The 40m Radiotelescope Control System

Pablo de Vicente

Observatorio de Yebes - CDT (IGN)

MINISTERIO DE FOMENTO

DIRECCIÓN GENERAL DEL INSTITUTO GEOGRÁFICO NACIONAL

OBSERVATORIO ASTRONÓMICO NACIONAL

Outline

- A software point of view of a RT: distributed system
- The infrastructure: ACS (ALMA Common Software)
- Programming new devices
- The ACU (Antenna Control Unit)
- The FS (Field System) connection
- Data acquisition
- The pipeline
- The pointing model implementation & analysis
- Lessons learnt





What is the control system of a telescope about?






The radiotelescope is a distributed system

- Operations and computations are distributed in different hosts

- Each physical device is associated to a software object living in a newtork (a "component").

For example:
a component -> Spectral backend

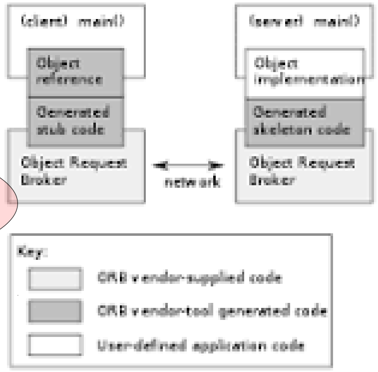
- Components talk to other components via ethernet

- Need for an infrastructure that - eases communication &
  - provides tools

CORBA
via Alma Common Software

What's CORBA ???

The Common Object Request Broker Architecture (**CORBA**) is a standard defined by the Object Management Group (OMG) designed to facilitate the communication of systems that are deployed on diverse platforms.



Common Object Request Broker Architecture - Wikipedia ...
https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture

Feedback

CORBA
via Alma Common Software

Alma Common Software = ACS

The ALMA solution: three CORBA implementations + services

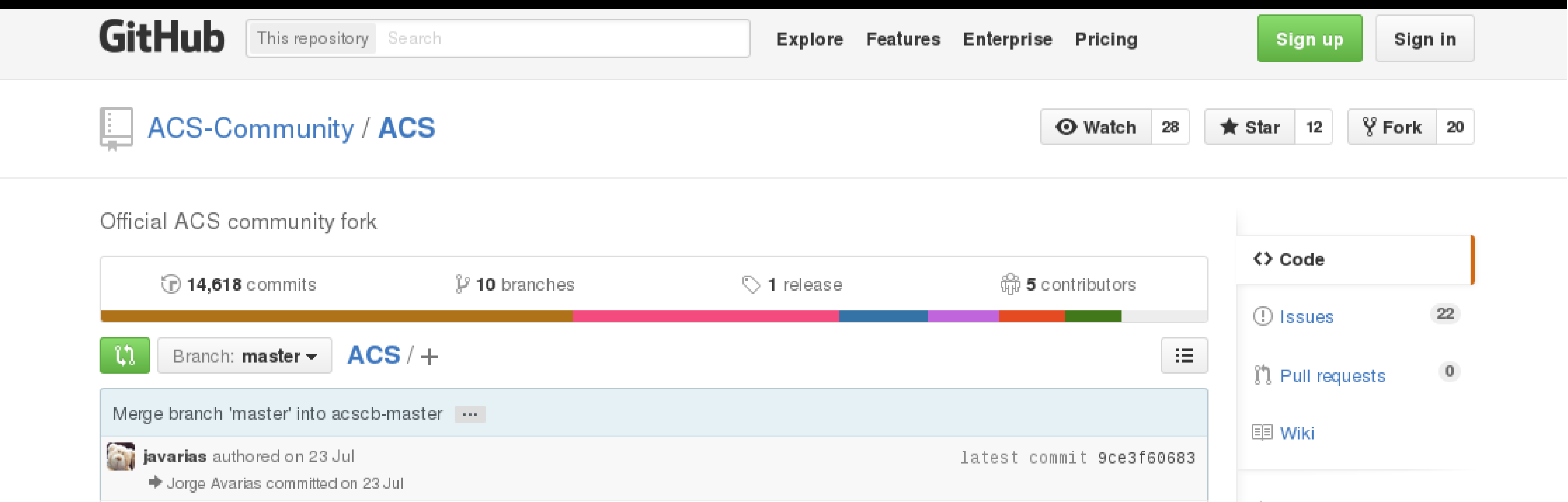
TAO/ACE, **OmniORB** and **JacORB** running in Linux

Why ACS?

- Decission taken in 2004
- CORBA is too complex. ACS hides its complexity to developers
- ACS provides useful tools, services & libraries
- Used by other telescopes
- ACS is free & supported by ESO
- Supports C++, Java, Python

ACS repositories

- Official release frequency ~ once per year
  Redhat Enterprise / Scientific Linux
- Community fork synched with the official release
  Other Linux distributions (Debian, ...)



How to program a component

- Each component exposes to the world:

  Its methods

  [Its characteristics]

- Do we want to generate a notification channel ?

IDL file

- Automatic generation of templates

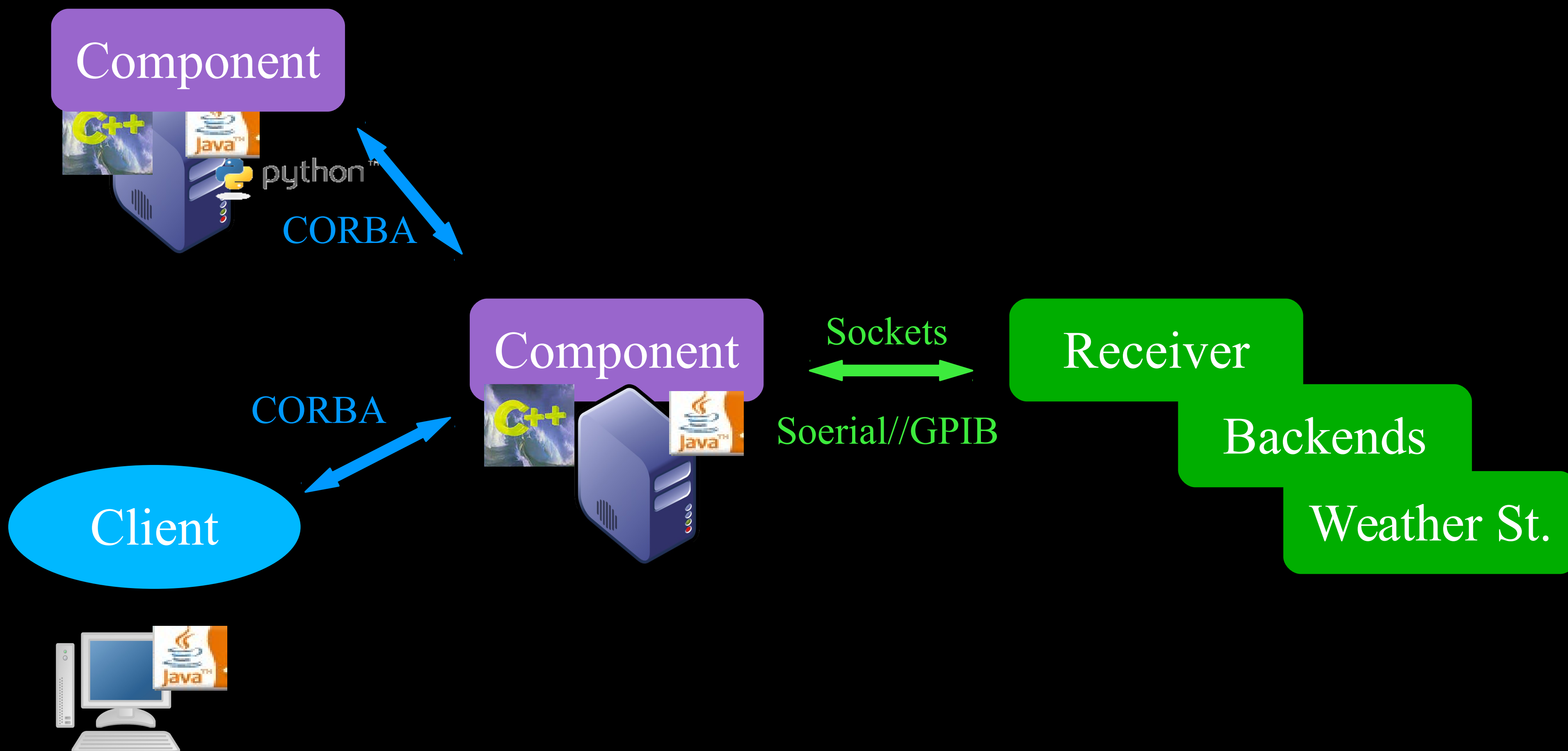
- Is it abstract or is it linked to a physical device?
- Physical device: ethernet/serial/GPIB connection?
- Do we need speed?

C++

Python

Java

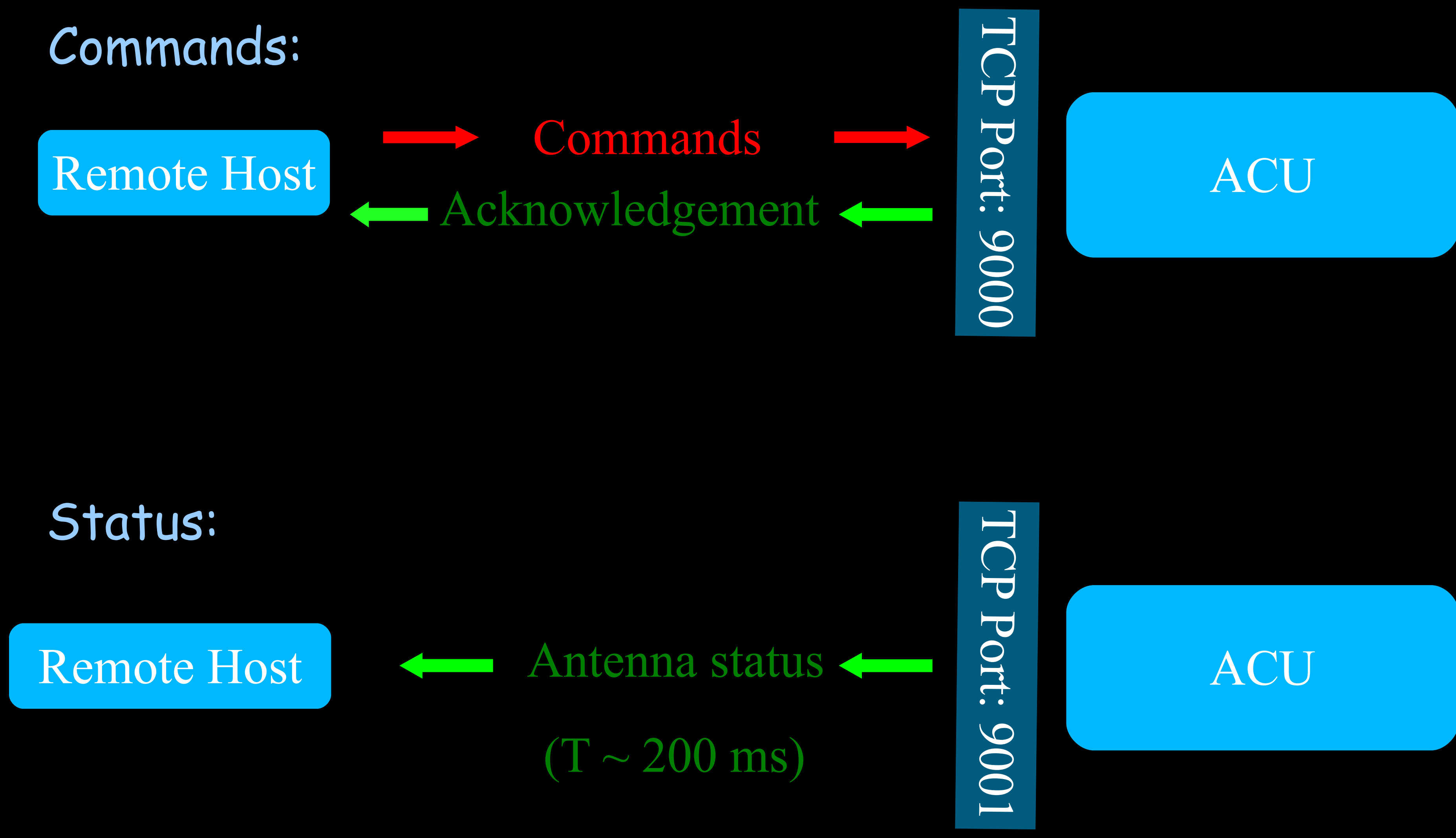
Clients & components relationship



The ACU: Antenna Control Unit

- Computer provided by the telescope building company
- Runs a real time operative system or a real time extension
- Controls the main drives (Az & EL) and the subreflector
- It is usually a black box

How to command and read the ACU remotely



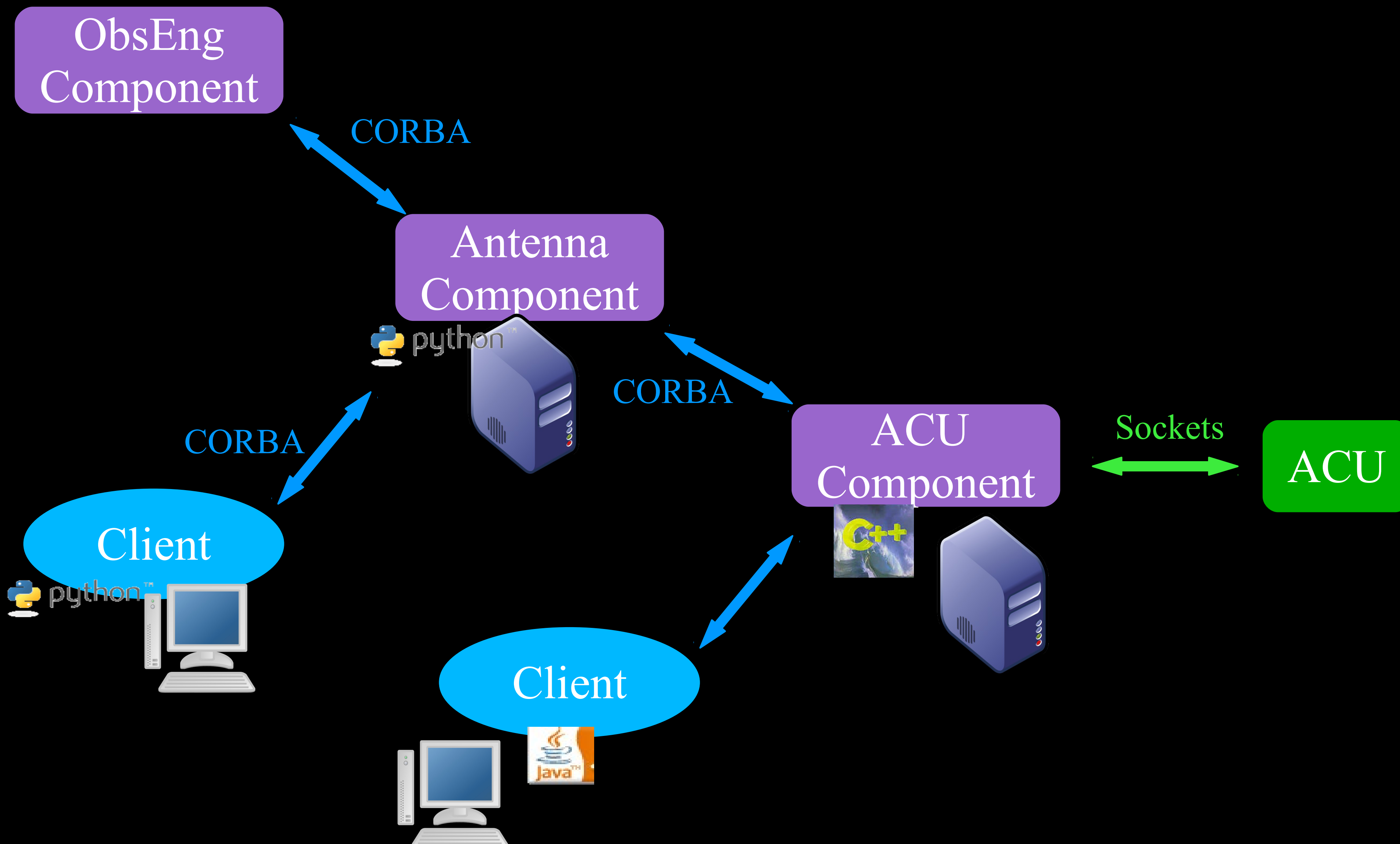
The ACU ICD: Interface Control Document

- Interface between the remote control and the ACU
- 2 Category of commands:

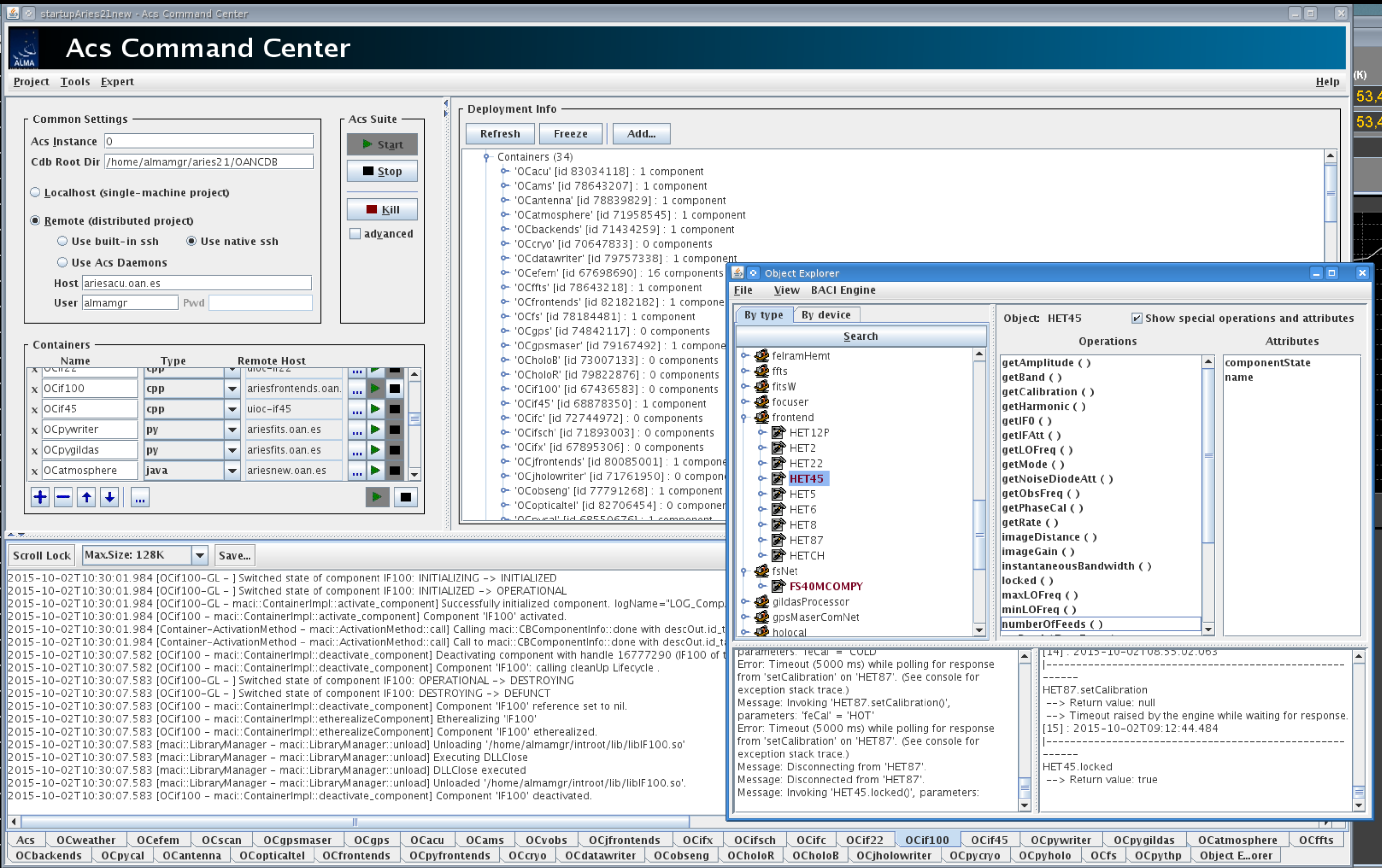
Mode commands	Trajectory & special commands
Positioning	Az/El Ra/Dec tracking tables
Pointing corrs.	Az/El Offset tracking tables
Time source	M2 Offset tracking tables
Master setting	M2 Offset elevation dependency

- Description of command parameters
- Description of status words and fields

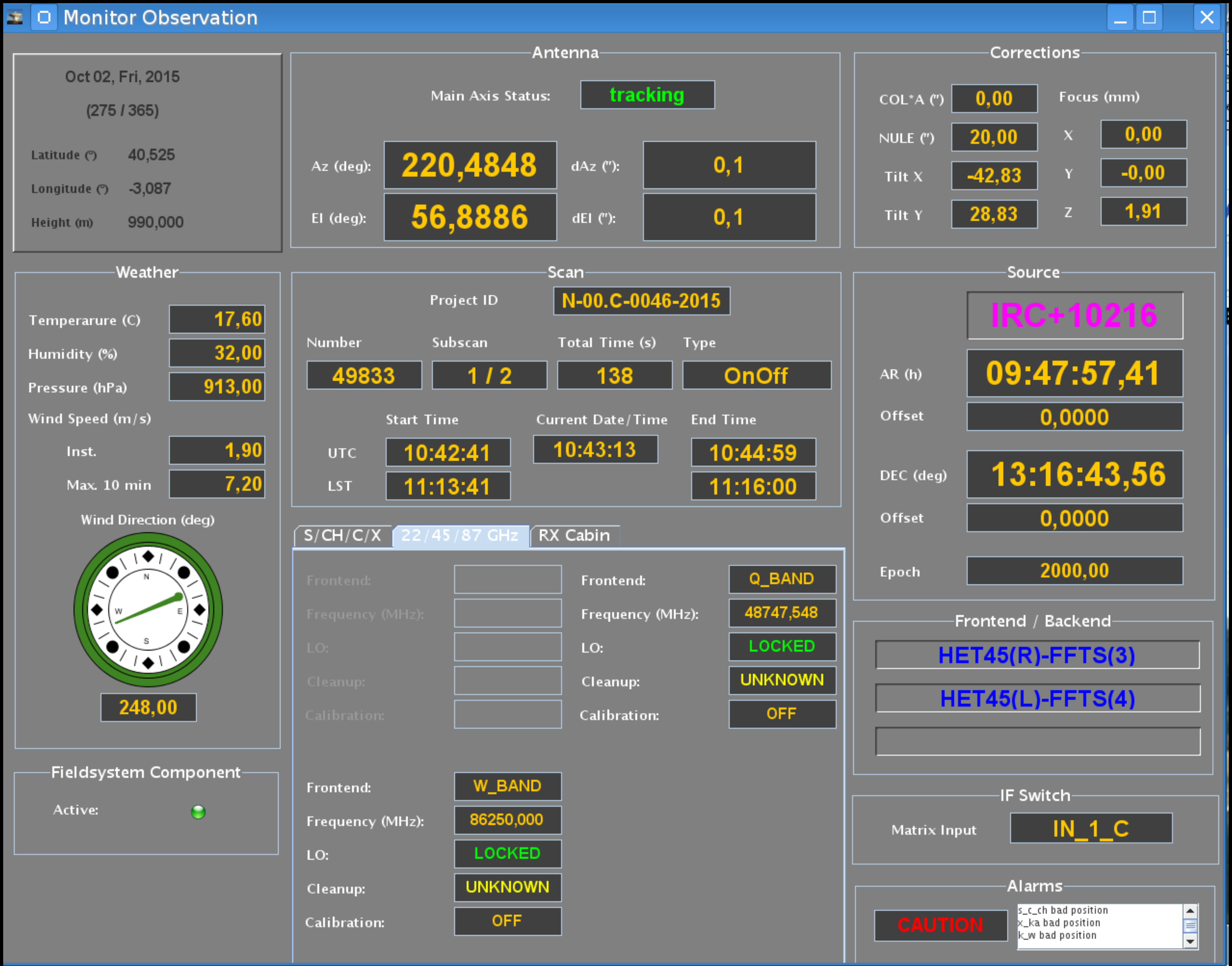
Software layers: abstraction and encapsulation



Graphical clients (ACS generic)



Graphical clients (specific)



Commanding the telescope (example)

Ipython shell

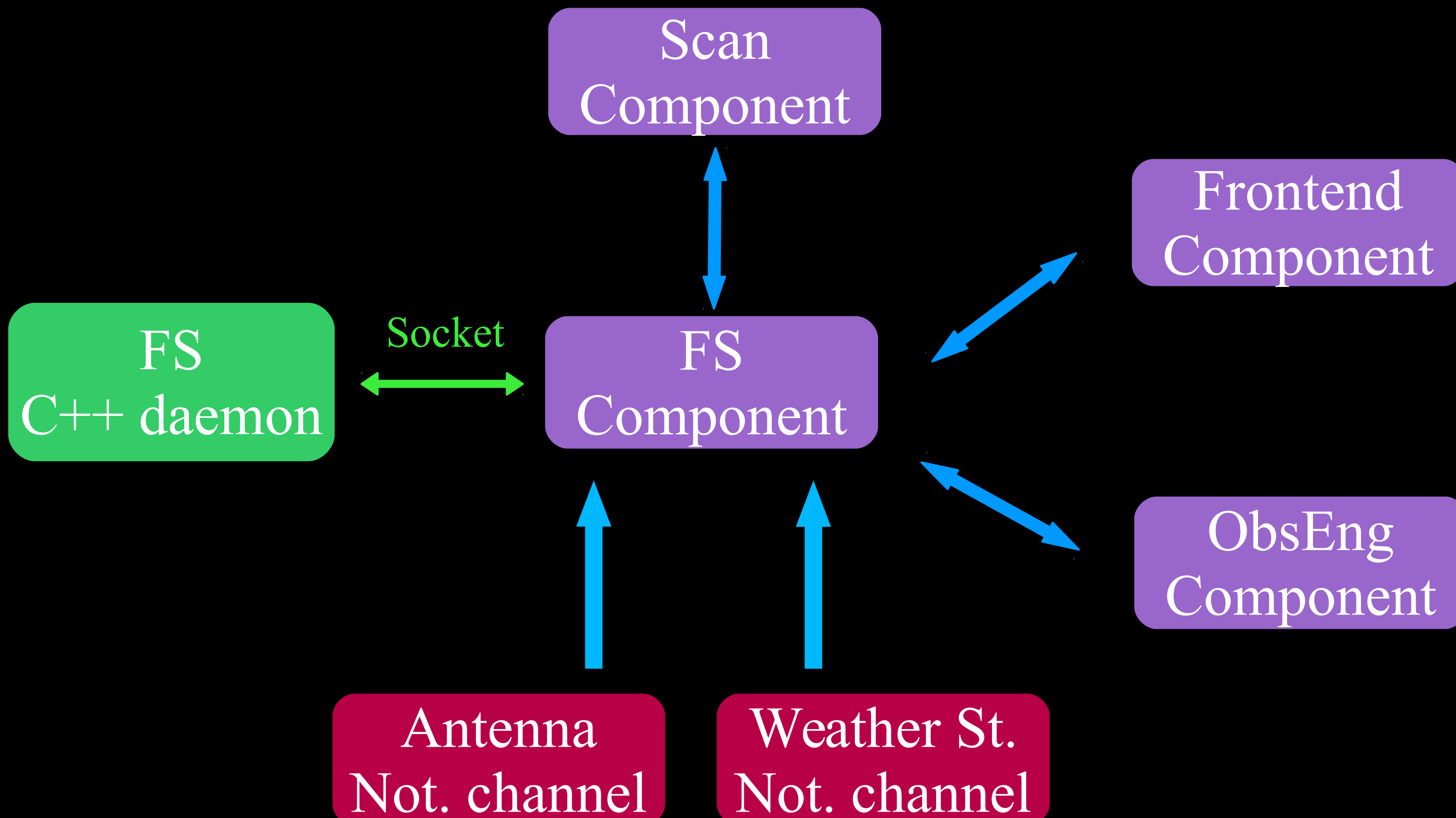
```
Operator ID (NN) ? PV

Observer ID (NN) ? PV

RAEGE> status()
M1:     stopped. Current position:  225.00015, 89.99948
M2:     inactive. Current position:   0.00   0.00  -0.00   0.00   0.00 Errors:  -0.00  -0.00  -0.00  -0.00  -0.00
        El Table offset position:    0.00   0.00   0.00   0.00   0.00


RAEGE> sourcecats(['/home/raegemgr/raege/Catalogs/user.cat'])

RAEGE> source('3C345')
[Errno 2] No such file or directory: 'user.cat'
Multiple matches:
Setting source velocity to 0.0.
Setting source 3C345 @ RA=16:42:58.810, Dec=+39:48:37.00  System: EQ 2000.0, Velocity: 0.00 LSR
Currently at Az=51.659 degs.  El=16.890 degs.
For the time being offsets have to be in the same system as the source system. Command again the offset please.
       (51.6586395371667, 16.889682957316857)

RAEGE> setTrajectoryParams(False, True)

RAEGE> on(60)
Beware: No frontend is being used. Data aquisition not available
Setting dataAquisition to 0
   Setting up an "on" scan with 1 repetitions.
No data acquisition!

 --- Scan: 1 (Type: Others) --- Start time: 08:32:39  End time: 08:33:42

(efem.dateTime(year=2013, month=10, day=15, hour=8, minute=32, second=39, millisecond=195),
 efem.dateTime(year=2013, month=10, day=15, hour=8, minute=33, second=42, millisecond=195))

RAEGE> point(3600,'arcsec',60,'otf',10,'no','x',0)
Data aquisition not available
No data acquisition!

 --- Scan: 2 (Type: Point) --- Start time: 08:34:56  End time: 08:37:33

(efem.dateTime(year=2013, month=10, day=15, hour=8, minute=34, second=56, millisecond=1),
 efem.dateTime(year=2013, month=10, day=15, hour=8, minute=37, second=33, millisecond=1))
```

Setting the source

Track the source 60 secs.

Pointing drift on the source

The FS component: connections in the Control System



Data Acquisition

- Data are captured and written in FITS files in real time

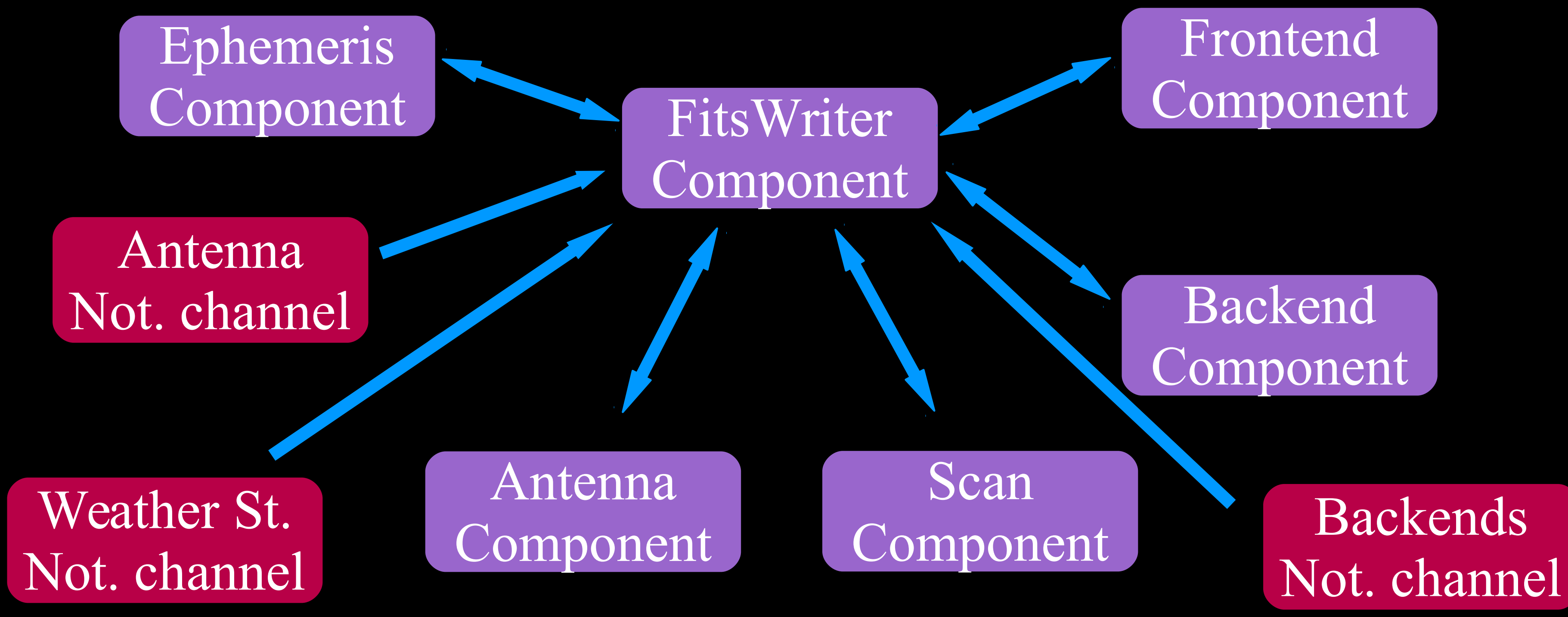
FITS → **File Format Transport System**

Every subscan → ONE file (MBFITS)

MBFITS: Multi Beam FITS

- Data organized in HDUs (Header Data Units)
- *All scan information is stored in FITS headers*
- Data from backends are in tables (**1 HDU per Front-Back**)
- Relative position information in tables
- Auxiliary information (weather, cal, …) in tables

Data Acquisition

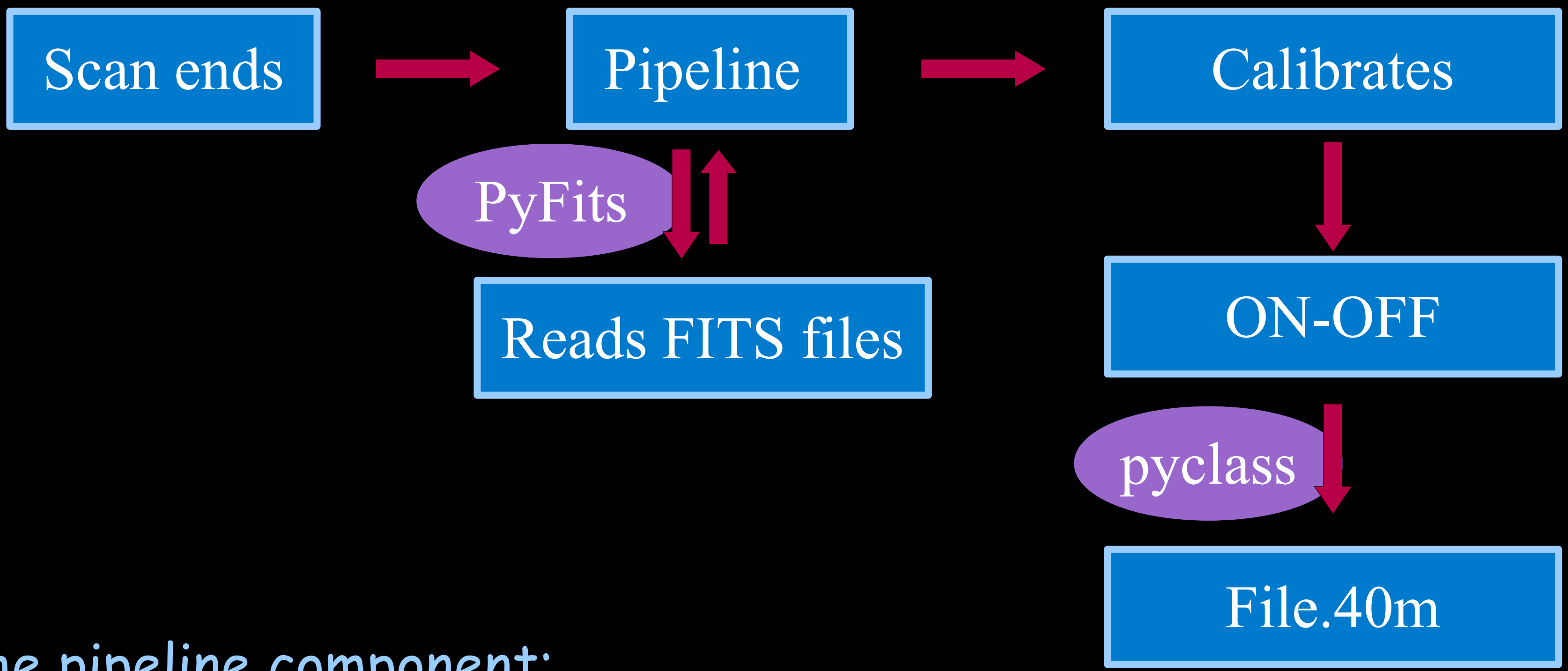


- Component coded in C ++ (faster)
- Component uses cfitsio library
- Component uses a home made C++ cfitsio wraper

Pipeline

- Coded in Python
- The pipeline generates GILDAS files: CLASS format

Pipeline (single dish)



- The pipeline component:
    - Uses a python FITS module
    - Generates calibrated data (atmosphere, hot/load cals)
    - Uses a pyclass filler

The pointing model: implementation

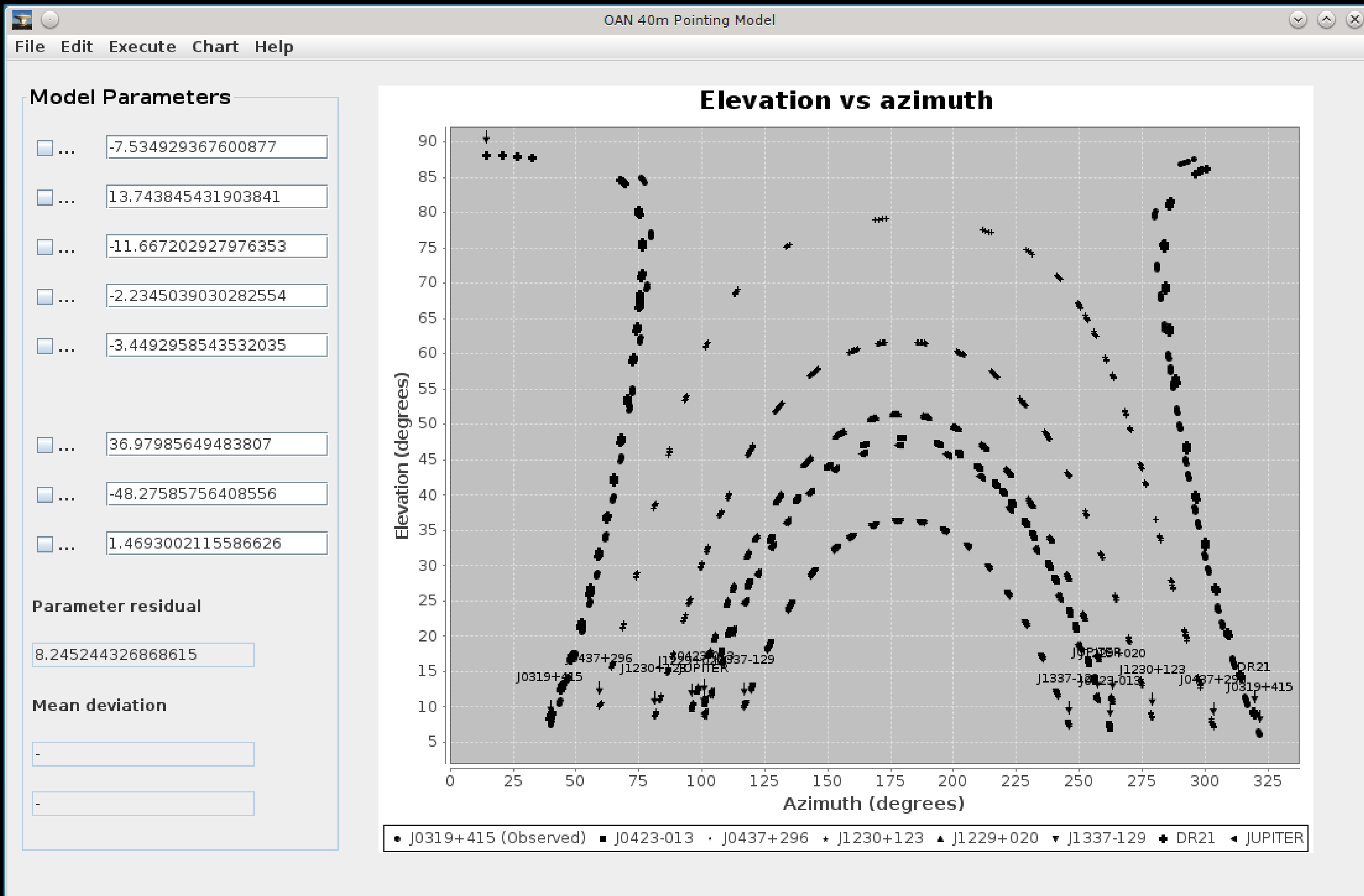
- The ACU implements a pointing model
  - composed of 9 parameters
  - which can be switched off

$$\delta El = P_1 + P_2 \sec(El) + P_3 \tan(El) - P_4 \cos(Az)\tan(El) + P_5 \sin(Az)\tan(El)$$

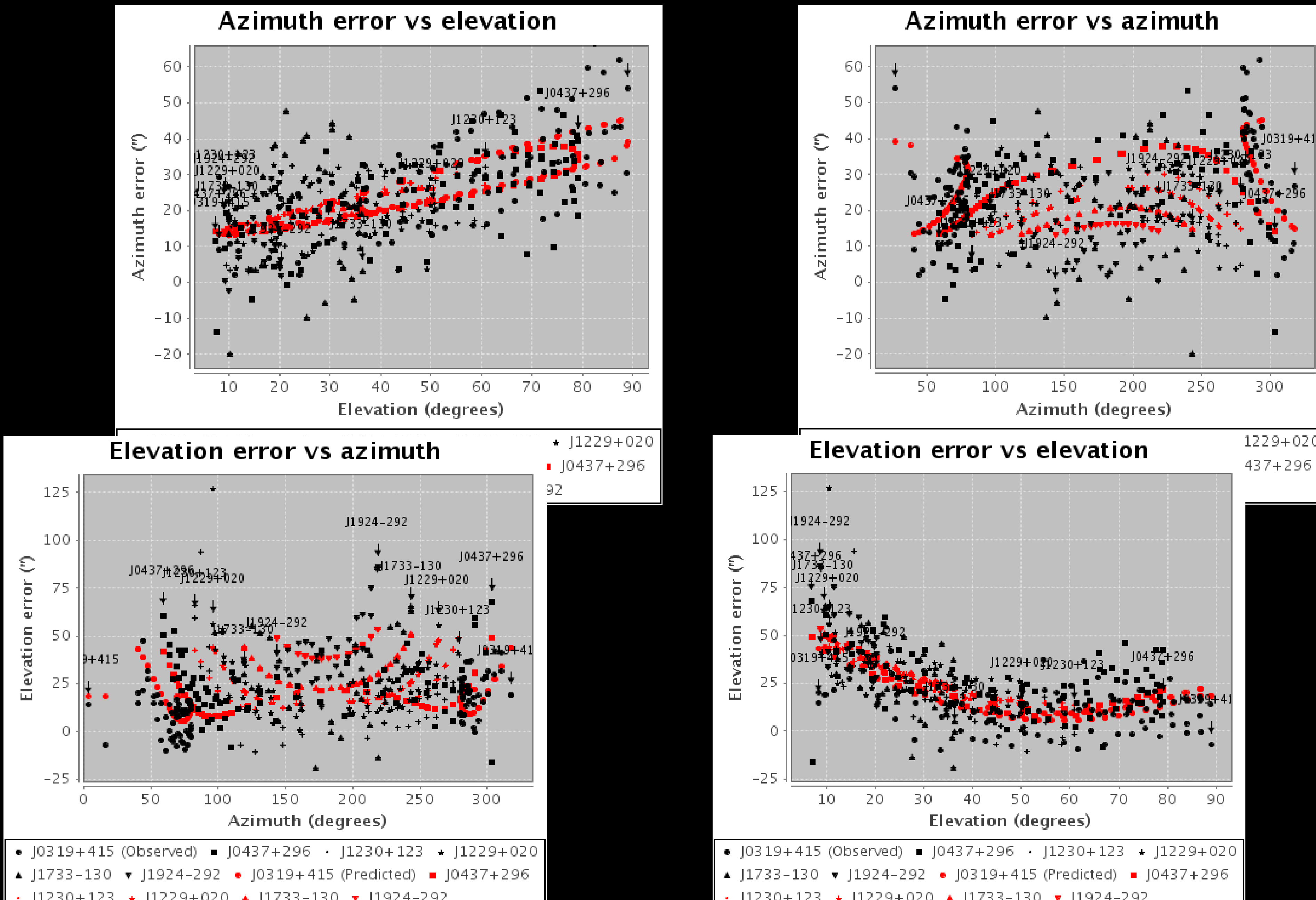
$$\delta Az = P_7 + P_4 \sin(Az) - P_5 \cos(Az) + P_8 \sin(El) + P_9 \cos(El)$$

- The antenna component uses a different model per receiver

The pointing model: analysis



The pointing model: analysis



Lessons learnt

- ACS has been a good choice: powerful and easy to use
- The github repository facilitates the upgrades
- The notification channel is an excellent service
- We have not explored other services: alarms

- Tables in the ACU are a simple & flexible solution

- MBFITS is a good choice, but not fully exploited
- FITS writer is fast but care is required when modifications

- Pyclass filler is a good tool for GILDAS.
- CLASS is an excellent option for the processed data

Radiotelescope Component Relationship

or

What's going on when you command a scan?