CONvergent
Radio
Astronomy
Demonstrator

CONRAD

CONRAD Architecture

CONRAD-SW-0011

Issue 1.0

2007/06/02

Keywords: architecture requirements CN1_Architecture CONRAD computing software

Prepared By:

Name	Signature	Date
Tim Cornwell Juan Carlos Guzman Jasper Horrell Yuantu Huang Malte Marquarding Simon Ratcliffe Ger van Diepen Maxim Voronkov		

Approved By:

Name	Signature	Date

Document History

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
0.2	2007/05/19	As listed	
	Initial version for internal review		
0.3	2007/05/28	Tim Cornwell	All
	All sections edited as per reviewers comments.		
0.4	2007/05/28	Juan C Guzman	Section 9
	Figure 11 and Section 9.2 updated. Front cover updated to latest template layout.		
0.5	2007/05/29	Tim Cornwell	Sections 1, 6
	New figure, some wording changes per reviewers comments		
1.0	2007/06/02	Jasper Horrell	All
	Revamped Analysis of Requirements section. Elsewhere minor text and figure typo scale fixes for issue-1, updated list of acronyms.		

Table of Contents

1 Introduction ... 5
1.1 Summary ... 5
1.2 Scope ... 5
1.3 References ... 5
1.4 Glossary ... 5
2 Analysis of requirements ... 7
3 Project philosophy ... 8
4 Telescope Management ... 10
4.1 CTOS Overview ... 10
4.2 CTOS Executive and Components ... 12
4.3 CTOS Monitoring and Control ... 13
4.4 Fault Handling and Logging ... 15
5 Data flow in a CONRAD telescope ... 17
6 The CONRAD Central Processor ... 20
6.1 Processing tasks ... 20
6.1.1 Data Sets ... 21
6.2 Visibility data distribution and processing ... 21
6.2.1 Calibration and imaging ... 21
6.2.2 Distribution of the data ... 22
6.2.3 Master/worker control ... 23
6.2.4 Interprocess communication ... 24
6.3 Visibility Data Access ... 24
6.4 Robustness ... 25
6.5 Interface to Monitoring and Control ... 25
7 The CONRAD Data Store ... 26
7.1 Internal use of the CDS ... 26
7.2 CONRAD Archive ... 26
8 User Interfaces ... 28
8.1 Web Interface ... 28
8.2 Python Interface ... 29
9 Deployment ... 32
9.1 Antenna Control ... 33
9.2 Time Server ... 33
9.3 Site Monitoring Station ... 33
9.4 Beamformers ... 33
9.5 Correlator ... 33
9.6 CONRAD Data Conditioner (CDC) and Router ... 33
9.7 Central Processor (CP) ... 33
9.8 CONRAD Data Store (CDS) ... 34
9.9 CONRAD Telescope Operating System (CTOS) ... 34
10 CONRAD software development processes ... 36
10.1 Structure ... 36
10.1.1 Code ... 36
10.1.2 3rdParty ... 37
10.1.3 Tools ... 37
10.2 Dependency Rules ... 37
10.3 Software Development Life Cycle ... 37

1 Introduction

1.1 Summary

The Convergent Radio Astronomy Demonstrator (CONRAD) is a computing collaboration between the meerKAT and MIRANdA telescope computing teams which aims to produce the common software required to operate and process/reduce the data from the two telescopes. The collaboration also includes some participation from LOFAR, limited in scope to the data pipeline area. The top level architecture of CONRAD is described.

1.2 Scope

The top level software architecture of CONRAD is described. This architecture has been developed to meet the CONRAD functional requirements [1]. The detailed design and implementation of CONRAD subsystems are not described in this document.

The CONRAD functional requirements document [1] and this architecture document lay out the MIRANdA and meerKAT computing teams' joint view of computing architectural requirements and implications at the time of writing and indicate the direction of the current technical development. We expect several iterations of these documents will be required in consultation with the wider projects before signature. Due to the pervasive nature of the telescope computing systems, we expect these documents to assist in firming up the project-level thinking around telescope specifications and design.

1.3 References

1. CONRAD Functional Requirements, CONRAD-SW-0010, version 1.0, 2007, J. Horrell, M. Voronkov, et al.
2. Measurement Equation design and implementation, CONRAD-SW-0003, version 0.1, 2007, T.J. Cornwell
3. Duchamp source finder. http://www.atnf.csiro.au/people/Matthew.Whiting/Duchamp/
4. Master/worker design, CONRAD-SW-0012, version 0.1, 2007, G. van Diepen
5. CONRAD Software Standards, CONRAD-SW-0007, version 0.1, 2007, R. Crida and M. Marquarding
6. CONRAD Telescope Operating System, CONRAD-SW-0009, version 0.1, 2007, S. Ratcliffe
7. AIPS++ Measurement Set, AIPS++ memo 229, http://aips2.nrao.edu

1.4 Glossary

ACRONYM	**Definition**
AgentX	Standard for extending SNMP agent capabilities
ACSM	Antenna Control Software Module
CDC	CONRAD Data Conditioner
CDS	CONRAD Data Store
CLI	Command Line Interface
CONRAD	Convergent Radio Astronomy Demonstrator
CP	Central Processor
CTOS	CONRAD Telescope Operating System
LDAP	Lightweight Directory Access Protocol
FPGA	Field Programmable Gate Array

GPIB	General Purpose Interface Bus
GPS	Global Positioning System
GPU	Graphics Processor Unit
I/O	Input / Output
M&C	Monitoring and Control
MIB	Monitor Information Base – a database of objects that can be monitored by SNMP
NTP	Network Time Protocol
OID	Object Identifier
PAF	Phased Array Feed
PED	Phased Experimental Demonstrator
RDBMS	Relational Data Base Management System (e.g. MySQL)
RFI	Radio Frequency Interference
RRDTool	Round Robin Database Tool
SNMP	Simple Network Management Protocol
SDLC	Software Development Life Cycle
TPM	Telescope Policy Manager
UI	User Interface
VLAN	Virtual Local Area Network
XDM	Experimental Demonstrator Model

2 Analysis of requirements

The functional requirements for MIRANdA and meerKAT [1] are still evolving. In particular, the overall scope of meerKAT is not yet fully defined and so any definitive statement as to the requirements is premature. However, we can analyze the current requirements, and adjust in future as new requirements are added. This practice gives us the ability to provide some costing feedback on new requirements before they become firm commitments.

Although meerKAT and MIRANdA will differ in certain key aspects, the following aspects are common and form the technical basis for the collaboration:

- Large number of antennas (30 – 100+) forming a radio synthesis array with max baseline < 10km,
- Large data volumes (up to Terabytes per hour) which will require distributed processing,
- Operation at centimetre wavelengths,
- Capable of continuum imaging to the confusion limit,
- Capable of integrating spectral line images for ~ 1 year (full sky) or ~ 3 months (single pointing), with large numbers of channels,
- Capable of polarimetric observations,
- Presenting a summed array beam for VLBI or pulsar observing,
- Situated in an isolated location, operated remotely, without a dedicated maintenance staff on-site,
- Fully automated pipelined processing for many use cases (incl. RFI mitigation),
- Small dishes as the concentrators (< 15m diameter),

In the MIRANdA case, a wide field of view is required (up to tens of square degrees). The field of view requirement for meerKAT may not be as large and the emphasis is likely to be on better sensitivity coupled with reasonable field of view.

In the use cases where fully automated pipelined processing is necessary, the primary interaction between astronomers and the CONRAD telescope will be via the CONRAD Archive.

The principal origin of many challenges lies in the combination of wide field of view, large number of antennas, and many spectral channels. The wide field of view, however obtained, brings complications –the wide-field component (phased array feed or horn array) will require special calibration, and the imaging algorithms may require innovations.

The large data rate demands real-time processing which implies many other features. For example, rather than write the visibility data to a self-contained export format, such as the AIPS++ MeasurementSet, the pipeline processing algorithms can query the state of the telescope directly from the relevant database. Thus the telescope should be seen as one complete and integrated system, including the pipeline.

The telescopes will both be deployed at remote sites for reasons of RFI mitigation. This deployment will command a high cost premium and an active RFI mitigation system which forms part of the pipeline processing is required to maximize this investment.

3 Project philosophy

We plan highly focused software

Our delivery timescales are short by any standards and budgets are limited. As a consequence, we need to move quickly and lightly and have encouraged our stakeholders to focus on key requirements. We aim for quality software that is simple in structure without being too limiting. We expect to spend only about 50 – 60 FTE-years on CONRAD software in order to support the basic modes. This is 5 – 10 % of the cost of the two telescopes – MIRANdA and meerKAT.

SKA relevance

Both meerKAT and MIRANdA are SKA pathfinder instruments and, while CONRAD architecture has been chosen with SKA scalability in mind, this is not a driving requirement. Rather, CONRAD builds experience in software and techniques that are relevant to SKA and will meet the requirements of the pathfinders, but is not intended to necessarily grow into the actual SKA software.

Collaboration and joint management

The collaboration and joint management approach seeks to demonstrate a viable model for SKA software development with distributed teams.

Novel requirements and novel approach

Our timescales, budget and certain novel telescope requirements drive novel architecture thinking. In particular, the telescopes need to support a model of simultaneous observations and processing, remote operations with difficult logistics, dealing effectively with large volumes of data, a large number of antennas and wide field-of-view imaging.

We try to use existing third party software as much as practical

We do not attempt to create everything from scratch, but rather make intelligent re-use and extension of widely deployed third party software, where suitable. In particular, we have taken note of software systems widely deployed in commercial environments with large numbers of networked components.

We allow partners to use commercial software as required but avoid lock-in.

Although commercial software may be suitable to use in parts of the system, the guiding principle for CONRAD in this respect is that the CONRAD software should not be tied to any specific vendor. This is to avoid long-term licensing issues and enable wider adoption of the code. For this reason, most of the third party software currently envisaged to be used is open source / open licence. For example, if an Oracle database is used for reasons of performance or scalability, it should still be possible to move the CONRAD software to a different database (e.g. PostgreSQL) without major effort.

Deploy early and often

We deploy working versions early and often, to catch integration problems as soon as possible. Our projects, MIRANdA and meerKAT provide numerous opportunities for this type of testing.

Don't over-design, refactor

Refactoring is preferred to initial (over)-design, since it reflects the reality of evolving understanding of the problem domain, as well as the possibility of changing requirements.

Design to interfaces

Designing to interfaces promotes decoupling of components and thus increase maintainability and facilitates refactoring.

We try to avoid hardware interface lock-in

Where possible the CONRAD computing systems should avoid lock-in to proprietary hardware interfaces (*e.g.* GPIB). This is for the same reasons as discussed for commercial software lock-in. Note that for reasons of gearing and long term support, the CONRAD computing systems seek to take advantage of widely adopted open standards for hardware and software.

Proposed changes in requirements will require analysis and costing.

Proposed changes in the computing requirements will not automatically be accepted and may require extensive analysis and costing prior to some decision around acceptance.

Substantially changing requirements will require revision of plans.

Requirements that change substantially may affect the balance of the collaboration and may also require a change in timescales or budget.

4 Telescope Management

Telescope management, monitoring and high level control is the responsibility of CTOS (CONRAD Telescope Operating System). This set of packages has been designed with several high level principles in mind. These are as follows:

- A top-level scheduler is responsible for overall telescope control.
- Control is performed at a high level with the subsystems of the facility taking responsibility for fine-grained control themselves.
- Monitoring is centralised, with each subsystem providing deep access to their internal monitoring points.
- Observing and processing of data will occur simultaneously.
- Fault management and escalation is centralized. However, failsafe modes and critical faults are handled by the subsystems themselves.
- Fastest loop handled by top-level control is of order 1s.
- A watchdog ensures availability of critical CTOS resources.
- Scientists will interact with facility data through an archive.
- It is essential that the system is always in a well determined state, implying that control must reside in as few places as possible.

4.1 CTOS Overview

The operation of CTOS is managed through one of two user interfaces, either web- or python based. Through these, the user can interact with the scheduler to create scheduled system tasks such as an observing run or scheduled maintenance. In addition, the user may, through these interfaces, also perform single-ended, non-scheduled tasks such as may be required for testing or debugging purposes.

The interaction with the scheduler produces schedule files that are passed to the schedule runner, which is ultimately responsible for ensuring proper execution of the specified tasks (monitor, control etc…). The two other points of control in the system are chiefly concerned with correct operation:

- The escalation manager is responsible for handling errors that occur during telescope operation. It will make decisions based on the severity of the fault, and other information, and enact these through interaction with the scheduler (in a severe case) or possibly just the relevant subsystem directly.
- A watchdog is in place to ensure critical system services are available at all times.

Apart from the control related tasks, a range of other components are present that are involved in ancillary functions such as monitoring, configuration management, user interface delivery and other system tasks.

The basic building blocks of CTOS and their relationship to each other is shown in the diagram below:

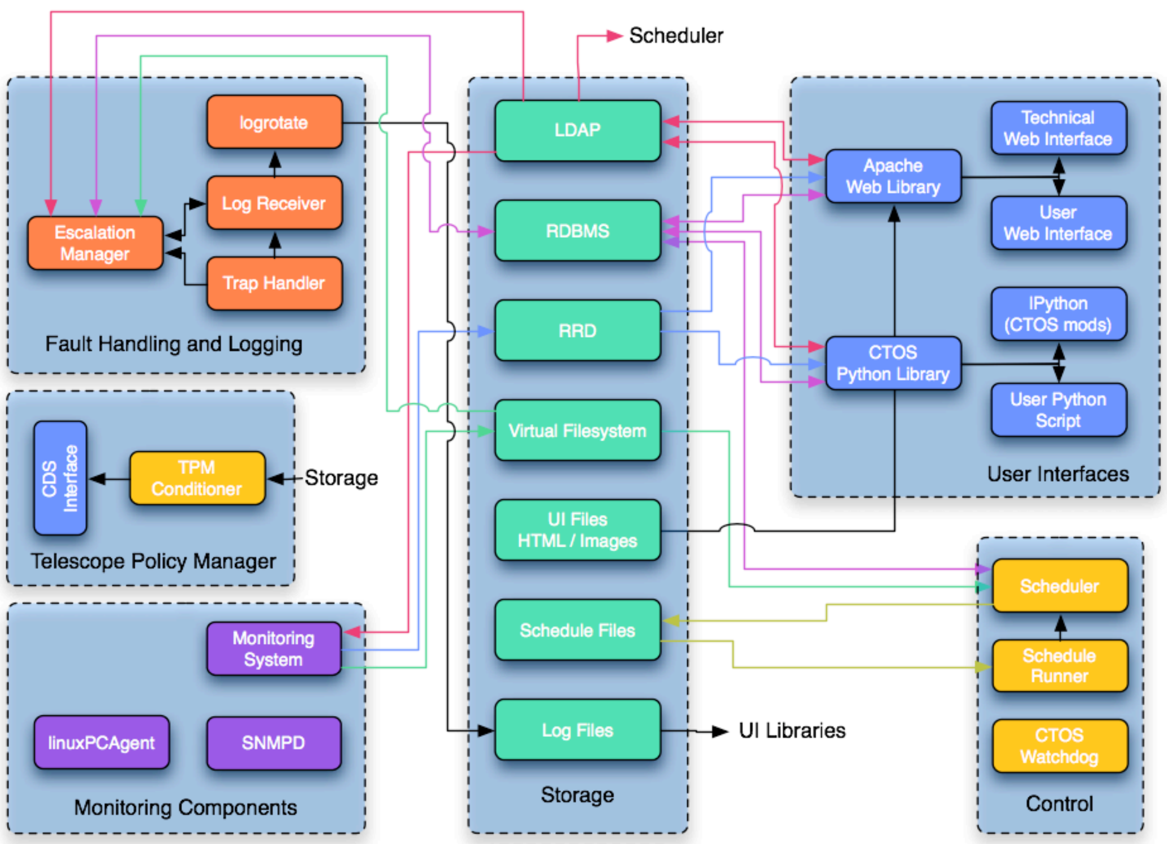


Figure 1 Basic building blocks of CTOS

To help clarify the diagram above the connections between each storage form and the relevant components is detailed in the table below:

Table 1 Connections between storage and components

	Storage Mechanism	Provides data to	Receives data from
LDAP	Disk – bdb backend	Monitoring system Escalation Manager Scheduler UI Libraries	UI Libraries
RDBMS	Disk – bdb backend	Escalation Manager Scheduler UI Libraries	Escalation Manager Scheduler UI Libraries
Round Robin Database	Disk - RRD	UI Libraries	Monitoring System
Virtual Filesystem	Memory based	Monitoring System	Scheduler Escalation Manager
UI Files (HTML/Images)	Disk – text/binary files	Tech / User Interfaces	N/A
Schedule Files	Disk – text/xml files	Schedule Runner	Scheduler
Log Files	Disk – text files	UI Libraries Escalation Manager (direct from log receiver)	Log Receiver (via logrotate)

A brief description of the components is as follows:

- Scheduler – responsible for coordinating user requests and producing appropriate schedule files for use by the schedule runner.
- Schedule runner – responsible for overall control of user tasks. All scientific control is performed by this task.
- Escalation manager – any errors or out of bound events are handled here. Ensures telescope remains operational in the event of faults.
- CTOS Watchdog – responsible for ensuring availability of core system components. More detail in the next section.
- LDAP – directory that stores the static physical configuration of the facility.
- RDBMS – database used by portions of the web interface for user data capture and on the fly configuration.
- RRD – a round robin database engine that is responsible for storing and ageing the monitoring data captured by the monitoring system.
- Telescope Policy Manager – collects data from various aspects of the system via the appropriate storage mechanism and provides conditioned meta-data to the CONRAD Data Store (CDS).
- Monitoring system – responsible for gathering deep, system wide monitoring information and storing in appropriate repository. Handles conversion and watch expressions.
- Virtual filesystem – this is a unix mountable virtual filesystem that exposes the relatively static LDAP hierarchy and associated dynamic values in a traditional filesystem manner.
- Log receiver – central logger receives logging information from all running user tasks and stores this in appropriate log files. Forwards errors on to the escalation manager.
- Trap handler – receives SNMP trap messages, formats these, and then forwards them to the escalation manager. Each trap is also logged with the logging system.
- linuxPCAgentX – Provides a wide range of monitoring points on standard linux PC's. Also contains control functionality for launching processes. Used by the watchdog and the monitoring system.
- SNMPD – System daemon that handles forwarding of SNMP set- and get requests to appropriate AgentX plugin.
- Apache web library – webserver used to provide access to the system web interfaces along with appropriate libraries for providing core backend services to the technical and user interfaces.
- CTOS python library – used for python based control of the telescope. Exposes the monitoring and control hierarchy directly to python and allows easy use of system level commands such as LDAP manipulation.
- Logrotate – responsible for ageing and compressing system log files.

4.2 CTOS Executive and Components

The following diagram shows the main CTOS system processes and their division between system and user space:

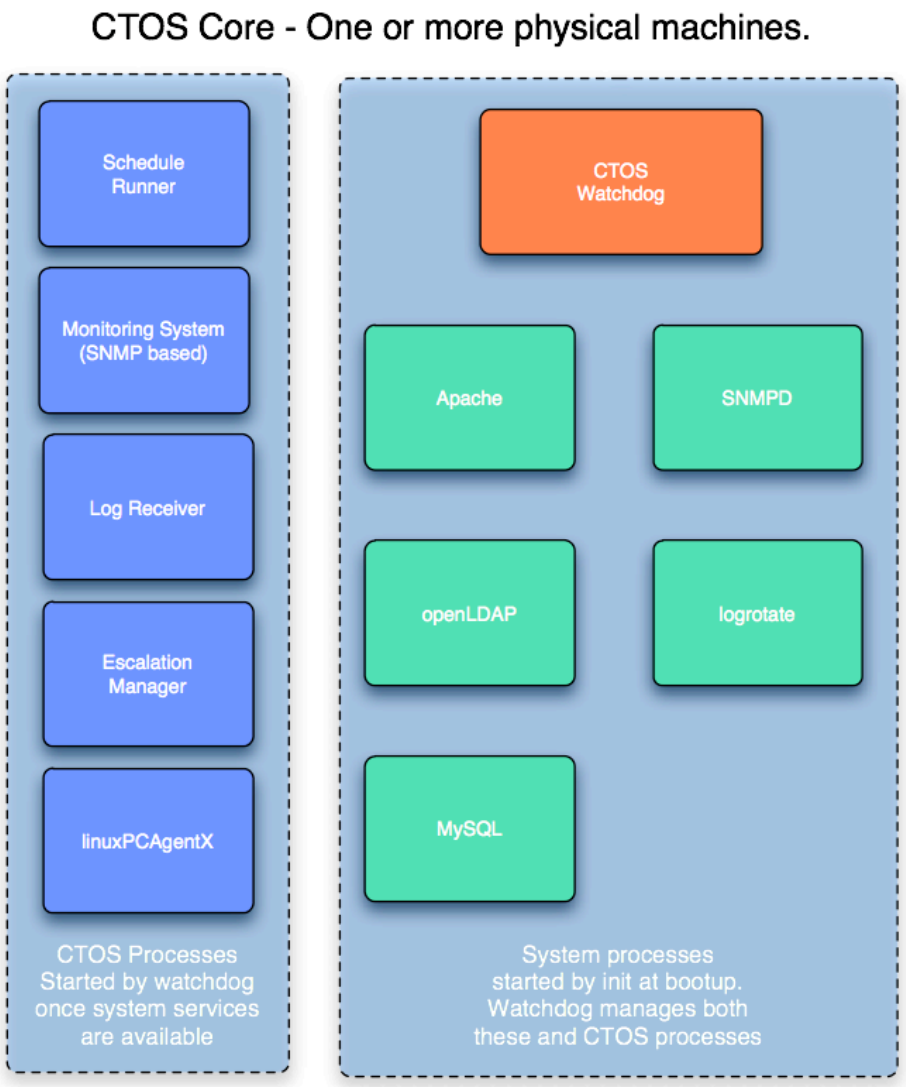

CTOS Core - One or more physical machines.
Schedule Runner
Monitoring System (SNMP based)
Log Receiver
Escalation Manager
linuxPCAgentX
CTOS Processes Started by watchdog once system services are available
CTOS Watchdog
Apache
SNMPD
openLDAP
logrotate
MySQL
System processes started by init at bootup. Watchdog manages both these and CTOS processes

Figure 2 CTOS processes

The system processes listed are started at boot time through the use of standard init scripts. This includes CTOS watchdog. Once the watchdog is satisfied that all of the dependant system processes are available it will launch CTOS processes itself.

From this point on the watchdog takes responsibility for ensuring that all of the system and CTOS processes stay running. Should a process terminate unexpectedly, the watchdog will re-spawn that process and log accordingly.

It is important to note that most of the processes shown do not have to reside on the same physical machine. The exceptions are that the Log Receiver and logrotate should be collocated, and that SNMPD and linuxPCAgentX must be available in all machines that form part of CTOS core. The single watchdog task will manage all of these distributed services.

4.3 CTOS Monitoring and Control

The diagram below shows a typical scenario involving monitor and control operations:

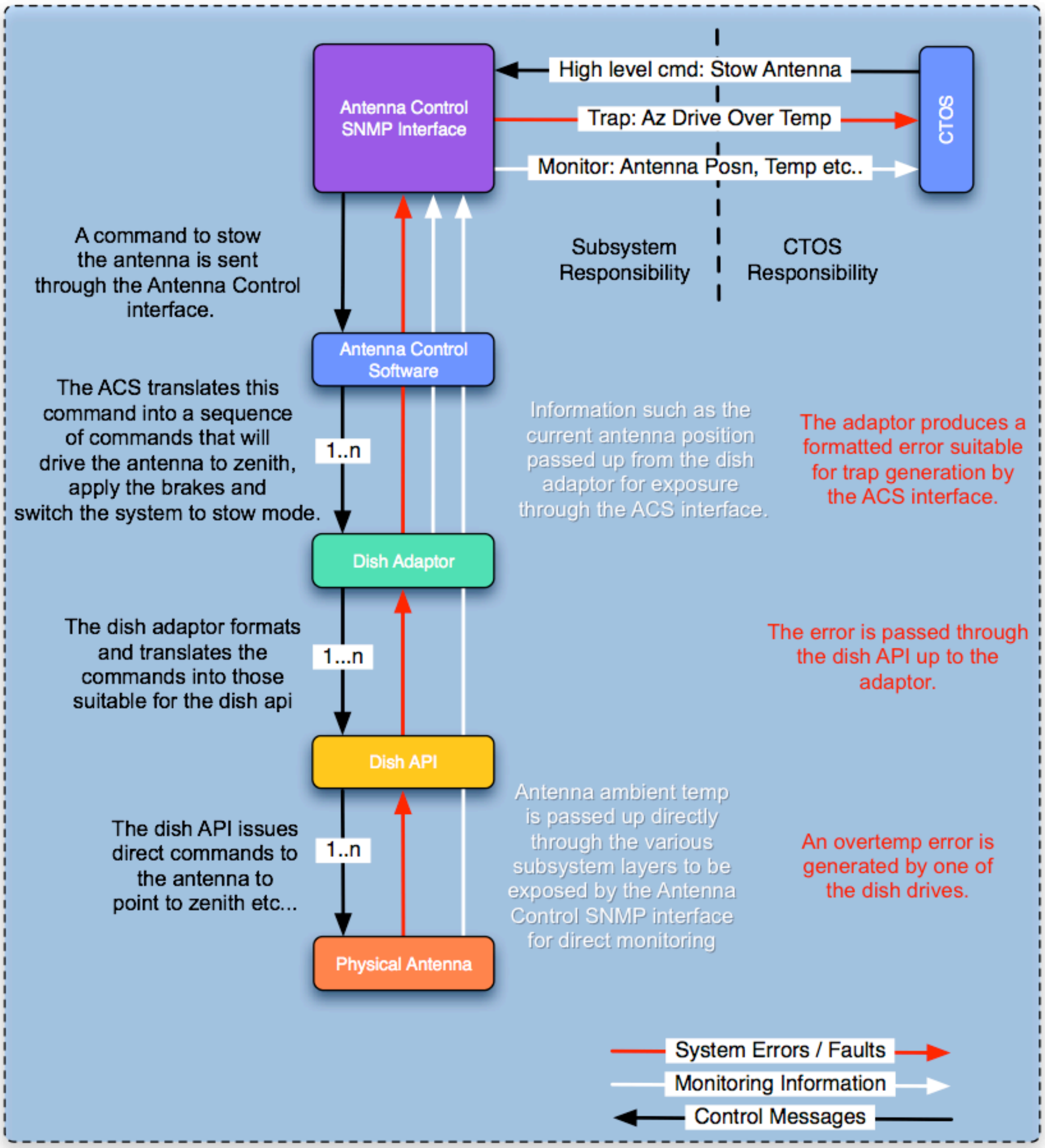

Antenna Control SNMP Interface
High level cmd: Stow Antenna
Trap: Az Drive Over Temp
Monitor: Antenna Posn, Temp etc..
CTOS
A command to stow the antenna is sent through the Antenna Control interface.
Subsystem Responsibility
CTOS Responsibility
Antenna Control Software
The ACS translates this command into a sequence of commands that will drive the antenna to zenith, apply the brakes and switch the system to stow mode.
1..n
Information such as the current antenna position passed up from the dish adaptor for exposure through the ACS interface.
The adaptor produces a formatted error suitable for trap generation by the ACS interface.
Dish Adaptor
The dish adaptor formats and translates the commands into those suitable for the dish api
1...n
The error is passed through the dish API up to the adaptor.
Dish API
The dish API issues direct commands to the antenna to point to zenith etc...
1..n
Antenna ambient temp is passed up directly through the various subsystem layers to be exposed by the Antenna Control SNMP interface for direct monitoring
An overtemp error is generated by one of the dish drives.
Physical Antenna
System Errors / Faults
Monitoring Information
Control Messages

Figure 3 Example of control and monitor flow

This shows that monitoring of a subsystem extends right down to the deepest levels, and although the interface to the data is somewhat abstracted, the information presented is still raw and not aggregated or modified by the subsystem itself.

The chain in white shows how monitoring information (in this case the ambient temperature reading for a particular antenna and the current position) is passed up from the relevant layers and made available to CTOS.

Errors that are not handled locally (more information on this case in the next section) may have to go through some conversions in order to produce a suitably formatted trap event that is sent directly to the trap handler.

The control chain in black shows the concept of high-level control. CTOS issues a command such as stow antenna. This may come from a variety of sources within CTOS: the scheduler (say for scheduled maintenance); the user may generate this command from the user interface; or the escalation manager might stow the dish based on a fault received. Once the antenna control subsystem has received this command it enacts lower level control to realise this command. Thus a single command may become a sequence of commands that control the drives of individual axes and ensure the brakes are applied once the stow position has been reached. This is all hidden from the control system and no tight external timing loops are required.

4.4 Fault Handling and Logging

The diagram below shows how the fault handling system and the centralised logger work:

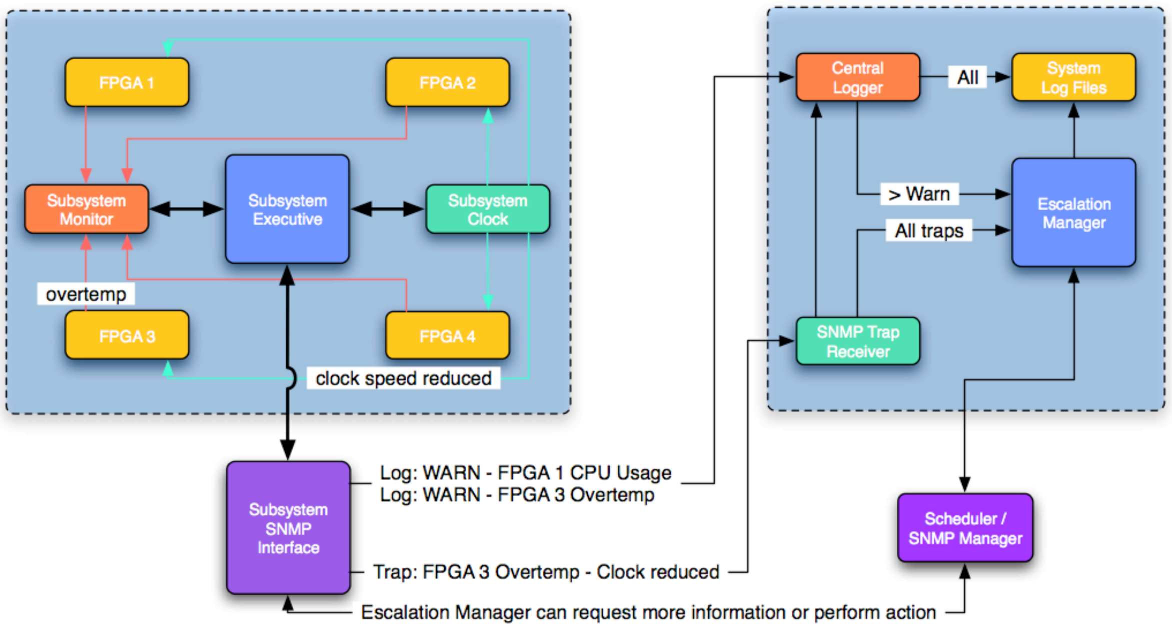


Figure 4 Example of fault handling and logging

This diagram shows the case of a subsystem that contains a number of FPGAs, each with its own embedded processing core. In the event of say a fan failure on FPGA 3 we do not want to rely on CTOS to have to handle the problem.

Instead the subsystem itself is responsible for these critical failures that can lead to hardware failure. In this example the subsystem executive will reduce the clock speed on the device in question in order to run cooler.

Even though the event is handled locally, a trap is still sent to the escalation manager, so that it can verify that the action taken by the subsystem is appropriate, and it can make decisions as to how this change will effect the operation of the system as a whole.

All other faults and problems (i.e. non critical) are forwarded to the escalation manager for handling as part of overall system control.

Problems that occur that are at a lower level (a warning) can be sent exclusively through the logging system. The example in the diagram shows a log warning of excessive use of the CPU on FPGA 1. The escalation manager will use this log information in conjunction with other monitoring information it may request from the subsystem to make a decision. This decision may involve a control command to the subsystem itself or may warrant more drastic action such as rescheduling the current run.

All traps sent by the system are also sent through the central logger in order to ensure dual data paths for critical information.

Each subsystem must provide a self-test function to CTOS via an SNMP interface. Upon reception of the self-test function, each subsystem should perform a self-test of its functionality as well as execute the self-test function on each hardware device that belongs to that subsystem. Therefore, it is required that each hardware device attached to a subsystem also provides a self-test function interface.

5 Data flow in a CONRAD telescope

The CONRAD telescopes will have a number of operational modes. One of the most demanding for data flow will be spectral line surveying of the entire sky. This is a high priority for MIRANdA.
The data flow in this mode for the 2km scale version of MIRANdA is shown in Figure 5. The data flow for meerKAT is of similar scale. The data flow scales as the baseline length and the number of spectral channels. For the 8km MIRANdA, the data flow from the correlator onwards will be about 4 times larger. Hence even after the correlator, we must be able to move about 1 – 5GB/s into the pipeline, through the processing steps, and eventually out to the CONRAD Data Store. Estimates of the processing load for MIRANdA lie in the rage 1-10 Tflops.

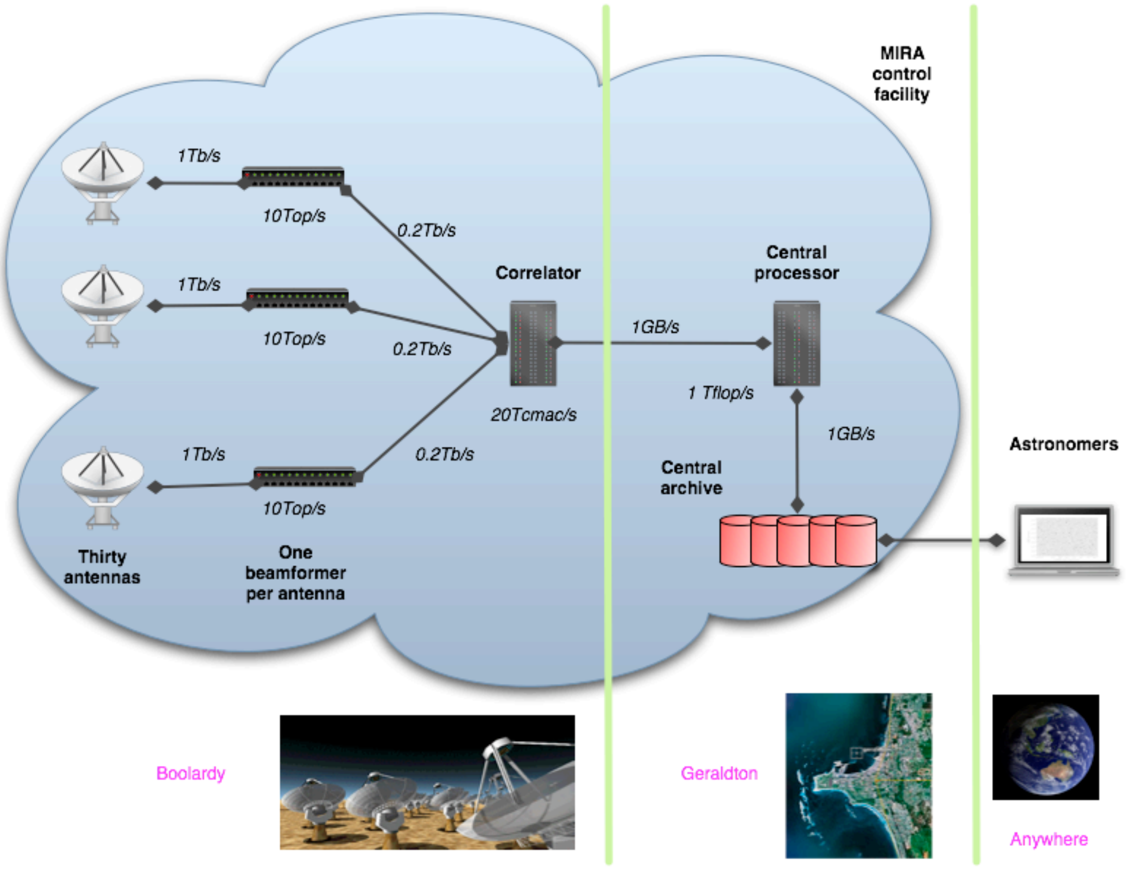


Figure 5 Overall data flow for MIRANdA in spectral line mode

The flow of data and control around the CONRAD subsystems is as shown in Figure 6. CTOS is responsible for controlling and monitoring all subsystems (red lines). The voltage data (purple lines) stream from the array beam former(s) into as yet unspecified devices, which are out of scope for CONRAD. The visibility data (green lines), which form the bulk of the data to be processed, stream from the correlator, through a data conditioning subsystem, and through a router to the central processor and the CONRAD Data Store.

A fundamental assumption is that the telescope is calibrated in real-time – essentially on a 20 minute timescale, or faster. A global model of the sky is known with sufficient accuracy to allow single pass estimation of calibration parameters, such as antenna gains, delays. The calibration parameters are applied to the data from which they were derived, and also fed back to the telescope for application to subsequent observations. Calibration parameters are applied as far upstream in the data flow as possible.

The calibrated data are processed into images as soon as possible. The images are then analyzed by extracting catalogs and then sent to the archive. Conversion to images may require all the data to be present (*e.g.* to facilitate deconvolution), in which case the processing cannot proceed until all the observations have concluded. In this situation, the pipeline must contain sufficient storage to hold at least two complete observations.

The visibility data must be stored close to the processing (this point is elaborated in the next section). This, combined with the fact that ~ 100+ machines may be required for the processing, means that less than 1% of the data will be located on any given computational node. A typical spectral line observation for MIRANdA would therefore deposit about 200-500GB of data per computational node. This is a large amount of data to read and write within 12 hours – possible but pushing the limits of non-RAID technology.

Meta data is likely to be much smaller in volume. It flows on a limited number of paths:

- CTOS controls and monitors all subsystems and writes the results into a monitor data base with automatic ageing.
- The Telescope Policy Manager conditions the monitor data and sends processed information to the CONRAD Data Store (CDS).
- The Central Processor queries the CDS for meta-data as needed for processing.

Hence the CDS does not communicate meta-data directly to any subsystem, apart from the central processor. The only input of meta-data to the CDS is via the Telescope Policy Manager (TPM). The job of the TPM is to implement a single centralized model for the telescope. The model filters and conditions the vast amount of monitoring data into a smaller set of meta-data that will be used by the Central Processor pipeline logic.

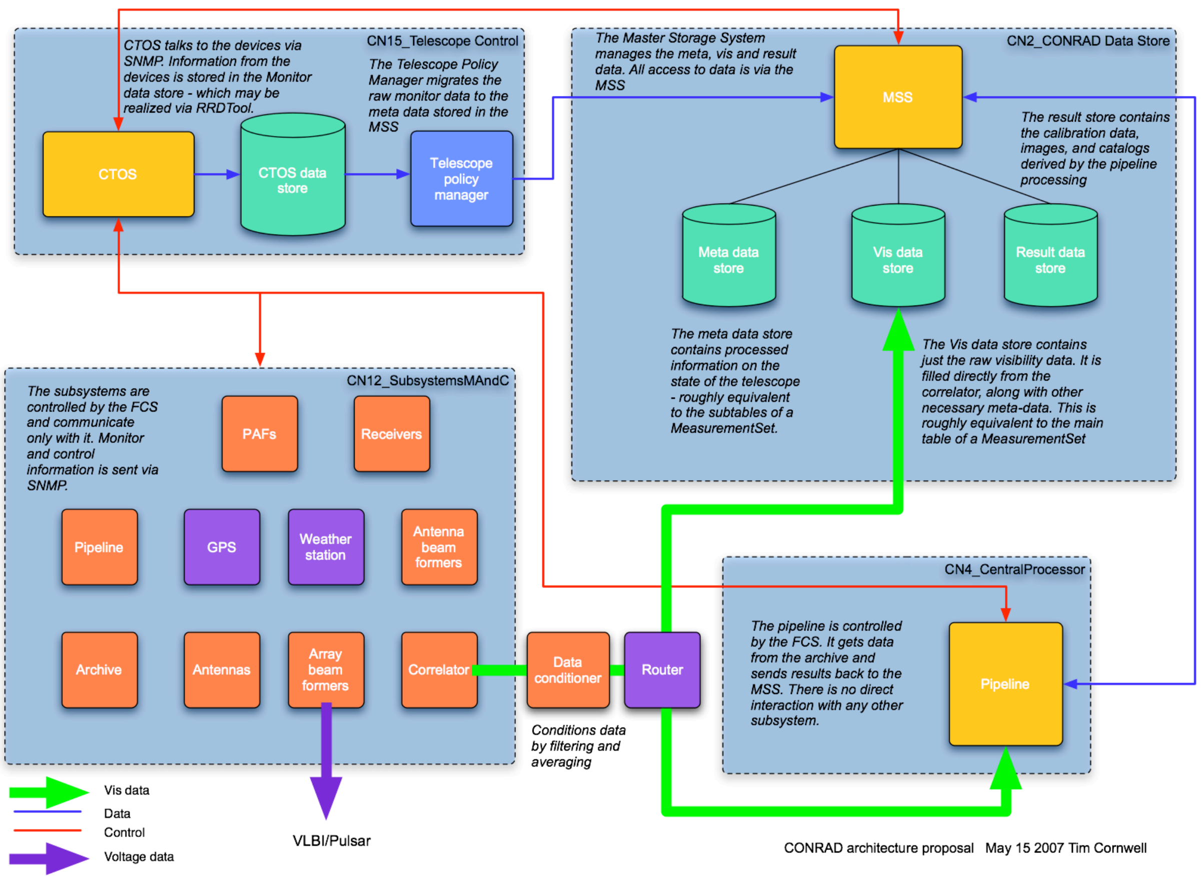

CN15_Telescope Control
CTOS talks to the devices via SNMP. Information from the devices is stored in the Monitor data store - which may be realized via RRDTool.
The Telescope Policy Manager migrates the raw monitor data to the meta data stored in the MSS
CTOS
CTOS data store
Telescope policy manager
CN2_CONRAD Data Store
The Master Storage System manages the meta, vis and result data. All access to data is via the MSS
MSS
The result store contains the calibration data, images, and catalogs derived by the pipeline processing
Meta data store
Vis data store
Result data store
The meta data store contains processed information on the state of the telescope - roughly equivalent to the subtables of a MeasurementSet.
The Vis data store contains just the raw visibility data. It is filled directly from the correlator, along with other necessary meta-data. This is roughly equivalent to the main table of a MeasurementSet
CN12_SubsystemsMAndC
The subsystems are controlled by the FCS and communicate only with it. Monitor and control information is sent via SNMP.
PAFs
Receivers
Pipeline
GPS
Weather station
Antenna beam formers
Archive
Antennas
Array beam formers
Correlator
Data conditioner
Router
Conditions data by filtering and averaging
CN4_CentralProcessor
The pipeline is controlled by the FCS. It gets data from the archive and sends results back to the MSS. There is no direct interaction with any other subsystem.
Pipeline
VLBI/Pulsar
Vis data
Data
Control
Voltage data
CONRAD architecture proposal May 15 2007 Tim Cornwell

Figure 6 Schematic of data flow

A number of processes occur within this data flow. These are summarized, along with the timescales on which they operate on data, in Table 2.

Table 2 Timescales for important processes in the data flow

Timescale	Subsystems	Operations
~1 ns to ~ 100 ms	Antenna beamformers, correlator	Beam-forming, spectral analysis, correlation, integration
~ 100 ms to ~10 s	Data conditioner	RFI excision, time and frequency integration, data re-ordering, fast transient detection
~10 s to ~ 20 min	Calibration pipeline	Calculation and application of calibration parameters, and feedback of parameters to the control system
~ 10 s to ~ 12 hours	Transient pipeline	Detection and cataloging of transient sources
~ 20 min to ~ 12 hours	Imaging Pipeline	Integration of spectral line cubes, updating of continuum images
~ 12 hours	Cataloging pipeline	Extraction of source properties into catalogs
~ 12 hours to 1 year	CONRAD Data Store	Accumulation of survey
1 year +	CONRAD Data Store	Accumulation of multiple projects

6 The CONRAD Central Processor

In existing radio telescopes, astronomical data are usually stored in a single archive, and processed on the desktop of the astronomer. The amount of data coming from new telescopes like MIRANdA, meerKAT, and LOFAR is so large (terabytes per hour) that the data from a single observation have to be stored and processed in a distributed way which can only be done on a large dedicated machine. The amount of time to process the data is very limited. The processing has to be finished before the next observation is done. It is important to note that by increasing the computational resources available, it is possible to decrease the data processing time.

Two principal types of data have to be processed:
- Continuum data. For MIRANdA this will typically consist of 64 channels resulting in a visibility data set of approximately 128 GBytes.
- Line data. For MIRANdA this will typically consist of 16384 channels resulting in a visibility data set of approximately 34 TBytes.

The amount of data is so large, that care has to be taken that data are accessed and shipped around as little as possible. Therefore the main motto of the Central Processor subsystem is: ***"Bring the processes to the data and not the data to the processes"***.

6.1 Processing tasks

Various tasks will be performed on the visibility data.
- Automated flagging to get rid of bad data points. The flagging can be based on prior knowledge of RFI or on data statistics (such as deviation from the median). The RFI data base must be updated if RFI is found.
- Calibration of the data by comparing predicted visibilities from a model with the measured visibilities. This involves solving non-linear systems for the given model parameters. Most calibration needs to be done on the continuum data. During calibration both instrumental (e.g. antenna phases and amplitudes) and image plane parameters (e.g. pointing errors) can be solved. The initial sky model for the observed field is derived from the global sky model. Parameters solved in a calibration run can be used in another observation.
- Imaging and deconvolving the data. Image plane parameters (*e.g.* beam shape) can be applied.
- Source finding in the image cube and updating the global sky model. Feature finding in an HI line cube (*e.g.* using Duchamp [3]).
- Sources can be subtracted from the visibility data, so weaker sources can be found.
- Searching for transient sources by imaging each integration and looking for changes.

Besides source and feature extraction, image analysis and display may also be needed. Existing tools can be used for this, but are unlikely to be able to process large image cubes efficiently. It may prove necessary to add distributed processing capabilities to such tools.

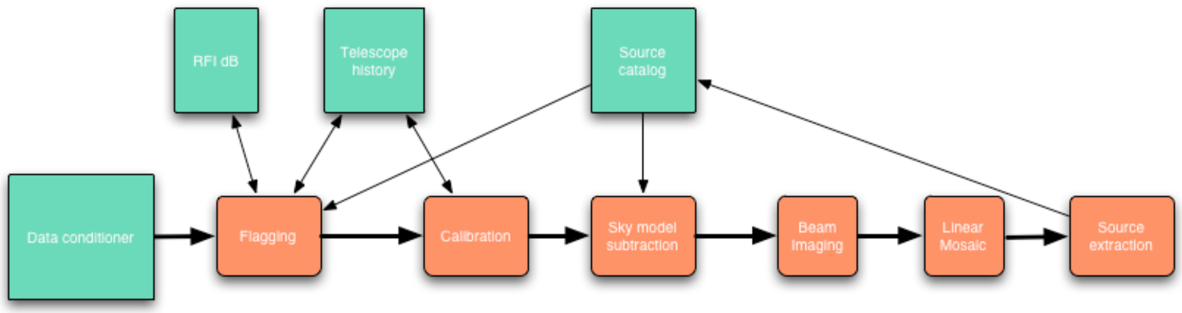


Figure 7 Sequence of operations in a typical pipeline to construct a source catalog

The bulk of the overall processing load lies in the convolutional resampling step necessary for imaging [5].

6.1.1 Data Sets

Several data sets are needed during the processing of the data.

1. Any visibility data format can be used by implementing suitable interface (DataAccessor) classes filling as needed from visibility data (directly from the correlator and conditioner) and meta-data from the CDS.
2. The image data are stored as AIPS++ images.
3. The calibration parameters must be held in a database.
4. The global sky model contains all known sources with their parameters. It can consist of a number of catalogs. The initial model of the observed field is derived from it and stored in a table. New sources are added to the table and will eventually be added to the global sky model.
5. HI source parameters are stored in an HI source database.
6. Newly found RFI is added to the RFI catalog during data flagging.

6.2 Visibility data distribution and processing

6.2.1 Calibration and imaging

We consider calibration and imaging as non-linear least squares problems: we observe some visibility data to which we wish to fit models, incorporating both calibration (such as antenna gains) and image terms (such as sky brightness). The fitting process is intrinsically non-linear but we decompose it into iteration of two parts:

- Prediffer – finds gradients of the visibility function with respect to the unknown parameters, as well as the overall residual visibility. These are then converted into normal equations.
- Solver – find solutions for the unknown parameters from the normal equations.

This split and terminology has been used at LOFAR. It is very well suited to distributed processing - the prediffer step may be distributed over partitions of the data each residing on a different processor, but the solution is most often global and must be calculated on a single processor.

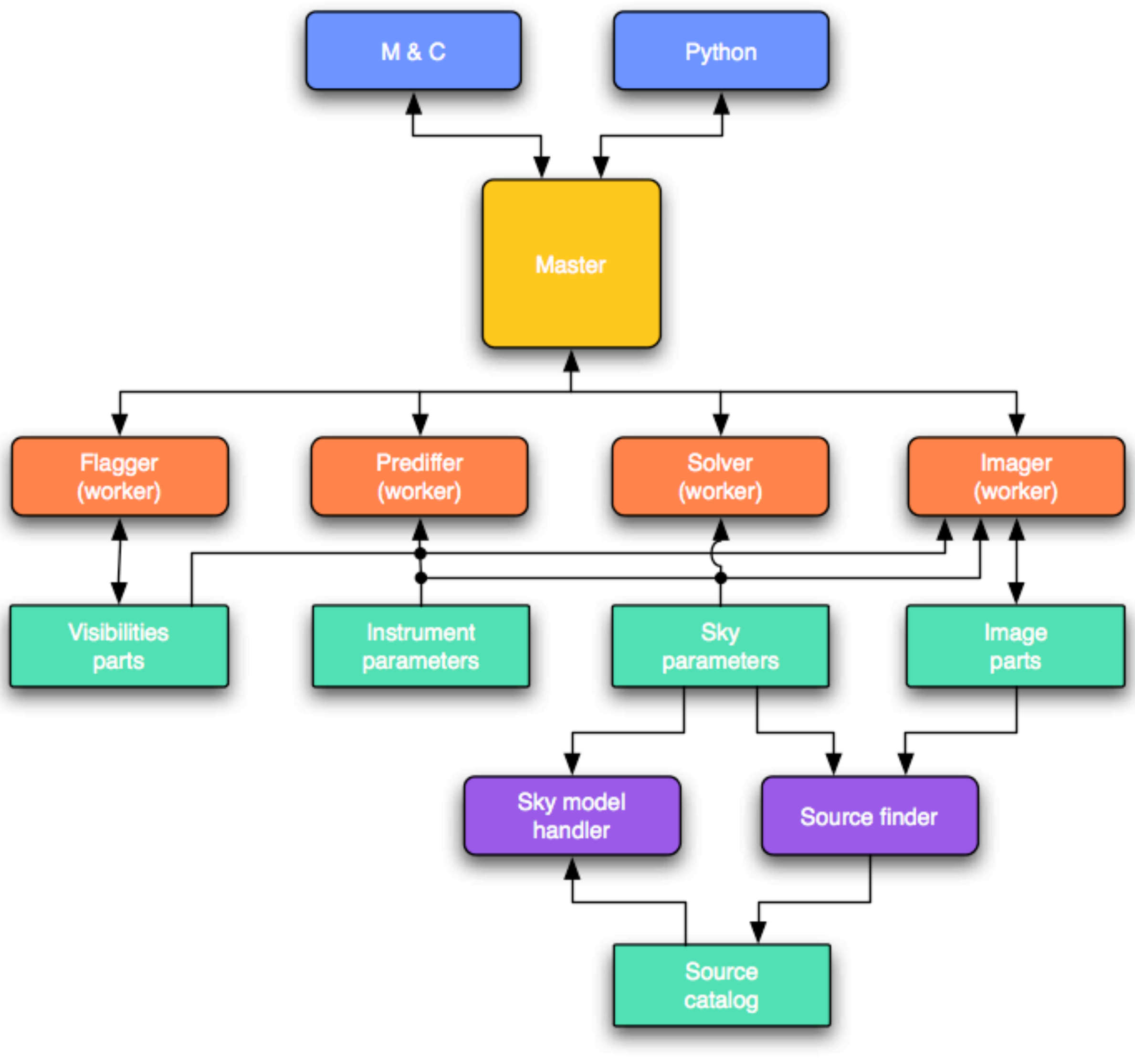


Figure 8 Central processor overview showing various classes of workers under the control of one master. Each worker is responsible for dealing with some aspect of the overall data processing.

6.2.2 Distribution of the data

The architecture of the data processing system has to be efficient with respect to IO, both disk and network.

As described above, the observed visibility data must be stored in a distributed way. The best axes to partition the visibility data are beam and spectral band. The image data can to be distributed in the same way. With this data distribution, the foreseen processing tasks can be executed without sending a lot of data between processes. Other partitions (*e.g.* in time or baseline) would require much more communication for one or more of the tasks.

The advantages of partitioning data by beam and frequency band are:

- Each beam/band can be flagged independently. If a band is stored distributed, some data exchange might, however, still be needed.
- Self-calibration needs to find solutions for a non-linear system. As described above, this uses a solver and many prediffers. Each prediffer forms normalised equations by comparing the

predicted UV data with the measured UV data. This requires sending normal equation matrices to a global solver, but usually they consist of few data.
- Dirty imaging can be done independently for each beam/band, and thus requires no data exchange.

The disadvantages are that:

- Deconvolution requires that each imaging prediffer process send normal equations (essentially a dirty image and point spread function) to a global solver (for solving the minor loop). However, for the bulk line data this is needed only once.
- Stitching MIRANdA beams together requires the images of 8 neighbouring beams. Therefore the beams should be stored on the same machine.
- Feature finding in an image cube is done in a separate program [3]. The program will be parallelised. Similarly, source finding in a continuum data cube.

6.2.3 Master/worker control

Distributed processing requires specialised control. Several software design patterns (*e.g.* master/worker, blackboard) exist that deal with distributed processing. As explained in CONRAD-SW-0012 [4], the master/worker pattern is best suited for the type of processing that needs to be done.
The system consists of the following components:

1. The master has the overall control. It gets the step to perform from a higher-level interface (*e.g.* a configuration file or a python script). It transforms such each step to one or more commands, sends them to the workers, and coordinates all these commands. Another important task of the master is to form the work domains as discussed in the next section. Only the master has the full picture, thus the master has to do the main loop over the data domain. Of course, it does not access any low level data.
2. The workers perform the actual work. There will be at least two types of workers:
    a. A prediffer reads the visibility data and forms normalized equations. Furthermore a prediffer can perform tasks like subtracting models and correcting data. The normal equations may be for calibration parameters or for images.
    b. A solver collects all equations and solves the system. The solution may result in updated calibration parameters or updated images.

New types of work can easily be added, because the master does not need to interpret the data received from workers. It only interprets a preamble and forwards the data to the correct worker. Of course, the master does need to understand new types of high-level steps.

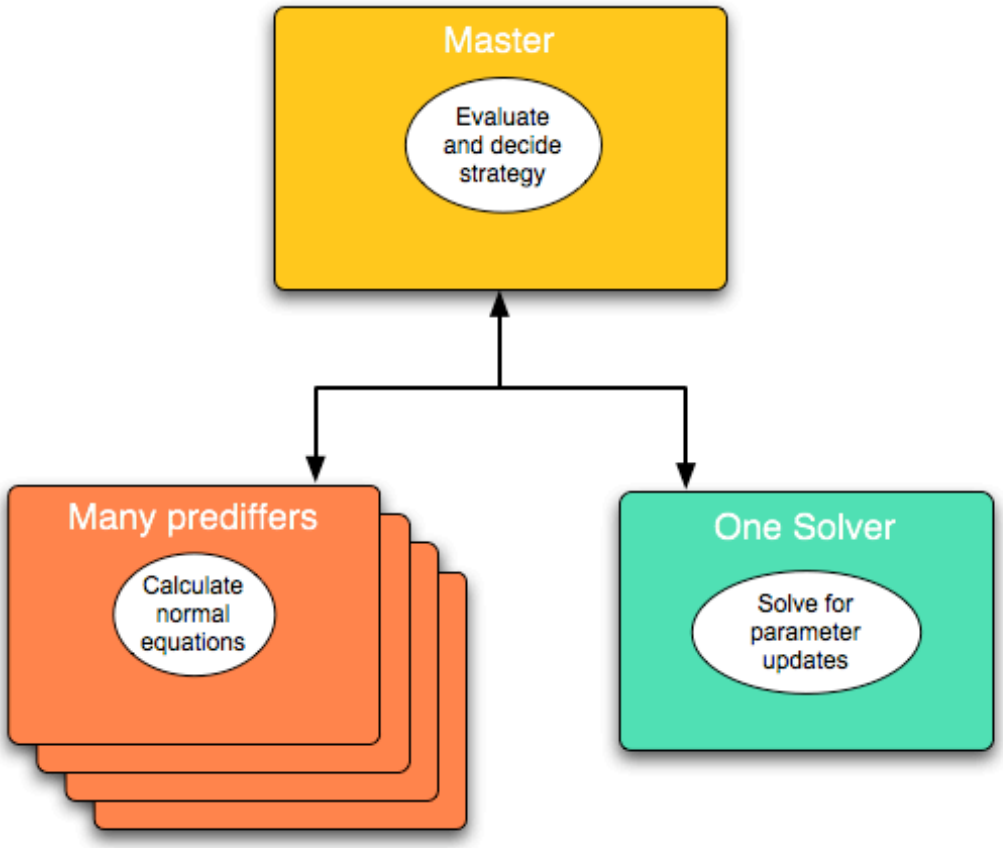

Master
Evaluate and decide strategy
Many prediffers
Calculate normal equations
One Solver
Solve for parameter updates

Figure 9 Master/worker design for a simple calibration procedure where multiple prediffers are used to send normal equations to one solver.

6.2.4 Interprocess communication

In principle there is no direct communication between workers. Instead all communication flows through the master. In that way the master can synchronise the processing. However, there can be cases where this can become a bottleneck. So when needed, the master can also be asked for assistance in setting up a direct connection between workers.

The software will be able to use several types of communication. Support for MPI and for sockets will be provided. MPI is useful for some supercomputers that do not support sockets or are optimized for MPI.

6.3 Visibility Data Access

Due to the large amount of data, the number of times the data can be accessed is limited. Therefore the data processing needs to be done such that as little IO as possible is done. This means that as many processing steps as possible should be done on a chunk of data that can be kept in memory.

Each chunk of data is called a work domain, i.e. the domain that a worker can be held in memory. By using selection (e.g. only long baselines) or by using data integration, the size of the work domain can be reduced.
It will not be possible to perform all processing steps in a single iteration over data chunks. Some steps will require different work domain sizes. It is important though to keep the number of iterations to a minimum.

For calibration a work domain will usually hold all frequency channels and a limited time range. For imaging, all times are needed, so only a limited number of channels can be held in memory. This means that the visibility data are stored in a way that both access patterns can be dealt with efficiently. One possible solution is to store continuum and spectral line data separately.

6.4 Robustness

Due to the distributed nature of the system, it is vulnerable to failures of disks or machines. Implementing sufficient duplication and fault handling code can increase robustness.

- Data can be replicated. This can be done using RAID systems or by writing the data twice or even more times. This is especially important for the continuum data. Note that this makes it possible to speed up the processing by using all copies. If there is no copy and a disk fails, the data should be discarded.
- The master must be able to deal with failures in the workers. If possible, it transfers work to another worker, otherwise the data are lost.
- A watchdog like process can be used to check if worker processes still exist and to restart them if needed and possible. This can also be used to initially start all processes.

Robustness will be built in over time. The initial test system will not have much robustness.

6.5 Interface to Monitoring and Control

The Central Processor will appear as a single system to M&C (provided by CTOS), so only the master process communicates with M&C. The M&C interface should be abstract in order to allow that an M&C system need not be used or that another M&C system is used (*e.g.* for LOFAR).

M&C will mainly consist of monitoring the progress of the data processing. It has yet to be defined which monitoring points will be built in. M&C does not have detailed control over the data processing. M&C will monitor the hardware the data processing is running on.

7 The CONRAD Data Store

The CONRAD Data Store (CDS) will store all the persistent data for a CONRAD telescope:

- Raw data (similar to the main table of visibility data set of AIPS++, the MeasurementSet) from the Correlator, and Catalogue, image (cube), and calibration data from the Pipeline.
- Metadata from the Receivers, the Beamformers, the Correlator, Site Monitor, through the Telescope Policy Manager.
- Observing proposals, observation descriptors, and logs from the Policy Manager of the Telescope Control. These data may be called miscellanea.

The visibility data volumes for MIRANdA are substantial (many Terabyte per hour). The actual decision as to whether the full spectral visibility data are to be retained in the CDS will be deferred to as late as possible.

Metadata, miscellanea data, image headers, catalogue, calibration data will go to metadata database(s). Several metadata databases may be used for searching efficiency. Raw data and images will be stored in raw disk files and indexed by a database.

7.1 Internal use of the CDS

The telescope state is stored in two places – first, in the raw monitor data collected by CTOS and stored in a Round Robin Database, and second, in a processed, filtered, and conditioned form in the CDS.

Only three subsystems will access directly the CDS:

- CTOS sends the conditioned monitor data to the CDS via the Telescope Policy Manager, and also performs high level monitoring of the CDS,
- The Data Conditioner sends visibility data to the local storage of the Central Processor and optionally to the CDS
- The Central Processor queries the CDS for the meta-data necessary for pipeline processing. In addition, the CP will send processing results to the CDS.

The Central Processor has considerable scratch storage available as needed for processing – perhaps up to 1 TB per computational node. This scratch space is volatile and is not visible to or managed by the CDS.

7.2 CONRAD Archive

The CONRAD Archive is the external interface to the CDS. This interface will enable the scientific community, most likely astronomers and engineers, to access and use all scientific data obtained by the telescope to do scientific research, to evaluate the performance and capability of the telescope, and to examine historical behaviour of time-varying phenomena of the universe.

The next figure illustrates the entire CDS system and interface to the archive.

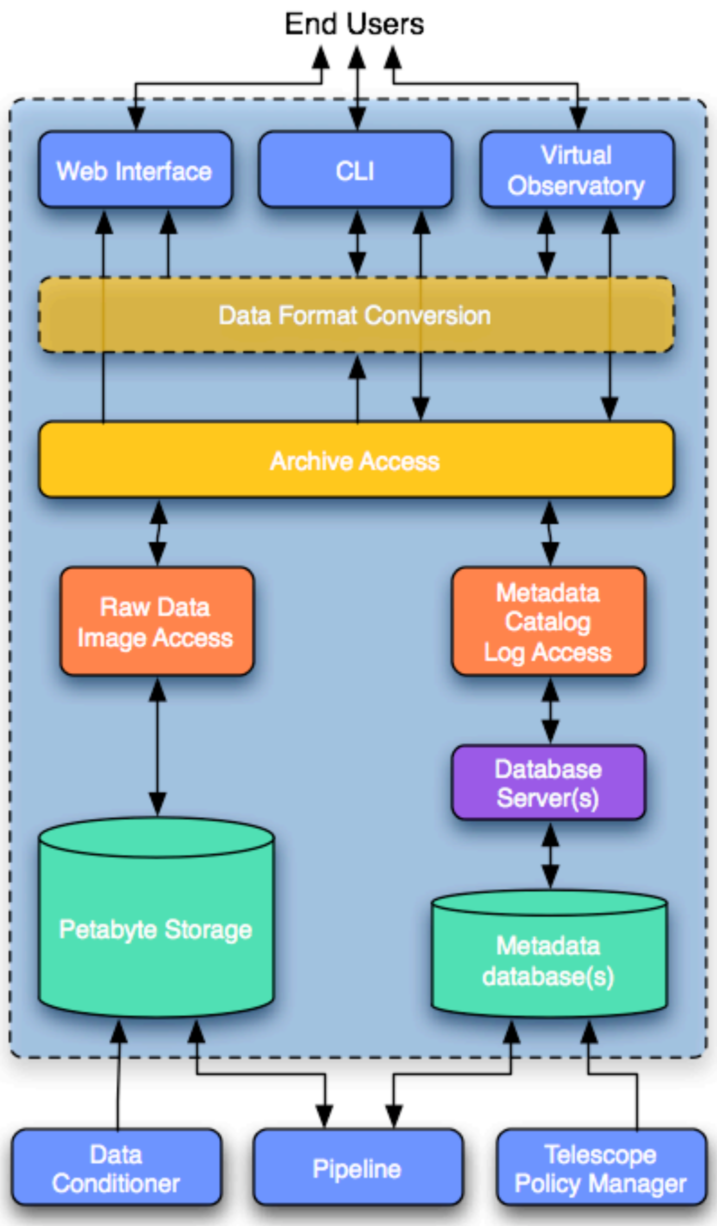


Figure 10 CDS layering

8 User Interfaces

Before discussing the user interfaces it is worth describing, in broad terms, the kind of users we expect to have to service with these UIs:

- Operators – responsible for the day-to-day scientific running of the facility.
- Maintenance staff – responsible for physical maintenance of the facility. They are also the first line of problem solving.
- Engineers – handle escalated support issues as well as upgrade and bug fix existing subsystems.
- Scientists – are the initiators of targeted scientific observations. They will be able to monitor the progress of their particular experiment.
- External scientists – are those who need access to scientific data but are not involved directly in the capture of this data.
- General public – is a consumer of high-level telescope information. This may include some visitor centre requirement on the UI side.
- Students and educators – require appropriate level information between general public and technical levels.

To service the needs of this diverse range of users we have adopted a two pronged approach to the provision of user interfaces. We will have a web based interface that primarily serves the operators, maintenance staff, external scientists and the general public.

A python based interface that exposes the telescope functionality in a slightly more raw form would be of most use to the engineers and perhaps scientists directly managing their own experiments.

8.1 Web Interface

This is the primary interface that will serve the largest group of telescope users. Our requirements for this interface are as follows:

- Rich – should provide all the functionality required by the target group in a single delivery mechanism.
- Simple – this does not denote a lack of features, rather that these features are organised in such a way as to provide a logical and understandable interface for the end user.
- Lighweight – the user should not have to download or install large plugins.
- Simple backend – we want to avoid complex third party web frameworks that will require ongoing maintenance and support.
- Cross platform – all standard platforms (*nix, MacOS, Windows) should have easy and standard access to the interface.
- Centrally managed – we must absolutely avoid the need to manage client software on the end users machines.

At this stage it would appear that a web delivered application using modern lightweight techniques such as AJAX can meet all of these requirements.

The diagram below shows the component breakdown for the web interface as it is currently planned:

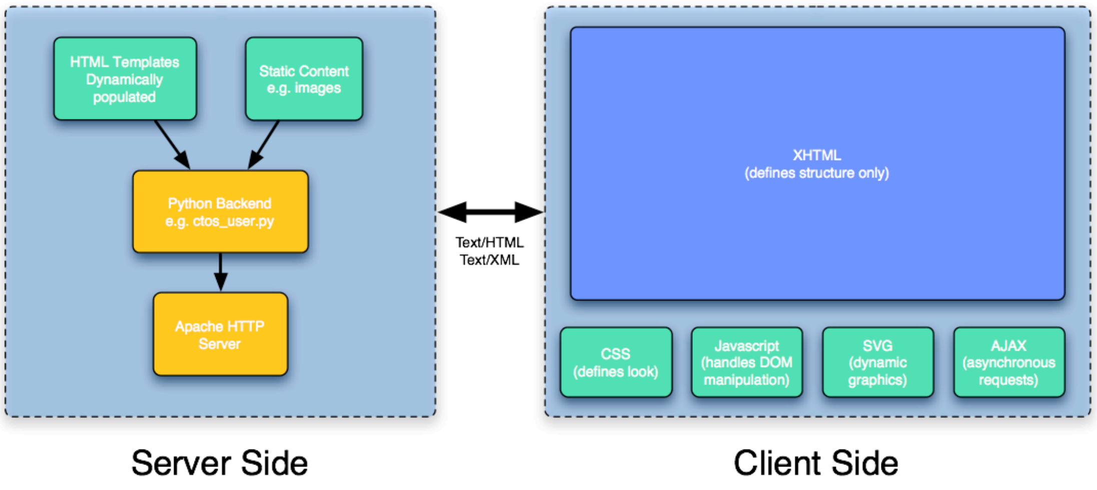

HTML Templates Dynamically populated
Static Content e.g. images
Python Backend e.g. ctos_user.py
Apache HTTP Server
Text/HTML
Text/XML
XHTML (defines structure only)
CSS (defines look)
Javascript (handles DOM manipulation)
SVG (dynamic graphics)
AJAX (asynchronous requests)
Server Side
Client Side

Figure 11 User interface factoring

The interface view is split into two to organize the more technical and administrative functions in a more logical grouping.

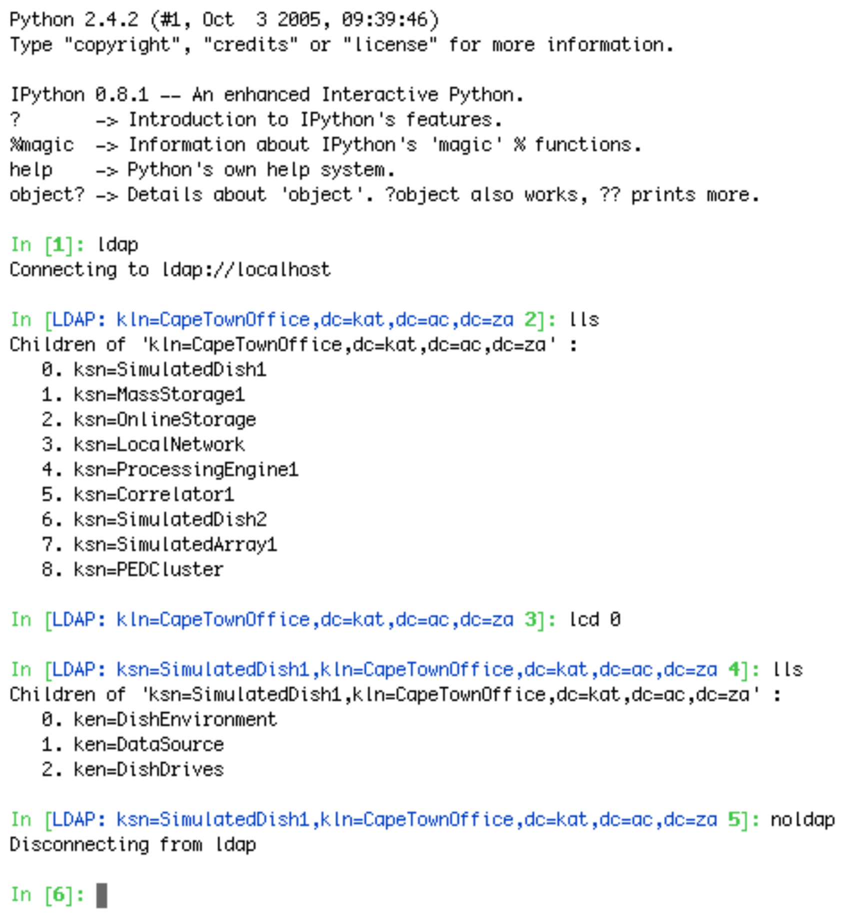
The user side is primarily concerned with managing and deploying system observations and experiments. It provides for a relatively unsophisticated end user.

The technical side contains access to the monitoring system, provides for individual control of the various subsystems, allows access to the LDAP repository, handles the configuration management system and various other technical views.

8.2 Python Interface

The python interface provides for script and interactive shell access to the telescope as a whole. Delivered as a python module that can be used directly in scripts or imported into an iPython shell for interactive telescope control, this interface will be primarily aimed to engineers and advanced scientific use.

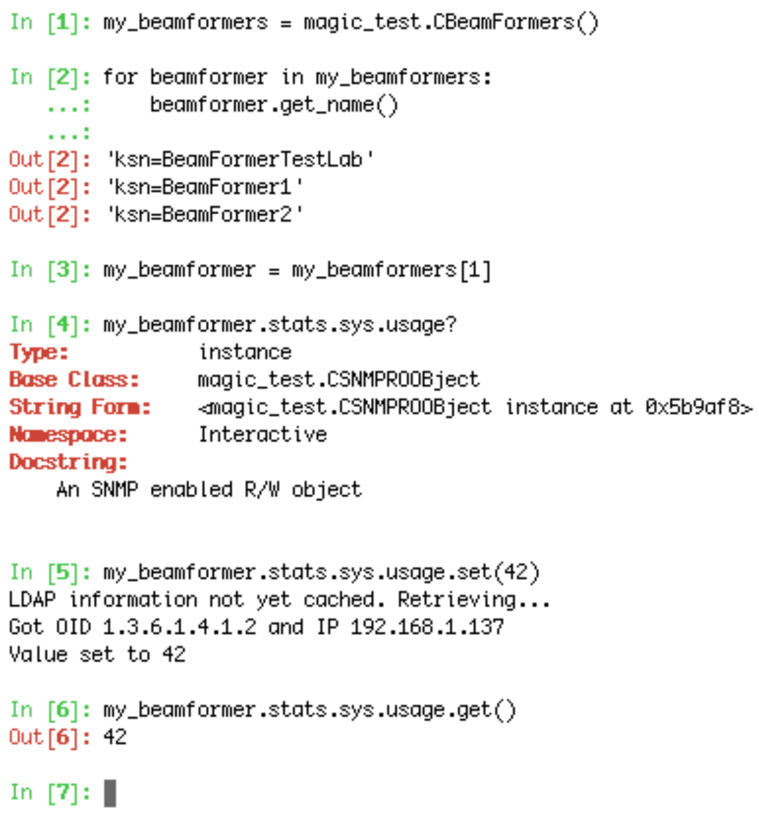
Two major groups of functionality are delivered. The first of these are a range of high level system commands that are wrapped up and exposed for ease of low-level access. For example we have commands (primarily useful in the iPython environment) for accessing and browsing the LDAP repository as if it were a local structure on disk:



```
Python 2.4.2 (#1, Oct  3 2005, 09:39:46)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.1 -- An enhanced Interactive Python.
?       -> Introduction to IPython's features.
%magic  -> Information about IPython's 'magic' % functions.
help    -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.

In [1]: ldap
Connecting to ldap://localhost

In [LDAP: kln=CapeTownOffice,dc=kat,dc=ac,dc=za 2]: lls
Children of 'kln=CapeTownOffice,dc=kat,dc=ac,dc=za' :
   0. ksn=SimulatedDish1
   1. ksn=MassStorage1
   2. ksn=OnlineStorage
   3. ksn=LocalNetwork
   4. ksn=ProcessingEngine1
   5. ksn=Correlator1
   6. ksn=SimulatedDish2
   7. ksn=SimulatedArray1
   8. ksn=PEDCluster

In [LDAP: kln=CapeTownOffice,dc=kat,dc=ac,dc=za 3]: lcd 0

In [LDAP: ksn=SimulatedDish1,kln=CapeTownOffice,dc=kat,dc=ac,dc=za 4]: lls
Children of 'ksn=SimulatedDish1,kln=CapeTownOffice,dc=kat,dc=ac,dc=za' :
   0. ken=DishEnvironment
   1. ken=DataSource
   2. ken=DishDrives

In [LDAP: ksn=SimulatedDish1,kln=CapeTownOffice,dc=kat,dc=ac,dc=za 5]: noldap
Disconnecting from ldap

In [6]: █
```

This easy access allows for direct manipulation of the ldap repository without resorting to graphical tools. Other functionality such as SNMP control, logging, and schedule management are also provided.

The second category is the exposure of all the system control and monitor points through an object based hierarchy. This allows scripts to have easy raw access to the configuration of the system as well as allowing the inclusion of monitoring data as feedback into these scripts. An example of this is shown below:



```
In [1]: my_beamformers = magic_test.CBeamFormers()

In [2]: for beamformer in my_beamformers:
   ...:     beamformer.get_name()
   ...:
Out[2]: 'ksn=BeamFormerTestLab'
Out[2]: 'ksn=BeamFormer1'
Out[2]: 'ksn=BeamFormer2'

In [3]: my_beamformer = my_beamformers[1]

In [4]: my_beamformer.stats.sys.usage?
Type:           instance
Base Class:     magic_test.CSNMPROObject
String Form:    <magic_test.CSNMPROObject instance at 0x5b9af8>
Namespace:      Interactive
Docstring:
    An SNMP enabled R/W object


In [5]: my_beamformer.stats.sys.usage.set(42)
LDAP information not yet cached. Retrieving...
Got OID 1.3.6.1.4.1.2 and IP 192.168.1.137
Value set to 42

In [6]: my_beamformer.stats.sys.usage.get()
Out[6]: 42

In [7]: █
```

This low level access will be derived directly from the MIBs for the various subsystems and as such will include every control and monitor point on the facility. The first time in a session that an end point is manipulated, LDAP information on the current IP and OID for that point are retrieved and cached. Thereafter the cached information is used (LDAP modifications trigger a global cache reset).

In the example above it shows how multiple subsystems of the same type are handled by providing a wrapper object with iterator behaviour.

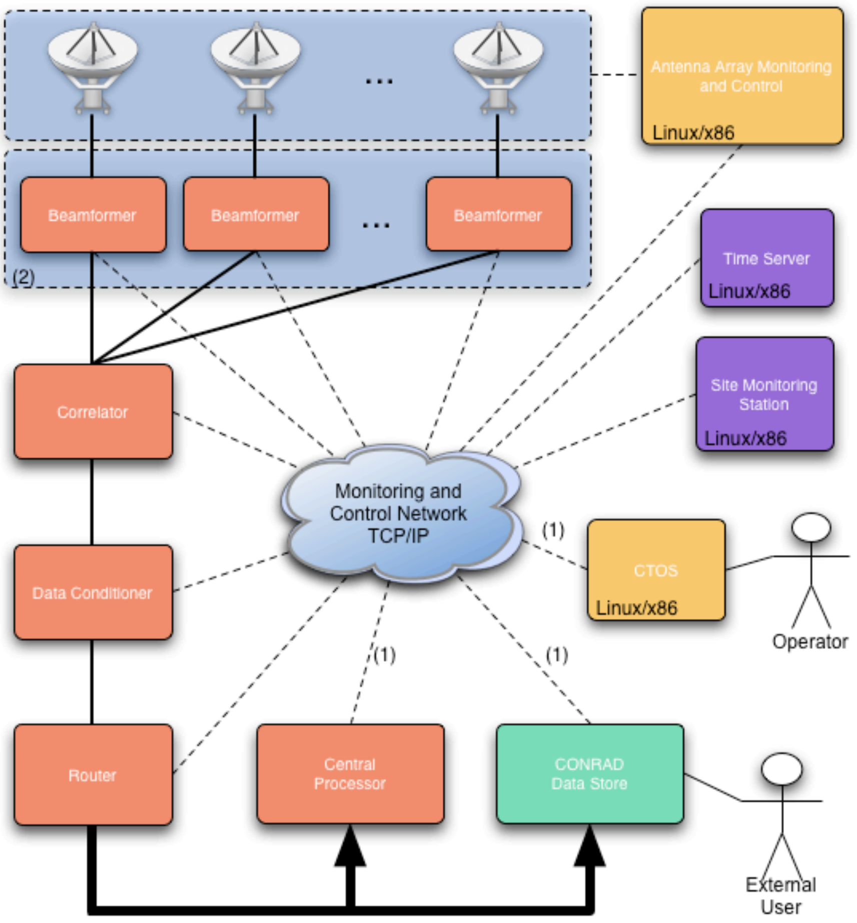
For monitoring points direct access to the current live value is provided via get() whilst a list of the last n points is available via get(n).

9 Deployment

Figure 12 shows the physical deployment of CONRAD on each telescope (MIRANdA and meerKAT). The diagram presents only external connectivity and does not describe the internal components of each node and their relationships.

The default workstation used on each physical node is Intel/AMD (x86 or x86-64 architecture) based PC running the linux operating system. Although some subsystems cannot be deployed in the default workstation, our design approach is to use the default platform as much as possible among the subsystems to keep maintenance cost at minimum (homogeneous system).

Two physical networks co-exist in the system: a high-speed data network for exchanging visibility data sets and an inexpensive network (ethernet) for monitoring and control. The network layout will take advantage of virtual local area network (VLAN) as much as possible.



- - - - - SNMP Based Monitoring and Control Network.

——— Dedicated link. Protocols To Be Defined.

▬▬ High Speed Data Interconnect.

(1) Central Processor and CTOS also exchange metadata with the Archive Subsystem through a customized TCP/IP interface.

(2) MIRANdA only.

Figure 12 Physical deployment of CONRAD

A more detailed description of each physical node in the deployment diagram is outlined in the following subsections.

9.1 Antenna Control

A dedicated workstation (linux x86 or x86-64) is to be used to control each antenna. The responsibility of the antenna control computer is to pass (and convert) the control commands from CTOS to the antenna as well as return monitoring information via SNMP interface.

9.2 Time Server

The time server is responsible to supply accurate time reference to the rest of the subsystems. It is connected to the reference time generator (atomic clock, GPS), which provides the accurate clock. Sub-systems implementing hard real-time control must know the time with 1 microsecond accuracy. High time accuracy is obtained using specialized hardware and connection to the time server. Sub-systems that do not require accurate time synchronization can obtain the time via network time protocol (NTP). No hard real-time closed loops are implemented within CTOS.

The time server consists of a Linux x86 or x86-64 workstation connected to the monitoring and control network via ethernet card. Monitoring information relevant to the operation and maintenance of the time is pass to CTOS via SNMP interface.

9.3 Site Monitoring Station

The site monitoring station is responsible for acquiring all the environmental and site information and sends them to CTOS via SNMP interface, such as power information, local weather, temperatures, humidity, etc. A linux x86 or x86-64 connected to the monitoring and control network via ethernet card will be used.

9.4 Beamformers

The focal plane array beamformers are present only in MIRANdA. Array beamformers will be present in both telescopes. The target hardware is yet to be defined. Each beamformer is connected to the monitoring and control network and provides monitoring and control information (such as weights) via SNMP interface.

9.5 Correlator

The correlator is responsible to produce the visibility data. Hardware implementation is yet to be defined. The correlator output is connected directly to the data conditioner via a dedicated parallel link. The correlator is also connected to the monitoring and control network and provides monitoring and control information to CTOS via SNMP interface.

9.6 CONRAD Data Conditioner (CDC) and Router

The CONRAD data conditioner (CDC) is responsible for flagging and averaging the visibility data. The output goes directly to the router via a dedicated parallel link. The router distributes the data to the CONRAD data store (CDS) and the central processor (CP). Both the CDC and router hardware implementations are yet to be defined. Both the CDC and router are connected to the monitoring and control network and provide monitoring information to CTOS via SNMP interface. The CDC may need access to meta data to perform its function.

9.7 Central Processor (CP)

The CP is responsible of processing and calibrating the data. For more detailed information about the responsibilities of the CP, see Section 6.

It is clear that to fulfill the strong requirements on processing speed and volume of data to be processed a high performance machine is needed. Furthermore a high disk-IO bandwidth is needed to achieve the required processing speed.

If the data are split in frequency, the processor interconnect bandwidth does not need to be very high. Gigabit ethernet will probably be sufficient, but the demands of flagging, image deconvolution, and source and feature finding are not clear yet.

The central processor is also connected to the monitoring and control network to exchange monitoring and control information via SNMP interface as well as to the CDS to exchange metadata information via customized interface (possibly using sockets).

The final hardware to be used is not defined yet. However several alternatives are being evaluated:

1. A cluster of multi-processor, multi-core systems. Preferably the cluster must be homogeneous, but it could be an option to have powerful machines for processing the more demanding jobs like the continuum data. The cluster nodes may optionally have accelerators such as the Cell processor, GPU or FPGA.
2. A supercomputer like the Cray XT3 or IBM BlueGene/L.

The software architecture is such that is possible to adapt to different platforms with the development of some specialised code for the accelerators.

We plan to equip each computational node with sufficient memory to minimise unnecessary disk accesses. The memory bandwidth must be sufficient to keep all cores supplied with data.

When using a cluster a few disk storage options can be used:

1. One or more local disks can be attached to each machine
2. For fault tolerance, a RAID can be attached to each machine or some machines (e.g. only the machines processing continuum data).
3. The cluster can be subdivided into smaller parts (with faster interconnects) and a large RAID system attached to each part. Optionally all these RAIDs can be combined into one parallel file system (e.g. Lustre).

For the Central Processor needs, the key parameters to evaluate between the alternatives are:

- Compute to I/O ratio matches the data processing needs.
- Cost.

At the moment is not yet clear what the compute to I/O ratio should be. However, we believe that the most cost effective solution is more likely to be a commodity cluster.

9.8 CONRAD Data Store (CDS)

The CDS is responsible of storing the raw and reduced data as well as metadata for later access. For a detailed description of the CDS, see Section 7. The hardware platform and RDBMS are yet to be defined. Although the metadata does not require vast amounts of disk space, the visibility data and image cubes do require a high-volume storage device (hundreds of TB) yet to be defined.

9.9 CONRAD Telescope Operating System (CTOS)

CTOS provides high-level control and is also responsible for managing all the monitoring information coming from all the subsystems. For more detailed information see Section 4.1. CTOS is deployed in one or two linux x86 or x86-64 workstations connected to monitoring and control network via Ethernet cards. Final component deployment on each workstation is yet to be defined. The Telescope Policy Manager (TPM) component exchanges metadata information with CDS using a customized interface via the monitoring and control network.

10 CONRAD software development processes

CONRAD software development processes do not enforce a specific architecture. They have been set up to support the project philosophy as described in section 3 and to facilitate geographically distributed development.

10.1 Structure

The CONRAD repository is split into three main areas:

- 3rdParty - packages not under development by the CONRAD team
- Code - packages implemented and maintained by CONRAD developers
- Tools - packages to enable building of code and documentation as well as testing frameworks.

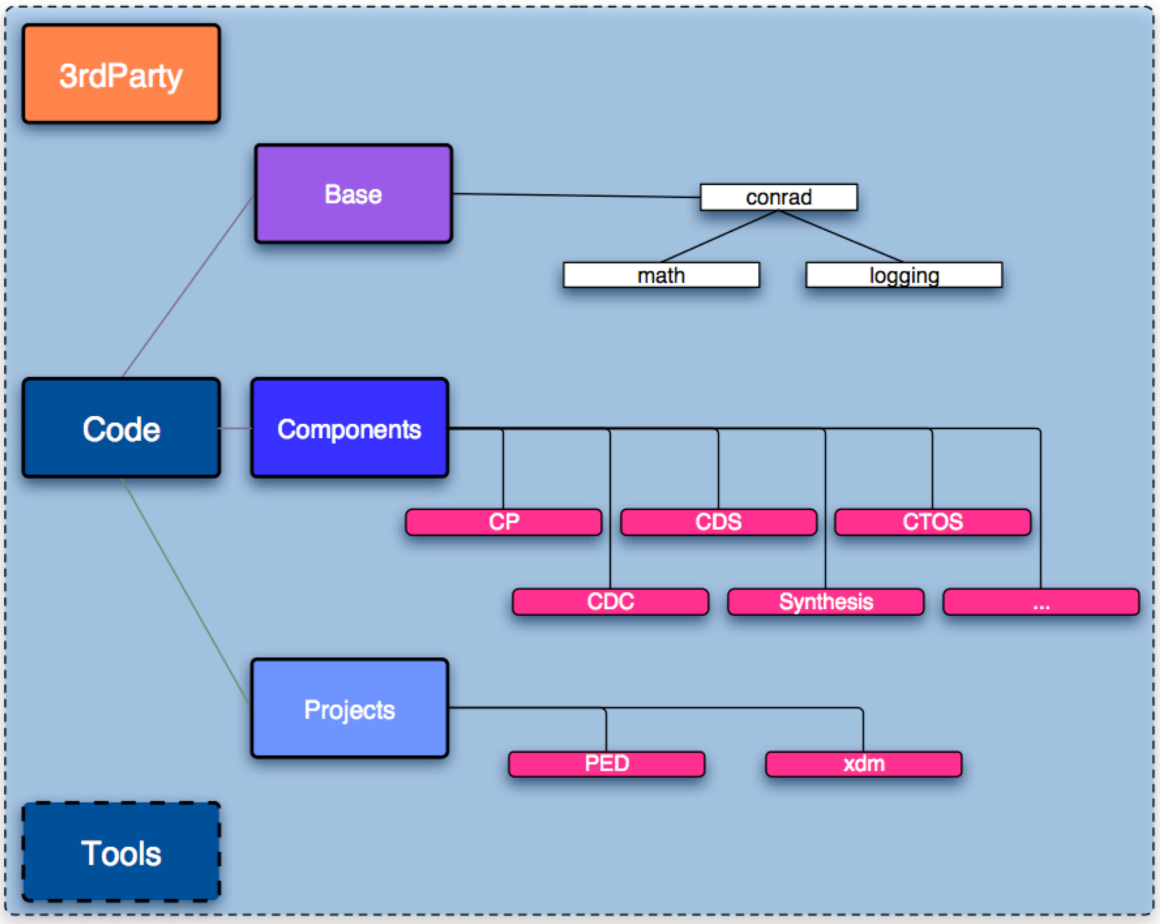


Figure 13 Repository Structure

(Coloured nodes are fixed repository names, white boxes show example packages)

10.1.1 Code

The CONRAD Code repository in turn consists of three main functional groupings:

- Base - common libraries and utilities
- Components - CONRAD work areas, which are shared by the collaboration
- Projects - partner specific integrated systems, e.g. xdm, Parkes testbed

10.1.2 3rdParty

Where possible, commonly used, well-established third party packages should be used for implementations. However, before these can be added to repository permanently a moderation process is required. This prevents bloating and possible duplication of functionality.

10.1.3 Tools

External and internal packages to facilitate the build process

10.2 Dependency Rules

The following dependency rules ensure a non-monolithic, extensible software system.

- 3rdParty packages can only depend on other 3rdParty packages.
- Base packages can depend on 3rdParty and could depend on other base packages.
- Components can depend on other Components as well as Base and 3rdParty
- Projects cannot depend on other Projects, but depend on all others.

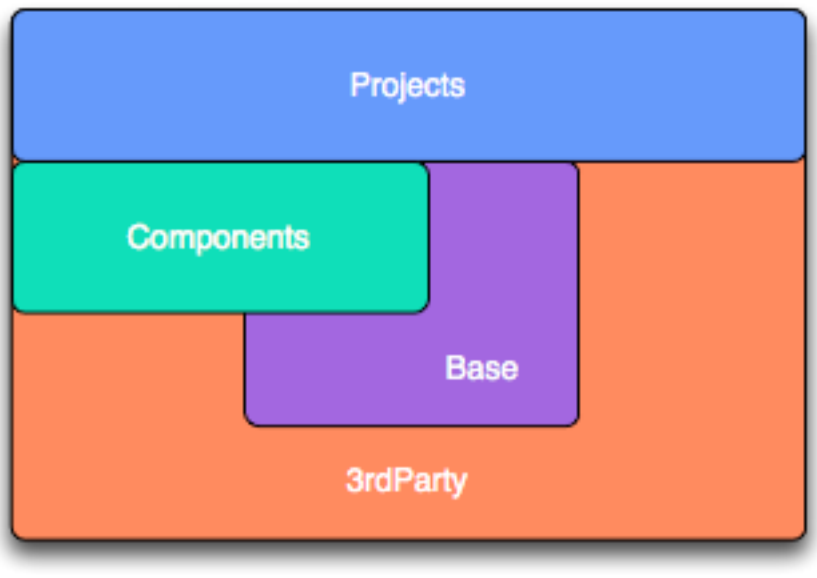


Figure 14 Dependency Hierarchy

10.3 Software Development Life Cycle

The software development life cycle is facilitated through an iterative development process.
To deal with dynamic functional requirements and architectural changes, well-defined guidelines for testing procedures are put in place, as well as design and code reviews.