MEERKAT CONTROL AND MONITORING - DESIGN CONCEPTS AND STATUS

L van den Heever, SKA SA, Cape Town, South Africa

Abstract

This presentation gives a status update of the MeerKAT Control & Monitoring (CAM) subsystem focusing on the design concepts and key design decisions. The presentation is supplemented by a poster of the current KAT-7 CAM system (including a demo). The vision for MeerKAT includes to: a) use Offset Gregorian antennas in a radio telescope array combined with optimized receiver technology in order to achieve superior imaging and maximum sensitivity, b) be the most sensitive instrument in the world in L-band, c) be an instrument that will be considered the benchmark for performance and reliability by the scientific community at large, and d) be a true precursor for the SKA that will be integrated into the SKA-mid dish array.

MEERKAT PROJECT UPDATE

KAT-7, 7-dish engineering prototype for MeerKAT, is already producing exciting science and is being operated 24x7. The first MeerKAT antenna will be on site by the end of this year and the first two Receptors will be fully integrated and ready for testing by April 2014. By December 2016 hardware for all 64 receptors will be installed and accepted and 32 antennas will be fully commissioned.

With regards the MeerKAT CAM subsystem we have completed the MeerKAT CAM Preliminary Design Review in July [1] & [2] with an international panel of domain experts, and the MeerKAT CAM team is currently developing the MeerKAT CAM requirements [3] for the MeerKAT Receptor Test System (first 4 receptors) to be ready by Jan 2014.

EVOLUTION OF MEERKAT CAM DESIGN

SKA South Africa started in 2004 with an XDM project, which was followed by the Fringe Finder project (the first 2 KAT-7 antennas), completed by end 2009. Then the full KAT-7 project followed and all 7 antennas are currently fully operational 24x7. SKA SA are now busy with the MeerKAT project (64 antennas in the Karoo by 2016). Many people were involved over the course of these projects in providing ideas for improvements and enhancements of the CAM subsystem as part of the CAM team and other teams in the project. The current MeerKAT CAM design is a result from all these efforts, a learning process on all these project, maturing our understanding of the requirements for a remotely operated radio telescope and most recently a concerted design effort to fully document and formally review the envisaged MeerKAT CAM design with a view towards potential scalability to the size of SKA Phase 1.

MEERKAT CAM DESIGN CONCEPTS AND KEY DESIGN DECISIONS

This section highlights the key concepts and subsystem-wide design decisions of the MeerKAT CAM subsystem.

Homogenous Node Management

The CAM subsystem ensures homogeneous node management (the term “node” is used for a virtualized container running a configured set of CAM processes).

- CAM nodes are virtualised across various physical servers (CAM hosts). The same suite of software is deployed on each CAM node.
- A single **headnode** is identified as the system controller for the set of nodes and has the only copy of the active configuration that is served from the headnode to all CAM nodes.
- Each CAM node (including the headnode) initially runs only a **katnodemanager** service that waits for the system controller to register the subset of CAM processes to run on that node (as retrieved from the active configuration on the headnode).
- This allows for seamless scaling when performance demands it as it is extremely easy to add new servers that host more nodes and distribute the processes amongst the new nodes. The only action required is updating the active configuration to identify the new nodes and the processes to run at each, and then restarting the system.

KATCP for Standardised Communications

Underlying the CAM concept of KAT-7 and MeerKAT is a standardized communications, reporting, controlling and logging protocol, the KAT communications protocol (KATCP) [4] & [5]. The KATCP protocol is a text based, human-readable protocol build on TCP/IP and supports flexible, run-time configuration by providing interrogation of sensors and requests/commands; these are used by the clients to discover the configuration of devices dynamically.

The KATCP protocol is specified as the CAM interface for all subcontracted and internal hardware devices and subsystems, as well as internal communication between CAM components. In cases where the subcontractor cannot deliver a KATCP interface, a Device Translator is

ISBN 978-3-95450-139-7

implemented by the CAM team to translate its specific protocol (like modbus, OPC, web-services, Ganglia metrics) to KATCP.

A key concept underlying the CAM implementation is the support provided in the KATCP protocol for **different sensor strategies** (sampling rates) per client. This enables each component to request sensor updates at the rate required by that component, e.g. the **kataware** component uses a different sampling rate to generate alarms than the **katmonitor** components use to archive historical sensor data .

Another key concept in KATCP is **standardised logging**. This allows devices that do not have access to storage to forward logs over their KATCP interface to the client. The log levels are standardized and the kind of information expected at each level is prescribed. The level of the KATCP logs to send over the interface can be set through the KATCP interface.

In summary:

- KATCP supports dynamic discovery through interrogation of monitoring points (KATCP sensors) and commands (KATCP requests)
- Interrogation of KATCP sensors provide details such as a description of the monitoring point, unit of measure, nominal, warning and error ranges, and absolute min/max values on sensors.
- Interrogation of KATCP requests includes help on parameters and usage examples.
- KATCP defines sensor sampling to always be a value/status combination where the status can be one of a defined set of status values e.g. nominal, warning, error, failure, inactive or unreachable (each of which is well-defined). Each sensor update is also time-stamped to indicate the time at which the sensor value was refreshed by the device.
- Interrogation also provides build state and version information.
- The KATCP guidelines defines standardized logging levels and logging format.
- KATCP is publicly released as a Python package on PyPi.
- KATCP devices support multiple connections
- KATCP sensors are exposed to all clients, but KATCP supports flexible reporting strategies per client for all sensors. Each client can define its own update strategy for sensors on a KATCP interface; e.g. periodic with a time period, event (on change), periodic but limited to a maximum update rate, etc.

KATCP is used for all CAM component, subsystem and hardware interfacing in MeerKAT. Most Inter-process communications internal to the CAM subsystem is also implemented through KATCP interfaces.

Standardised Central Logging

The KATCP guidelines also specifies standardized logging for devices/subsystems. The CAM proxy layer, gathers and stores all KATCP logs centrally. The level of logging exposed on each KATCP interface is configurable via KATCP request. This provides a consistent mechanism and formatting for system-wide logs and a central store of system logs to support fault finding and engineering tests. The CAM provides a web interface for viewing on-line system logs, which allows the log sources to be filtered by source and level by the user.

Proxy Layer and KATCP Device Translators

All hardware devices and MeerKAT subsystems are protected from direct access through a layer of proxies implemented by the CAM subsystem. All engineering/support/system components/tools connect via the proxy layer and not directly to hardware devices/subsystems. A proxy may expose one or more devices/subsystems, and the proxy layer provides a consistent level of aggregate monitoring information for all its hardware devices/subsystems. A proxy may implement special configuration/control for a device (e.g. the Receptor proxy implements pointing corrections for antenna pointing, and the Data proxy implements delay calculations and gain corrections for the Correlator Beamformer (CBF)). The proxy layer also gathers the KATCP logs from devices and passes it on as device-logs through Python logging to a centralized logger that stores and displays all logs centrally.

With the Device Translators and specification of KATCP interface for all subcontractors and subsystems, the complete instrument is managed, controlled, monitored and logged in a standard way, and exposed to the rest of the system through the proxy layer. This allow for the CAM team to develop a Simulator that simulates the KATCP interface and device behavior for each device/subsystem, so that the complete CAM system can be functionally exercised and qualified in a fully simulated environment.

Fully Simulated System

The CAM subsystem implements a fully simulated system up to the KATCP interface of each hardware device and subsystem. It is possible to run a configuration including only simulated devices, or any combination of real and simulated devices combined. This allows full software development, unit testing and integration testing, including CAM subsystem qualification testing without dependency on the availability of hardware.

Although the full KATCP interface for each device is implemented in the simulators, the actual functionality of all the hardware components are not fully implemented; each simulator implements behaviour to the level required by CAM integration testing. However, antenna pointing and modes are simulated with realistic timing, and a representative simulation of the data output of the correlator are implemented.

While the CAM team is responsible for developing most of the simulators, some of these device simulators are contractually delivered by the subsystem contractor to ensure that, given their knowledge of the device, the behaviour of the device is reflected with sufficient accuracy by the device simulator. Having a fully simulated system available is critical to CAM Qualification Testing and has proven to be one of the most valuable lessons learnt very early in SKA SA. Even though it takes time to develop the simulators and keep them aligned with changing interfaces, the CAM Team will not hesitate one moment if asked whether to do this or not.

Adaptive System based on Interrogation

One of the most powerful features of the KATCP protocol is its support to interrogate KATCP servers for monitoring points (KATCP sensors) and commands (KATCP requests). Interrogation of sensors and requests, down to device level, through KATCP, supports fluid run-time detection of system configuration; e.g. when a new monitoring point is added to any level of the system (including a hardware device), the rest of the CAM automatically discovers the change on-line on connections, and includes it in the monitor store and in its interfaces.

The CAM by design builds on this in-time interrogation and extends that by adapting to the discovered interface in real-time. All detected sensors are automatically sampled and monitored, included in the archive, aggregate reporting, and even health and status displays through rolled up sensors, without the need for any configuration. Sensor displays and archive displays automatically adjust to present the sensors detected during interrogation without having to manually change the displays in any way. CAM also exposes all discovered requests in its low level device control. This allows for seamless integration as new versions of controllers/hardware are rolled out with different sensors and requests.

The CAM's adaptability to the underlying system changes ensures flexibility during the construction phase where new antennas are constructed and rolled out periodically. This has proven extremely useful during testing, integration and commissioning of KAT-7.

Even the CAM health and status operator displays have been designed to automatically adjust to report on the receptors found in the system configuration, without a need to recompile or change the display configuration when a receptor is added or removed. Likewise the CAM sensor graph display that is used to extract archived monitoring data has been designed to automatically present all discovered sensors without the need to recompile or reconfigure any CAM components.

Flexible Central System Configuration

The CAM provides a powerful and flexible system configuration in human readable text files to support integration and incremental rollout of receptors. It caters for any combination of real and simulated devices as CAM "sites" and is templated using **Genshi** [6] for easy setup and maintenance. Each CAM site has a single central active configuration, deployed only on the headnode, which is served to all CAM nodes in the CAM site by a **katconfserver** component running on headnode.

The system configuration also includes configurable health displays, aggregate sensors with user defined programmatic rules, sampling strategies for monitoring and archiving and alarm configurations and actions.

Soft Real-time Control with Ethernet as a Field Bus

Soft real-time implies that there will be no time critical control loops in the MeerKAT CAM software and, where necessary, real-time control is decentralized and pushed down to devices. The CAM subsystem issues commands to devices with a specific timestamp; the exact timing is handled by the device in question. Synchronised execution/scheduling is implemented by syncing time from a central Timing and Frequency Reference across all devices and specifying the same start time, which is then independently handled by each node.

Hierarchical and Distributed Monitoring

For MeerKAT, the KATCP guidelines have been extended to include consistent failure reporting on each device through a proper FMECA process, as well as standardised device-status reporting for health monitoring. By dictating that high level failure sensors and health monitoring sensors are to be implemented by each KATCP device or subsystem, only a relatively small set of standard health and failure sensors have to be routinely managed and monitored, improving scalability. This simplifies the design of e.g. the health-monitoring GUI display, since all devices report consistently; the lower level device sensors (e.g. individual temperature sensors) are also available to the system, allowing hierarchical drill down when more detailed information is required. Furthermore, it gives the parties with the best knowledge of device behaviour and failure modes and their impact on the overall telescope system (the KATCP device vendors and SE) the capability to incorporate this knowledge directly into the CAM subsystem.

Each proxy also rolls up health information for communication, sensors and functional status through aggregate sensors that indicate the worst status of all communications (comms.ok) and the worst status of all sensors (sensors.ok). Each KATCP device reports an evaluation by the unit itself of whether it is still functioning correctly or not, notwithstanding the presence of warnings or errors on some sensors (device_status as ok, degraded or fail). These provide single points of monitoring to roll up in hierarchical health reporting and

ISBN 978-3-95450-139-7

availability and a single mechanism to display sensors for fault finding; drill down and interrogation of lower levels are only required when errors are reported. Failure detection sensors are further supported by standardised failure detection logging described in the KATCP guidelines.

Similar to proxies that consolidate reporting across all the devices they control, each node manager consolidates reporting across all CAM processes it manages. Standard sensors are exposed for each process (e.g. process_ok, process_running) in the same way as the aggregate reporting is done on the proxy level (e.g. sensors.ok, comms.ok and all.ok), and is thus reflected on the health display.

Distributed monitoring is implemented by running a local **katmonitor** component on each CAM node that monitors and archives the sensors of all the CAM proxies and other CAM components running on that node. The katmonitor instance writes its monitoring points to a central **katstore** archive server through network file system mounts routed through the bulk network, avoiding network traffic bottlenecks. Any new nodes added to the system are automatically included in the system monitoring and archiving by simply adding the node to the configuration and running an instance of **katmonitor** on that nod. The new monitoring points are automatically included into the rest of the system through interrogation and the adaptive design of the CAM subsystem.

DEVELOPMENT ENVIRONMENT

Software Development Environment

The current KAT-7 CAM system is developed in **Python** and MeerKAT CAM will be built on the KAT-7 CAM architecture with **PostgreSQL** [7] databases.

The Debian GNU/Linux based **Proxmox** [8] VE hypervisor OS is used for server virtualisation. Deployment is managed using the Python library, **Fabric** [9], for scripting the automated deployment in conjunction with an internally developed host configuration database. **Ubuntu LTS** [10] is used as the standard operating system throughout the CAM system..

Internal real-time chat is facilitated by an internal **IRC** [11] chat server. The IRC server is also used to broadcast critical alarms, since the active operator should always be on IRC. The **Mantis** [12] bug tracker tracking software is used for issue tracking during CAM development, and also for tracking error or bug reports for the telescope as a whole.

Version control is through **Subversion** (svn) [13] and our continuous build server is based on **Jenkins** [14].

On-line documentation (including CAM Architecture Descriptions, Component Specification Records, Component Design Records, Version Description Docuements and Deployment Procedure) is in **ReStructured Text** (RST) [15] format built with **Sphinx** [16].

Hardware Environment

The CAM system is hosted on standard COTS servers. Currently development is proceeding on DELL R410 servers, but in principle any PC servers could be used. CAM software is, however, not run directly on the CAM servers but on virtual CAM nodes using Proxmox. Persistent storage is hosted by an NFS server. The NFS server will be backed by a Storage Area Network (SAN) provided by the MeerKAT Science Processing subsystem.

FINAL NOTE

More MeerKAT talks and poster

The MeerKAT team is also presenting another talk "Virtualization and Deployment Management for the KAT-7 / MeerKAT Control and Monitoring System" [17] and there is a poster too [18] at this conference.

REFERENCES

[1] L van den Heever et al, "MeerKAT CAM Design Description, Rev 1", SKA SA, August 2013, http://tinyurl.com/MeerKAT-CAM-Public-Docs
[2] L van den Heever et al, "MeerKAT CAM Development and Qualification Plan, Rev 12, SKA SA, August 2013, http://tinyurl.com/MeerKAT-CAM-Public-Docs
[3] L van den Heever et al, "MeerKAT CAM Requirement Specification, Rev 2", SKA SA, August 2013, http://tinyurl.com/MeerKAT-CAM-Public-Docs
[4] KATCP Guidelines http://pythonhosted.org/katcp/_downloads/NRF-KAT7-6.0-IFCE-002-Rev5.pdf
[5] KATCP Python Library http://pythonhosted.org/katcp
[6] Genshi templating Python library http://genshi.edgewall.org/
[7] PostgreSQL open-source database http://en.wikipedia.org/wiki/PostgreSQL
[8] Proxmox server virtualisation management solution http://pve.proxmox.com
[9] Fabric application deployment Python library http://docs.fabfile.org/
[10] Ubuntu LTS Long Term Support release https://wiki.ubuntu.com/LTS
[11] IRC Internet Relay Chat http://en.wikipedia.org/wiki/Internet_Relay_Chat
[12] Mantis issue tracker http://www.mantisbt.org/
[13] Subversion http://subversion.apache.org/
[14] Jenkins continuous integration server http://jenkins-ci.org/
[15] RST reStructuredText Markup Specification http://docutils.sourceforge.net/rst.html
[16] Sphinx Python documentation generator http://sphinx-doc.org/
[17] N. Marais, "Virtualization and Deployment Management for the KAT-7 / MeerKAT Control and Monitoring System", THCOBA06, these proceedings.
[18] C. de Villiers, "MeerKAT Poster and Demo", TUPPC023, these proceedings.