CBF is instructed to deprogram "array1", freeing up its FPGA resources, and **katpool** frees resources allocated to **subarray1** by moving them from "pool1" to the free pool.

## MONITORING AND INTERVENTION

Figure 2 shows the main component interactions for telescope monitoring and interventions and is a logical overlay for Figure 1 – the monitoring connections are always open, whether an observation is active or not.

A number of **katmonitor** instances[8] make KATCP connections to all CAM proxies and all CAM controller com-

[4] CAM could support more subarrays, but limitations in other subsystems, the design of human factors in the control room, and the number of receptors in the telescope limit the useful number of subarrays to four.

[5] For scheduling, observations are broken up into several indivisible SBs.

[6] Interactions with SP are not shown, but follow a similar pattern to CBF.

[7] *katcorelib* only exposes functionality that the observation script should have access to by pruning the telescope object. Scripts could easily bypass this mechanism, but they are vetted by CAM beforehand. A future implementation might introduce an intermediate CAM component to limit access.

[8] Multiple instances allow multi-process scaling.

ISBN 978-3-95450-148-9

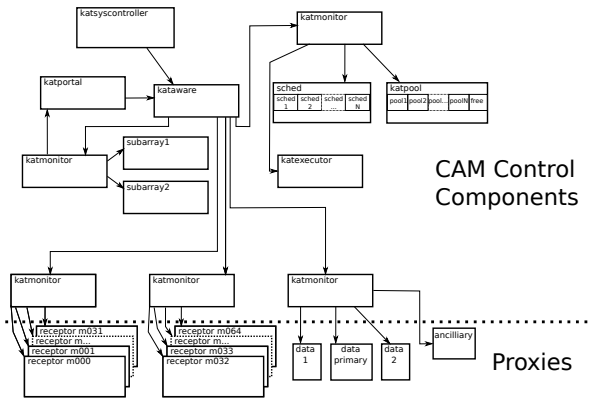


Figure 2: CAM Monitoring and Control Intervention.

ponents, and subscribe to every sensor by setting sensor strategies. **katmonitor** performs two functions on the received sensor samples: They are buffered in memory for the CAM sensor store [6], [7], and aggregation rules are applied in real time. Aggregation rules are defined in **katconfig** e.g. True if all the receivers on all receptors[9] are cold. The result of the rule is exposed as a new KATCP sensor on the **katmonitor** interface.

A single **kataware** instance observes all the **kataware** instances and applies alarm logic to the aggregate sensors. Alarm rules are read from **katconfig**. Each alarm has a rule for triggering an alarm level (e.g. WARN or ERROR), and an optional alarm action. **katportal** and **katsyscontroller** observe the **kataware** alarm sensors. **katportal** notifies logged in LO and CA users as appropriate; SMS, internet relay chat (IRC) or email notifications may also be configured. **katsyscontroller** implements the actions defined for the alarm and is responsible for taking telescope-wide action to ensure safe and reliable operation. It can make KATCP connections (not show in in Figure 2) to any proxies or CAM components needed to execute an action. Actions can be predefined (e.g. stow all receptor antennas due to high wind), or an arbitrary script file may be executed.

## DISCUSSION AND SCALING ANALYSIS

MeerKAT CAM avoids the need for dynamic device discovery since all the nodes in the system are described in **katconfig** (see [1] for more details). This works well since MeerKAT is fairly static in terms of physical configuration; receptors are fixed in place, and have space for a limited number of receivers. While MeerKAT may potentially be expanded with more receptors, receptors are fairly homogeneous in configuration. Only a limited amount of unique calibration data is required per unit, making it almost trivial to add more receptors to **katconfig**.

[9] A **katmonitor** instance can only apply aggregate rules to the subset of proxies it is connected to. A multi-level **katmonitor** design is planned that will remove this limitation.

Direct TCP connections between components avoid the complication and overhead of message-bus middleware. All critical CAM operations are idempotent, so the combination of TCP and KATCP's request / response or timeout paradigm has been found to provide sufficient reliability. Some messaging middleware provide message multicast features that might improve efficiency for sensors with multiple listeners. Analysis of the MeerKAT design shows that the **katmonitor** components are the only high-bandwidth consumers of sensor data, implying that sensor multicast would not save much traffic.

From the CAM perspective, scaling a radio telescope essentially implies managing a larger number of receptors and associated sensor data. The CAM proxies are inherently scalable, since a separate proxy process is used per receptor and no state is shared. **katmonitor** is similarly scalable. The CAM sensor store and **katportal** GUI are already designed for performance scalability [6], [5]; **katportal** might need some redesign due to human factors. Most of the CAM control components (**sched**, **katpool**, **subarray1...N** and **katexecutor**) have very low transaction rates or are not directly exposed to the number of receptors.

A potential bottleneck is the observation scripts since they need to address every receptor. However, only a handful of receptor interactions are required per SB (continuous antenna pointing is handled by the receptor proxies), so this approach should suffice even for thousands of receptors. A similar concern applies to **kataware**. The addition of a multi-process receptor instruction repeater component would address this bottleneck should it ever become a limiting factor.

## REFERENCES

[1] L. van den Heever, "MeerKAT Control and Monitoring - Design Concepts and Status", SKA SA, October 2013, ICALEPCS 2013 Proceedings MOCOAAB06.

[2] J. Manley, et al., "SPEAD: Streaming Protocol for Exchanging Astronomical Data", CASPER, June 2012, `https://casper.berkeley.edu/wiki/SPEAD`

[3] "Collaboration for Astronomy Signal Processing and Electronics Research", CASPER, `https://casper.berkeley.edu/wiki/Main_Page`

[4] S. Cross et al., "Guidelines for Communication with Devices", SKA SA, July 2012, `http://pythonhosted.org/katcp/_downloads/NRF-KAT7-6.0-IFCE-002-Rev5.pdf`

[5] T. Alberts and F. Joubert, "The MeerKAT Graphical User Interface Technology Stack", SKA SA, these proceedings THHC3O01.

[6] M.Slabber, "Overview of the monitoring data archive used on MeerKAT", these proceedings, THHD3O06.

[7] M.Slabber and M.T. Ockards, "Illustrate the Flow of Monitoring Data through the MeerKAT Telescope Control Software", WEPGF065 these proceedings, ICALEPCS'2015, Melbourne, Australia (2015).





ISBN 978-3-95450-148-9