Highlights of Spanish Astrophysics VI, Proceedings of the IX Scientific Meeting of the Spanish Astronomical Society held on September 13 - 17, 2010, in Madrid, Spain. M. R. Zapatero Osorio et al. (eds.)

The control system of the 40 m radiotelescope

Pablo de Vicente[1], Rubén Bolaño[1], and Laura Barbas[1]

[1] Observatorio Astronómico Nacional (Instituto Geográfico Nacional)

Abstract

The 40 m radiotelescope of the *Observatorio Astronómico Nacional* saw first light in mid 2007 and started operations in 2008. Its remote control system has been designed, developed and implemented at the *Observatorio Astronómico Nacional*. This development started before the completion of the telescope and still continues nowadays, providing more and better features to the telescope while it is operated regularly for astronomical observations. In this article we review the software infrastructure selected (Atacama Large Millimeter Array software), the typical procedure followed to develop software to control and monitor the equipment and we present a global overview of the relationship between the different parts of the control system and how they interact during an astronomical observation.

1 Introduction

The *Observatorio Astronómico Nacional* (OAN) operates a 40 m radiotelescope located in Yebes, Spain since 2008. It is equipped with dual polarization cooled frontends at 2, 3.6, 5–6, 8 and 22–24 GHz and a single polarization receiver at 87 GHz. It can make continuum observations using several continuum detectors connected to an 8 channel analog to digital converter, and spectral observations with a Fast Fourier Transform spectrometer of 8 sections which can be configured in 4 different spectral confirgurations with instantaneous bandwidths of 100, 500, 750 and 1500 MHz and resolutions between 6 KHz and 180 KHz.

A radiotelescope is a complex system composed of an antenna, several receivers, continuum and spectral detectors and auxiliary equipment to aid the observations. Such panorama is the picture of a distributed system in which a central software coordinator must synchronize all the different devices that are used during the observations, and in which software components communicate among them, usually through a Local Area Network.

The first step in the development of the control system was to choose and install a communication software infrastructure which provided services and tools that ease the communication between the different parts. Later we developed software to monitor and control each individual device, to synchronize all operations between devices and to acquire and write the data from the backends. In the following sections we will describe the previous tasks.

2 The software infrastucture selection

During 2004 we searched for a state of the art communication software infrastructure which could fulfill the following requirements: It should handle a distributed system supporting C++ and Python as programming languages and be based on free software. It should be scalable and easy to maintain and install, and be well documented.

CORBA was the most obvious choice for interprocess communication because it allows the communication to be done independently of the location where the processes run. CORBA is also platform and language independent and there are several free software implementations. However its main drawback is its steep learning curve. This was crucial since our total developer time for the control system was less than 2 FTE.

In 2005 we adopted the Alma Common Software (ACS), in use for the ALMA interferometer and in other telescopes like APEX [7] and the HPT. ACS is based on CORBA but hides its complexity, implements the component container paradigm and provides services and tools to developers. ACS (see [1] for a description) supports C++, Java and Python. It is supported by a team of developers from ESO and NRAO and its lifetime is probably longer than that of the OAN radiotelescope (2025). Hence ACS is a project actively maintained and frequently upgraded. Since our adoption of ACS in 2005, ACS has also been chosen by other observatories [2].

ACS, is distributed both as a source package and, preferably, as a binary tar. It is only oficially supported under Linux Red Hat Entreprise but we have installed it on several Linux Debian versions after some local tuning. Since our first adoption of ACS 3.1 we have migrated to almost every new release and currently we use ACS 8.0.2 under Debian Lenny.

3 Software development in ACS: equipment

Physical devices (equipment and sensors) are associated to software objects living in the network, called components, and each physical device is mapped to an instance of a component. Each component exposes to the rest of the world, its methods, properties and characteristics so that other components can use them to communicate with it. These are described in a language neutral way using the Interface Definition Language (IDL). Components are managed by containers, which act as the interface between the outer world and the component.

Once a component is defined via the IDL, ACS creates a set of templates which are inherited by the implementation of each individual component.

The software development for instruments is usually done in C++, although we have written some components in Python and Java. Instruments can be connected to the Local Area Network, either via an ethernet card, or GPIB or serial ports. In the two later cases we use converters to ethernet or the devices are directly connected to a PC running Linux which acts as interface. The lowest level code is usually a non ACS class which commands and monitors the equipment. The next software layer is the implementation of the ACS component which uses the previous class via calls. In order to test the component, a client usually written in Python, is used. The client may be deployed in any host in the LAN which has ACS previously installed, regardless of the host where the component/container is running.

4 Special components

Two types of components associated to physical devices deserve a special mention: the frontend components and the Antenna Control Unit (ACU) component.

Frontend components run on a small equipment in the receiver cabin, close to the IF units since they manage analog and digital signals. This is achieved via embedded diskless PCs with Debian and a small version of ACS. The "driver" for the analog cards was developed at the OAN and the OS and ACS software was also installed in house.

The ACU [5] was provided by MT-Mechatronics as part of the servosystem complying all specifications provided by the OAN. It runs under two diskless Windows XP embedded cards with iTwinCAT, a real time extension from Beckhoff which uses an IRIG-B signal to keep the system in time with the UTC. The ACU commands the main axis motors (azimuth and elevation) and all motors related to the subreflector, the vertex, fans and the main mirrors in the receiver cabin.

The ACU is commanded from our component through sockets. All commands are synchronous and send an acknowledgment but they may take some time to complete. The status of the antenna is sent in a structure of variables through another socket each 200 ms. The trajectory generator is based on a combination of two tables: main tracking tables with time and position in the sky either in horizontal or equatorial coordinates, and superposition tables which add special movements on top of the former tables. These are also implemented via time and offsets in the sky either in horizontal or equatorial coordinates. This philosophy determines the design of the software for generating scans since no feedback from the backends or frontends can be used to control the movement of the telescope which is predetermined at the beginning of each scan.

5 High level components

There is also another kind of components not associated with equipment which can be considered high level or abstract components. These are used for generating the tracking tables from the observing patterns required by each observation, computing ephemeris, managing and storing scans, writing the data in FITs files, running a simple pipeline and synchronizing the whole observation.

To understand how all these components interact we will summarize the way a single dish observation is performed. Figure 1 shows the relationship between the different

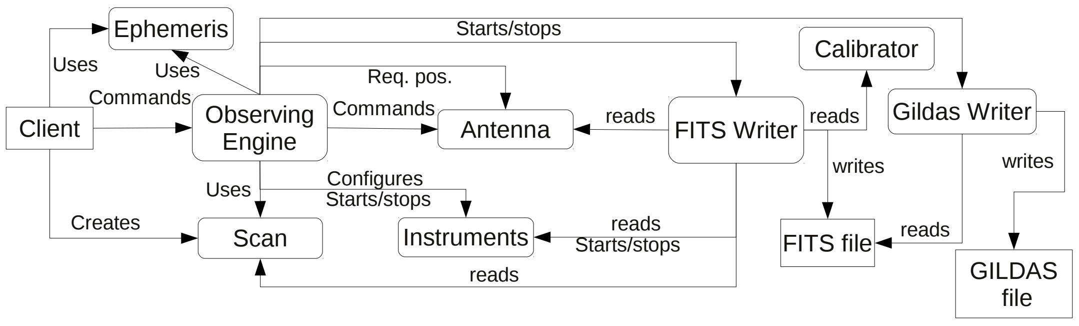

Figure 1: Relationship between the components during a single dish observation.

components on a typical single dish observation.

The astronomer uses an interactive command line Python client to create an observation on an astronomical source. The coordinates are computed in the ephemeris component for the current date and time. The frontend and backend combination and its setup (sky frequency, bandwidth, backend integration time, number of channels and frequency resolution) is defined and stored in the scan component, and a start command is sent to the observing engine. The observing engine component reads all the information from the scan, computes the coordinates, decides if the observation is possible either because the source is above the horizon or the antenna is iddle, and computes the time it will take to slew to the target. While the antenna slews, the observing engine sets the frontends and backends. Once the antenna is on source, and at the beginning of each subscan the observing engine starts the data acquisition. Every time a subscan is completed the observing engine stops the data acquistion. Data acquisition is done via two components which generate FITs files with a summary of the observation, monitor data from many sensors and the data. Once the scan is finished the observing engine starts the pipeline which produces continuum or spectral data compatible with GILDAS.

VLBI observations require a different process since they are managed by a third party program, called Field System, in use in almost all observatories of the world. We have developed a component that reads the information from the Field System and injects almost all commands related to the frontends and the antenna to the observing engine which acts as explained above. Data acquisition is directly handled by the Field System which uses the information provided by the observing engine on wether the antenna is on or off the source to tag the data.

The client used by the astronomer is an IPython shell with several features like history, autocompletion, syntax highlighting, and all tools available in Python which allow to write complex macros. This client retrieves updated information from Internet like coordinates of satellites orbiting the Earth or the latest DUT1 value published by the IERS.

The scan component, written, in Python generates the scan and subscan pattern, computes slew times between subscans, and stores internally all the information regarding the next observation. It contains two copies of the scan: the one being observed and the one being prepared. The latter is copied to the former when the observation is triggered. The scan information is divided in four sections: general information, source information, patterns to be used and frontend-backend setups. Currently the following patterns are available: on/off, pointing drifts, on and VLBI (simple tracking scans) scans, on the fly scans and raster scans. For the time being non-linear drifts, like spirals, have not been defined and customizable scans are not possible. The frontend-backend section is prepared for multibeam receivers since it stores information on the connections between horns and backend sections.

The ephemeris component uses an in-house library developed in C++. Apparent equatorial coordinates for solar system bodies are obtained from the JPL Planetary and Lunar Ephemerides DE405. It is also possible to compute the coordinates for satellites orbiting the Earth from its NORAD coordinates or the parameters provided by Intelsat. In both cases the updated parameters are automatically retrieved from Internet.

We have developed two different FITs writers. The one used in regular single dish observations is coded in C++, and uses NASA cfitsio library. We have adopted the MBFITS standard [6] which supports data from multibeam receivers. Each subscan is written in a different and separate file and each FITS file contains several HDUs. Two HDU, plus the primary HDU, hold generic information on the scan and on variables monitored regularly. Three more HDUs contain information per frontend-backend combination and one of these include the data. All HDUs, except the primary one, have binary tables. Data is written after each dump from the backends, and if the scan were aborted all data acquired up to then is recorded in the file. Data acquisition is done with two different philosophies: data is requested to the backend and the method blocks until the integration finishes, or the backends use a continuous cyclic integration mode, such that when data is ready it is pushed on a notification channel to which the FITS writer is suscribed. These two philosphies require two different strategies in the FITS writer in the component.

The second FITS writer was developed in Java, uses the nom.tam.fits library and it is only used in holography observations. FITS files are compliant with IRAM holography standard. Data from different subscans are written in a single file. Integrated data is kept in memory and written to disk after the subscan completes.

Amplitude calibration is done in a special scan in which the voltage from cuadratic detectors is measured. For frequencies equal or below 22 GHz, a noise diode is used. Calibration scans are composed of 5 scans in which the antenna measures the power off and on source with the noise diode on and off. A fifth subscan measures the offset of the backend once the signal from the receiver is disconnected. Observations at 87 GHz are performed measuring the power on hot and cold loads and on and off the source. The zero offset is also measured. Results from these calibration scans are supplied by an "amplitude calibration" component to regular scans and written in the monitor HDU of a regular FITS file. In this way all the information to calibrate a scan is in the FITS file and no third-party information is required.

A simple pipeline is available. This component has been developed using a python library independent of ACS which can be used as a standalone client. Continuum pointing observations, skydips and on/off spectral observations generate GILDAS compatible data from the FITS files using Python binding for CLASS. Spectra and pointing scans are generated after the scan is finished. Maps in the continuum are stored on simple ASCII files and processed later by GREG to create GDF images.

Graphical clients, written in Java, are available to help observers and operators. These clients monitor the antenna, frontends, backends, observation status and other sensors. A connection to KDE application KStars has been written so that the beam of the telescope is depicted on the sky together with the brightest radiosources. This has demonstrated to be a powerful tool to diagnose antenna problems. A web interface also shows the state of the observation allowing users out of the OAN to monitor its status.

6 Notes on the code

The control software has been written by an average of 2 FTEs since 2005, in a non-continuous effort. 0.5 FTE has been devoted since mid-2007, when the telescope saw its first light [5], to characterize the antenna at different frequencies while testing and debugging the software [3, 4]. The code is composed of more than 130 000 lines in C++, Java, Python, IDL, XML, HTML, PHP and MySQL, and it is licensed under the GPL. CVS is used as the versioning control system. All development has been done by personnel of the OAN.

References

[1] Chiozzi, G., Jeram, B., Sommer, H., Caproni, A., Pleskob, M., Sekoranjab, M., Zagar, K., Fugate, D.W., Di Marcantonio P., & Cirami, R. 2004, SPIE, 5496

[2] Chiozzi, G., Caproni, A., Jeram, B., Sommer, H., Wang, V., Plesko, M., Sekoranja, M., Zagar, K., Fugate, D. W., Harrington, S., Di Marcantonio, P., & Cirami, R. 2006, SPIE, 6274

[3] de Vicente, P. 2008, Informe Técnico OAN 2008-8

[4] de Vicente, P. 2010, Informe Técnico OAN 2010-10

[5] de Vicente, P., Bolaño, R., & Barbas, L. 2007, Informe Técnico OAN 2007-8

[6] Muders, D., Polehampton, E., & Hatchell, J. 2005, APEX Report APEX-MPI-IFD-0002, Rev. 1.57

[7] Muders, D., Hafok, H., Wyrowski, F., Polehampton, E., Belloche, A., Knig, C., Schaaf, R., Schuller, F., Hatchell, J., & van der Tak, F. 2006, A&A, 454, L25