STATUS OF THE ASKAP MONITORING AND CONTROL SYSTEM

J.C. Guzman, CSIRO Astronomy and Space Science, Sydney, Australia

Abstract

The Australian Square Kilometre Array Pathfinder, or ASKAP, is CSIRO's new radio telescope currently under construction at the Murchison Radio astronomy Observatory (MRO) in Mid West region of Western Australia. As well as being a world-leading telescope in its own right, ASKAP will be an important testbed for the Square Kilometre Array. This paper gives a status update of the ASKAP project and provides a detailed look at the initial deployment of the monitoring and control system as well as major issues to be addressed in future software releases before the start of system commissioning later this year.

PROJECT UPDATE

ASKAP is located in the Shire of Murchison, a remote outback mid-west region of Western Australia, approximately 400 km northeast of Geraldton and 800 km north of Perth. The location also corresponds to the Australian SKA candidate core site. This region has been identified as ideal for radio-astronomy due to its very low population density and hence a lack of man-made radio signals that would otherwise interfere with weak astronomical signals. The site is now named the Murchison Radio-astronomy Observatory (MRO).

Approximately 75% of ASKAP time will be used as a survey telescope carrying out systematic observations of the entire southern sky. The rest of the time will be allocated to targeted observations either guest projects or target of opportunity observations. For more information about ASKAP project visit [1].

At the time of writing this paper there are 9 antennas installed, 6 of them have successfully completed their Site Acceptance Test. One of them (Antenna 29) has a Single Pixel Feed (SPF) L-band receiver and is being used as a part of the Australian Very Long Baseline Interferometry (VLBI) instrument.

All Integrated Product Teams (IPT) have passed their Critical Design Review and the project is well into production mode. The first full Phase Array Feed (PAF) front-end package, back-end electronics and control software has been installed on the Parkes 12m test-bed antenna in May 2011 and has been used in interferometry mode with the Parkes 64-m antenna to measure the PAF performance as well and to be a testdbed for the monitoring and control software system of ASKAP.

The next big milestone is the installation and first light of the Boolardy Engineering Test Array (BETA) system comprising of 6 ASKAP antennas at the MRO fully equipped with PAFs, front-end and back-end electronics, and monitoring and control software. Subsequent major project milestones are described in Table 1

Table 1: ASKAP Project Milestones

Milestone	Date
MRO infrastructure completed (including central site building and power for BETA)	Jan 2012
BETA "first light" and start of commissioning	Feb 2012
Construction of all 36 antennas complete	May 2012
Pawsey centre building (host of the ASKAP supercomputer) in Perth complete	Sep 2012
MRO power plant (ASKAP)	Dec 2012
ASKAP (with PAF) complete	Dec 2013

EVOLUTION OF THE ASKAP MONITORING AND CONTROL SOFTWARE

Software Architecture Update

After the Computing Preliminary Design Review (PDR) meeting in 2009, where we presented the first design of the ASKAP Software Architecture, and subsequent discussions with System Engineering and other IPTs, we agreed to extend the definition of the TOS (as defined in [2]). The new TOS contains all the software components related to monitoring and control of the facility and provides both raw data (visibilities) and telescope meta-data for the data processing stages.

The Central Processor (CP) group contains all components related to data reduction pipelines such as calibration and imaging, including services necessary to perform those tasks such as access to sky models and RFI data. This functional decomposition allowed us also to align to the overall ASKAP Architecture, on which TOS and CP subsystems were identified. Figure 1 presents a logical view of the current overall ASKAP software architecture.

Figure 1 also shows were EPICS lives within the software architecture. The actual implementation followed a more conventional tiered design, similar to other telescopes and particle accelerators, on which there is a middle layer providing local control and interfaces to the hardware sub-

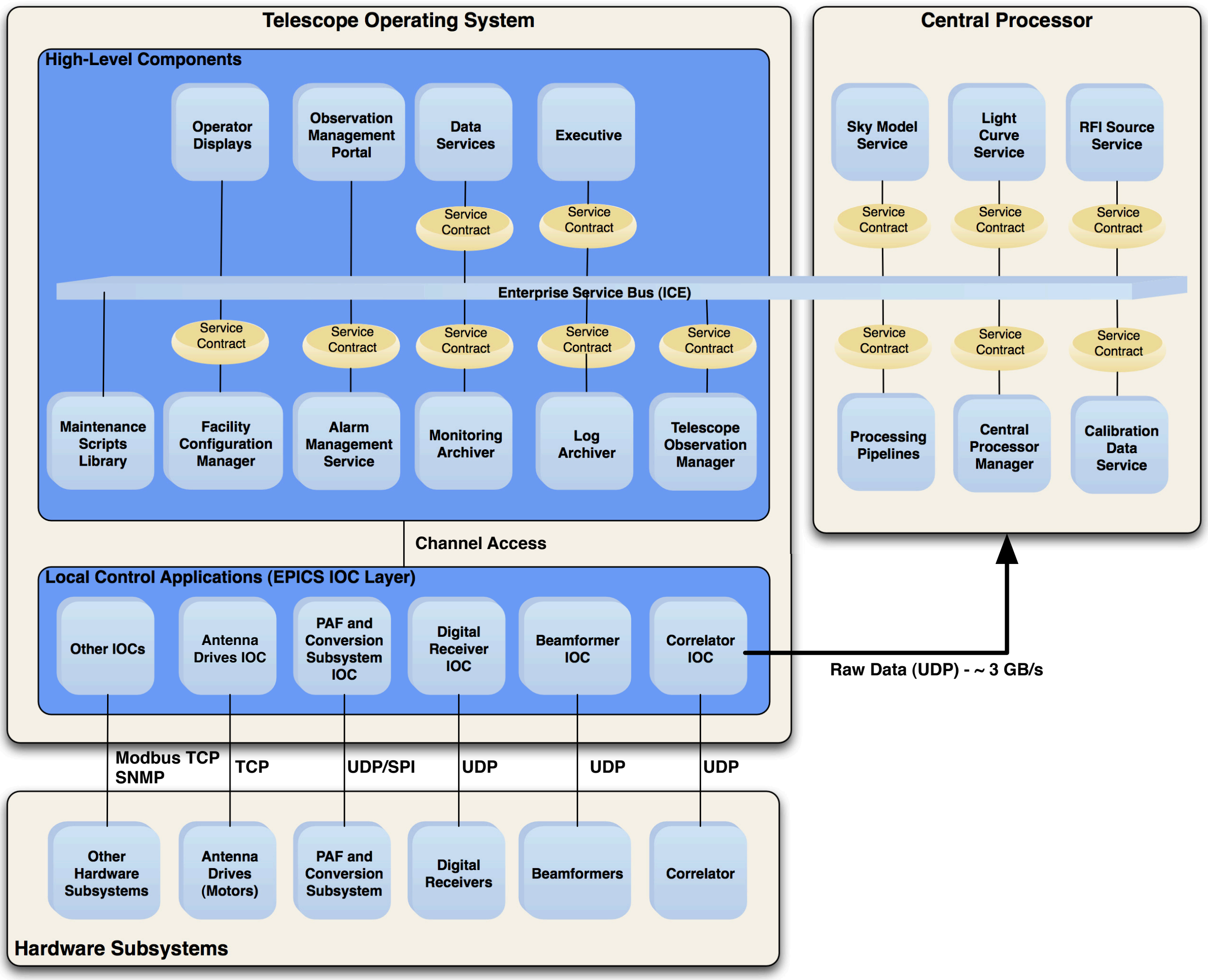

Figure 1: Logical view of the ASKAP software as implemented in latest version of TOS.

systems. This layer is implemented using EPICS IOC core. High-level components such as Monitoring Archiver, Telescope Observation Manager and Alarm Management Service use Channel Access to monitor and control the hardware subsystems. EPICS has several drivers already available for most common hardware field-bus such as SNMP and Modbus TCP. In case of our in-house UDP-based protocol (used in all our custom hardware) we have developed drivers using the EPICS C/C++ driver framework called ASYN [3].

A summary of the current control system specification is described in Table 2.

One of our requirements is to provide a flexible tool to perform maintenance operations on one or more hardware subsystems. We have developed a Python library for maintenance operations and system verification called Maintenance Script Library (MSL). This library is currently in used by engineers and instrument specialists to build specialised scripts that verifies one or more subsystems. This scripts can also be run by maintenance engineers after a piece of hardware has been replaced. The library supports concurrency and reporting. The Python library is built on top of PyEpics3 EPICS Python library developed by Matthew Neville [4].

TOS SOFTWARE RELEASES

The Build System

Since we rely heavily on several third-party software packages such as EPICS and ICE source code, we developed a powerful build system that ”wraps” several language-specific builders: scons (for C++ packages), EPICS make (for EPICS base, tools and applications), ant (for Java packages and applications), setuptools (for Python packages and applications) and autotools (for third-party packages that use autoconf). In addition, the build system supports recursive compilation, documentation generation (doxygen, Sphynx and javadocs), execution of unit and functional testing, release process via debian packages and Linux and OSX development environments.

The build system is implemented in Python and integrates also with Subversion (version control system) and

Table 2: ASKAP Monitoring and Control System Specifications

	BETA (6 Ant)	ASKAP (36 Ant)
Total Number of I/O Points	100,000	600,000
Total Number of Archived I/O Points	40,000	240,000
Monitoring data archival rate	130 GB/year	1 TB/year
Monitoring data archival rate max	1 kHz	1 kHz
Highest ”soft” control loop rate	1 Hz	1 Hz
Estimated number soft-IOC Linux computers	10	30
Science ”raw” data output rate	400 MB/s	3 GB/s

Hudson [7] (continuous integration tool).

The Development and Release Process

We have adopted an iterative and light-weight approach to our software development process, incorporating some best practices from agile methodologies. We do not use complicated Gantt Charts for our day-to-day software development planning, instead we have software release-based milestones. Every month, usually at the beginning of the month we produce a fully tested release of TOS software. Each release it is then installed and verified in the Parkes system. This monthly cycle follows the principle ”integrate early, integrate often”. This approach has been proven very useful, especially to adapt to requirement changes and external factors (hardware delay) on which we can arrange the amount of features that goes into one release or another.

Two tools has allowed us to put this methodology in practice and worth mention:

- Redmine [6], an open source project management tool that features issue tracking, milestone support, subversion integration and wiki. This tool is also being used by other ASKAP IPT and it has been a good way to do work exchange and tracking between IPTs.
- Hudson [7], an open source continuous integration tool, very easy to install, use and maintain. This tool allow us to perform continouous checking of build software as well as unit test coverage and some functional testing. Developers gets notified if they break the build.

KEY TECHNOLOGIES

EPICS

We have been using EPICS framework for the past 2 years and recently in production code in Parkes since July 2011. Overall we are very happy with the EPICS software and related tools, and more important with the openness and friendliness of the EPICS community.

Some of the EPICS tools we are currently using besides EPICS IOC core software are listed here:

- ASYN [3] for low-level EPICS driver development.
- Existing EPICS third-party drivers: devSNMP and modbus.
- PyEpics3 [4] as our Python Channel Access Client library.
- Alarm Handler (ALH). Offers a quick implementation (but limited) of the Alarm Management Service but soon to be replaced by most likely Control System Studio (CSS) BEAST.
- Sequencer and State Notation Language for state programming in the IOC applications.
- EDM for our Engineering GUIs, but starting to migrate engineering GUI development to CSS BOY.
- Control System Studio (CSS) [5]. We have just started to use CSS BOY for some of the engineering GUI and we are looking now to integrate CSS into our repository as well as some other CSS tools like BEAST, PV table, Data Browser.

ICE

We now have under our belt a good years worth of experience using ICE, and recently we have been discussing some of its short-comings (version 3.4.1). None of the short-comings are major issues and we decided that even though it may not be the most ”elegant” solution, our ICE implementation delivers do what we require in the area of high-level component communications.

Here we present some of the major issues, mainly related to ICE publish/subscribe implementation called IceStorm:

- Difficulty in debugging applications deployed in IceStorm. This is primarily due to the fact there are no observability tools which operate at the message or topic level.
- Lack of reliability features in IceStorm, particularly the inability for a subscriber to identify the fact that the broker/channel (IceStorm server) has gone away. There is no auto-reconnect feature as is present in other pub/sub middleware.
- No support for queues (only topics) so temporal decoupling is not possible.
- There is no guarantee of delivery of message. When a message is sent/published, if the subscriber is not running then the message is lost. IceStorm does not hold (or even persist) the messages and redeliver when the subscriber comes back up.

Despite these shortcomings, we have found the core functionality (Interface definition, RPC, Locator service) to be very good. Since the high-level component communication is not dominated by publish/subscribe communication these ICE issues are not critical, but we are looking into ways to improve them.

CURRENT AND FUTURE CHALLENGES

Over the past year and as we ramp up our software development one of the major challenges we encountered was how to deal with the creation of Interface Control Documentation (ICD). Since control software in our domain tends to cut across many subsystems and it is at the end of the data change, successfully delivery of fully tested software closely dependent on two things: ICD or something equivalent and the hardware to play.

An ICD is vital for developing the local control software or IOC, since it tells us with some degree how the hardware communicates and functions. We can also build emulator to assist the IOC development, integrate the IOC into the overall system for early testing and system debugging while we wait for the deliver of the full hardware.

ASKAP hardware/firmware developers responsible for production of hardware subsystems were not used (or did not have much experience) in producing ICDs. Their normal way of writing ICD and any formal documentation is to design the hardware, implement the firmware, implement some C-code software to verify the hardware and then write the ICD, by then it was too late for us and operators were forced to use the C-code program to operate the hardware while we debug and test IOC software. It took a lot of time and effort to come up with some mutual arrangement on which some form of ICD is written, even if not complete (it actually seldom is) so we can start developing, integrating and testing early into the overall system while they continue implement more functionality in the firmware.

Having an iterative approach with monthly releases have helped us greatly to adapt to changes in ICDs and late delivery of hardware.

Since we have to concentrate in the core software for start of BETA commissioning, we postponed some TOS feature to be developed over the next year or so, in particular:

- Integration of Control System Studio (CSS) into our software repository and the use of BEAST as our Alarm Management Service, with extension to capture alarms from High-level components via ICE.
- Implementation of the Facility Configuration Management.
- Implementation of the Observation Management Portal using web technologies.
- Optimisation and Refactoring of our code

We are very excited and looking forward for the upcoming months. We are hoping to get the first fringes by the end of 2011 and shortly after the first images from BETA.

REFERENCES

[1] ASKAP website `http://www.atnf.csiro.au/projects/askap`

[2] J.C. Guzman, ”Preliminary Design of The Australian SKA Pathfinder (ASKAP) Telescope Control System”, ICALEPCS’09, Kobe, October 2009

[3] ASYN EPICS driver framework website `http://www.aps.anl.gov/epics/modules/soft/asyn`

[4] PyEpics3 Python CA Library website `http://cars9.uchicago.edu/software/python/pyepics3`

[5] Control System Studio (CSS) website `http://cs-studio.sourceforge.net`

[6] Redmine Project Management Tool website: `http://www.redmine.org`

[7] Hudson Continuous Integration Tool website: `http://hudson-ci.org`