The 40m Radiotelescope Control System

Pablo de Vicente

Observatorio de Yebes - CDT (IGN)
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 network (a "component"). For example:
a component -> Spectral backend

Components talk to other components via ethernet

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

- Decision 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?

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

IDL file
Clients & components relationship

- Client
- Component
- Receiver
- Backends
- Weather St.

Languages:
- C++
- Java
- Python
- Corba
- Serial/GPIB

Connections:
- Client to Component via Corba
- Component to Receiver via Sockets
- Backends to Weather St.
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

Commands:

Remote Host → Commands → Acknowledgement → ACU

TCP Port: 9000

Status:

Remote Host ← Antenna status ← ACU

TCP Port: 9001

(T ~ 200 ms)
The **ACU ICD: Interface Control Document**

- Interface between the remote control and the ACU

2 Category of commands:

- **Mode commands**
  - Positioning
  - Pointing corrs.
  - Time source
  - Master setting

- **Trajectory & special commands**
  - Az/El Ra/Dec tracking tables
  - Az/El Offset tracking tables
  - M2 Offset tracking tables
  - M2 Offset elevation dependency

- Description of command parameters
- Description of status words and fields
Software layers: abstraction and encapsulation

ObsEng Component

Antenna Component

ACU Component

Client

ACU

Sockets

CORBA

CORBA

CORBA
Graphical clients (ACS generic)
Graphical clients (specific)
Commanding the telescope (example)

Ipython shell

Operator ID (NN)? PV
Observer ID (NN)? PV

RAEGE> status()
ML: 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/raegengr/raeg/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 velocity to 0.0.
Setting source velocity to 0.0.
Currently at Az=51.659 degs. El=16.899 degs.
For the time being offsets have to be in the same system as the source system. Command again the offset please.
(51.6586395371667, 16.89662957316657)

RAEGE> setTrajectoryParams(False, True)

RAEGE> on(00)
Beware! NO frontend is being used. Data acquisition not available
Setting dataAcquisition 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

(efer.date_time(year=2013, month=10, day=15, hour=8, minute=32, second=39, millisecond=195), efer.date_time(year=2013, month=10, day=15, hour=8, minute=33, second=42, millisecond=196))

RAEGE> point(3600, 'arcssec', 0.0, 'off', 10, 'no', 'x', 0)
Data acquisition not available
No data acquisition!

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

(efer.date_time(year=2013, month=10, day=15, hour=8, minute=34, second=56, millisecond=2), efer.date_time(year=2013, month=10, day=15, hour=8, minute=37, second=33, millisecond=31))
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

- Ephemeral Component
- Antenna Component
- Weather St. Not. channel
- Antenna Not. channel
- Scan Component
- Backends Not. channel
- Backend Component
- Frontend Component

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

- Coded in Python
- The pipeline generates GILDAS files: CLASS format
Pipeline (single dish)

Scan ends → Pipeline → Calibrates

PyFits

Reads FITS files

ON-OFF

pyclass

File.40m

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

\[ \delta \text{El} = P_1 + P_2 \sec (\text{El}) + P_3 \tan(\text{El}) - P_4 \cos(\text{Az}) \tan(\text{El}) + P_5 \sin(\text{Az}) \tan(\text{El}) \]

\[ \delta \text{Az} = P_7 + P_4 \sin(\text{Az}) - P_5 \cos(\text{Az}) + P_8 \sin(\text{El}) + P_9 \cos(\text{El}) \]

The antenna component uses a different model per receiver
The pointing model: analysis

<table>
<thead>
<tr>
<th>Model Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ ...</td>
<td>-7.534929367600877</td>
</tr>
<tr>
<td>□ ...</td>
<td>13.743845431903841</td>
</tr>
<tr>
<td>□ ...</td>
<td>-11.67202677976353</td>
</tr>
<tr>
<td>□ ...</td>
<td>-2.2345039030282554</td>
</tr>
<tr>
<td>□ ...</td>
<td>-3.4452958543532035</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter residual</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.245243268666615</td>
<td></td>
</tr>
</tbody>
</table>

Mean deviation

<table>
<thead>
<tr>
<th>Elevation vs azimuth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azimuth (degrees)</td>
</tr>
</tbody>
</table>

- J0319+415 (Observed)
- J0423-013
- J0437+296
- J1230+123
- J1239+020
- J1337-129
- DR21
- JUPITER
The pointing model: analysis

Azimuth error vs elevation

Azimuth error vs azimuth

Elevation error vs azimuth

Elevation error vs elevation
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

What's going on when you command a scan?

- Client
  - Commands
  - Creates Scan
    - Requests information
    - Database
      - Requests position
        - Ephemeris
          - Uses
            - Observing engine
              - Commands
                - Client
                  - FITS data writer
                    - Writes
                      - Calibrator GILDAS writer
                        - Reads
                          - Database
                            - Requests data
                              - Instruments
                                - Req. pos.
                                  - Antenna
                                    - Stops
                                      - Observing engine
                                        - Configures
                                          - starts/stops
                                            - Database
                                              - Writes
                                                - Database
                                                  - Requests information
                                                    - Scan
                                                      - Database
                                                        - Creates
                                                          - Ephemeris
                                                            - Uses