MWA Metadata, Telemetry, Monitor and Control - design, implementation, and lessons learned

Design Goals:

- That it should be possible to determine the configuration, desired state, and actual state of the telescope both at the current instant, and at any time in the past, from the M&C database. Past configuration and state data should be just as easy to read (in software) as the current configuration. Archiving log files and keeping configuration files in a 'git' repository isn't good enough.
- That the M&C system should be able to reach every hardware and software setting in the array, so that the same scheduling system can be used for both science observations and engineering tests. This has been incredibly useful for us. For example, it allows us to schedule daily engineering tests using the same interface as normal observations. It has also allowed observational techniques we never imagined would be useful when the M&C system was designed.
- That the schedule controller would do its best to carry out observations even if some elements (receivers, tiles) were not communicating, or had been flagged as non-functional, rather than failing with an error. The telescope will never be 100% working, so the philosophy should always be to go ahead with whatever bits of the telescope are working at the time of the observation.
- That the specific choice of which tiles (and dipoles) to include in a scheduled observation should be instantiated as the observation is started, not days or weeks in advance when it's added to the schedule database. We run daily engineering tests to check the health of every dipole on every tile, and flag dipoles with 'bad' total power spectra to be excluded from the analogue summing junction inside the beamformer at the time of the observation. We have the same ability to exclude tiles so that no data is recorded for them, but in practice, it's never actually been used or useful. Instead, we record data from all working tiles, all the time, and choose which tiles to flag out of the observation at the time the data is reduced.
- That the M&C system should be able to mitigate the effects of hardware failures. For example, the schedule can specify which individual dipoles in each tile should be switched out of the final sum for a tile, and which tiles should be physically powered down to prevent emitted RFI. Emitted RFI from tiles was only a problem during early testing. The M&C system still has the capability to power down specific tiles for an observation, but we haven't had to use it since ~2009.
- To make the M&C system a collection of small, inter-operating tools rather than a monolithic system, and to use existing open-source applications, libraries and tools wherever possible. MWA software was written by many different scientists and students, most only working on it part-time, and none of them professional software developers. Many people were only involved with the project for a year or two. Using many small scripts, daemons and tools reduced the startup time for new people working on the code, and minimised the damage when they left.

LESSON LEARNED: Pick one programming language to work in… and make it Python. It is absolutely essential for astronomers to work closely with professional software developers on the telescope control software. It's easy to find developers who can work in Python. It's pretty much impossible to find astronomers who can work in Java.

We divide M&C into three main areas:

- Telescope ***configuration*** (physical locations and connectivity of tiles, receivers, cables, etc).
- Telescope ***state*** (any hardware and software settings that can be changed while running, for observations or engineering tests).
- Telescope ***telemetry*** (sensor data describing the health or performance of the hardware and software).

Telescope Configuration:

- This covers any aspect of the telescope hardware or software that can't be changed at run-time, in software.
- Examples include physical positions of tiles, connectivity (what tile is connected to which port on which receiver, software version numbers, data paths through the correlator, etc).
- The actual, physical configuration can only be changed by human beings.
- The M&C system needs a representation of that physical configuration, and that representation must be kept up to date, using:
    - A web GUI for entering tile positions, cable types and lengths, connectivity, etc, into a PostgreSQL database.
    - Automatic detection using tile/receiver/etc numbers embedded in the data stream.
- **Manually entering and updating the configuration is boring, time consuming, and prone to human error. The more the system can auto-discover the better.**

LESSON LEARNED: Wherever possible, give every digital hardware component a unique serial number, and every software component a version number. Pass all those serial numbers and version numbers along in the data stream, and record them with the data! I cannot stress this enough. Digital serial number chips cost a few cents, and embedding this data in packet headers or a side-channel of metadata is a negligible cost compared to the total data traffic.

Telescope State:

- Represents every hardware and software setting that can be changed at run-time, either for observations or for engineering tests.

- Examples include analogue attenuations, channel selection, beamformer parameters, time/frequency averaging settings, whether data is being captured or not, etc.
- Every setting has a *desired* state (what the user wanted it to be, at any particular time) and an *actual* state (what it was actually known to be at that time, or that the actual hardware state was unknown).
- The reality of working with a 'large N' telescope means that the desired state of the telescope, as defined by the observing schedule, will never completely match the actual state.
- The state (desired and actual) is stored in a PostgreSQL database, and every row in every table contains a time range (starttime and stoptime columns) for which that state entry applies.
- The contents of the 'desired state' tables form the telescope *schedule*, defining what it should be, or should have been, doing at any given time.
- One or more 'observation controllers' run continuously, reading the 'desired state' tables and communicating with all the hardware and software elements of the telescope to command them to change state as and when the schedule specifies.
- The observation controller/s, along with other software monitoring the status of different elements, maintain another set of tables with the *actual* state of every element of the telescope, at all times.
- Database constraints prevent entries in the past from being modified. For current observations (if 'starttime' is in the past, and 'stoptime' is in the future) only the 'stoptime' column can be changed.

LESSON LEARNED: Part of the process of requesting an observation should be defining 'minimum useful' thresholds for that observation, in terms of number of elements working, RFI level, etc, because a 100% functional telescope will never exist. This can be used in the longer term (to decide when to schedule a particular observing block) and in real time (to decide whether to actually record data or not, given the current observation and telescope state).

Telescope Telemetry:

- Measurements of temperature, voltage, current, fan speed, humidity, network switch traffic, SQL database activity, etc, from around the telescope.
- They could be:
  - Measurements that need to be recorded with the data, and used during the data reduction (analogue beamformer temperatures, for example).
  - Measurements that might be used to diagnose problems found in the data at some later time, so should be saved in case they are needed.
  - Measurements that should be monitored in real-time to detect and mitigate faults (server fan speed, motherboard temperature, etc), but aren't worth keeping forever.
- We use:
  - Icinga2 for fault detection, reporting, and actions to prevent physical damage. It has a web interface to view faults and schedule downtime, alerts the operations team via email/SMS/etc, and triggers automatic actions like shutting down power supplies.
  - Graphite/Carbon for 'round-robin' databases of over 10000 time-series telemetry data sources, each tuned to store data for a certain time and time-average old data for longer periods. It also lets the user visualise the time series data in configurable web dashboards.
  - Observium to monitor port status and traffic on all of our network switches, via SNMP.
  - PostgreSQL to record telemetry data that is (or might be) relevant to the telescope data products.
- Permanently archived data, stored with the output data products, includes beamformer temperatures recorded every few minutes, and full 0-300 MHz power spectra recorded for each tile approximately once every 64 seconds for RFI monitoring.

LESSON LEARNED: Decide early how to manage these three classes of telemetry (SQL database, round-robin database, text files, etc), but be prepared for measurements to change category after a few years. You don't want to have to scrape the data from years of archived log files (in different formats) when someone decides that it's important…

LESSON LEARNED: If you store telemetry data in a text file, don't just write it to a log file embedded in the usual error/debugging messages. Use a human readable log file for errors and debugging, and a CSV file (or similar) for telemetry data.

LESSON LEARNED: However the telemetry data is stored, build the interface for visualising that data at the same time you start storing it. Storing data that you can't visualise or use is pointless.

LESSON LEARNED: Give every discrete hardware unit at least one bright externally visible power LED, and if it contains a processor, flash an LED in the main loop to show that the software is running. Technicians need a quick and foolproof way to tell if a box is powered up and working, powered up but not working, or turned off, without opening the (RFI-tight, dozen-screw) box. Having software-controlled lights on the boxes we have out in the desert has probably already saved as much money in technician time as the capital cost of the hardware.

LESSON LEARNED: Software controlled lights are essential for internal rack-mounted gear too. There needs to be a way for a human to quickly verify that the physical locations and wiring of equipment (a

station of TPM's, for example) matches the internal M&C representation of that hardware. Something as simple as a light-chaser scrolling across the LED's of all of the TPMs in sequence would make a mis-identified bit of hardware stand out to the eye. Software controlled lights are also incredibly useful to identify faulty units that need to be swapped out, so a good unit isn't replaced by accident.

In Summary, looking ahead to the SKA:

- Treat all the specifications and requirements with a grain of salt – the prototype won't match the specs exactly, and the 'final' product will be controlling different hardware, and have different requirements.
- Make everything more flexible than you think it needs to be, because adding features retrospectively is really hard.
- Plan early on how the metadata (telescope configuration, state, and telemetry data) will be packaged and follow the data to archiving and the end user.
- Don't assume that M&C software is a product that will ever be finished and delivered. The hardware will evolve over time, and users will demand new features. The MWA has been fully operational since mid 2013, but the M&C software has evolved dramatically since then – paying back technical debt (rewriting thousands of lines of code), adding features, and supporting the end user with better metadata handling and access to data.