Development of the control system for the 40m radiotelescope of the OAN using the Alma Common Software

P. de Vicente[1], R. Bolaño[1], L. Barbas[1]

[1]Observatorio Astronomico Nacional, Yebes, 19143 Guadalajara, Spain

Abstract

The Observatorio Astronómico Nacional (OAN) is building a 40m radiotelescope in its facilities in Yebes (Spain) which will be delivered by Spring 2006. The servosystem will be controlled by an ACU (Antenna Control Unit), a computer running Windows XP and TwinCAT, a real time extension developed by Beckhoff and provided by the company contracted for installing the antenna servo system. The ACU may be commanded from a remote computer or from two local panels.

The radiotelescope is an instrument composed of antenna, receivers, backends, and auxiliary equipment connected through a Local Area Network (LAN). Its control system has to deal with a distributed environment which needs to be remotely controlled and monitored from external heterogenous users (astronomers and engineers). We have chosen the Alma Common Software (ACS) framework because it fulfills the requirements of the control system for the 40m radiotelescope. The control system requires multiple processes simultaneously working and being synchronized. ACS provides an implementation of the component/container paradigm via Common Object Request Broker Architecture (CORBA) and also provides general purpose utility libraries, hiding the complexity of CORBA to the developer. ACS supports Python, C++ and Java, which allow the users to manage the radiotelescope using applications which run in different operative systems. ACS is supported by the European Southern Observatory (ESO) and the National Radioastronomy Observatory (NRAO) for the Atacama Large Millimeter Array (ALMA) with a lifetime similar to our radiotelescope. This is an important guarantee for the OAN with a very reduced software team.

We present an overview of the software architecture of the radiotelescope and the current status of the development of the components. We have grouped individual instruments with a common relationship in packages. Each instrument is controlled by an ACS component and the whole package is managed by one container. Each container tipically runs on a Linux computer. We have developed components which use in home developed libraries for using GPIB, serial, and ethernet ports which allow us to manage instruments via different hardware interfaces. We have also developed components to compute astronomical ephemeris. All our software is contributed with a LGPL license to the CVS contrib area of ESO.

Introduction

The Observatorio Astronómico Nacional (hereafter OAN) is building a 40m radiotelescope in its facilities in Yebes (Spain) which will operate between 2 and 110 GHz and which will be delivered in spring 2006. This telescope has been designed by the german company MAN Technologie and is being built by the spanish contractor Schwartz-Hautmont. It will be devoted to single dish and Very Long Baseline Interferometer (VLBI) observations with the European VLBI Network and the global VLBI network.

The antenna servosystem will be controlled by an antenna control unit (ACU) supplied by the german company BBH. The ACU is composed of two different CPUs which control the main axis of the antenna and the subreflector respectively. The main axis CPU will run Windows XP with a real time extension called TWINCAT developed by Beckhoff. The subreflector CPU will run VxWorks.

The ACU may be commanded by a remote computer using TCP connections through a local area network (LAN) and/or from two local control touch panels attached to the Windows CPU through

serial ports. The interface control for the ACU which is almost definitive has been jointly defined by the OAN and the BBH personnel and currently is being implemented by BBH.

The absolute time (UTC) will be kept in the ACU computer using TWINCAT. The OAN will provide an IRIG-B signal from a GPS receiver to the IRIG-B card in the ACU computer. This receiver will simultaneously synchronize other equipment using NTP (Network Time Protocol) across the LAN. This setup will allow the ACU to be synchronized with other equipment and to the UTC with an error of a few ms.

BBH has provided recently an antenna simulator with an 80% of its functionality which contains the local control panel software, the remote interface for the main axis of the antenna and large parts of the real-time control software including the antenna trajectory generator. This allows the OAN software team to develop the remote control software and partially test it against the simulator previous to the antenna delivery reducing the commisioning time.

Choice of ACS

The radiotelescope is a complex instrument composed of different equipment: antenna, receivers, backends and auxiliary devices (weather station, GPS receiver, temperature probes and radiofrequency synthesizers...) which can be controlled and monitored remotely through a LAN and which in some cases need to be synchronized in time with the UTC. Therefore it is a distributed real time environment. These equipment can be used by engineers and astronomers with a different philosophy. Engineers will tipically need to monitor and control individual equipment mainly for testing purposes. They usually use Windows computers and want to work remotely since in many cases the equipment will be located in unaccessible places and only be dismounted and taken to the laboratory in case of malfunctioning. Astronomers will use the radiotelescope as a whole, without knowledge of individual equipment, and need to control and monitor the telescope remotely. Astronomers usually use Linux and require an automatic operation mode which allows to make scheduled observations in absentia.

This scenario requires a system which fulfills these requirements. We believe that CORBA is a good choice to achieve this goal because it allows the communication to be done independently of the location where the processes run. CORBA is also platform and language independent and fits the heteregoneous needs of astronomers and engineers. However, CORBA has a steep learning curve and the remote control of the OAN 40m radiotelescope is being developed by a team of 3 people partially devoted to this work. There is not enough man power and time resources to learn new complex tools . Currently the total developer amount of time for the 40m radiotelescope control is 1.5 FTE.

ACS uses the component/container model via CORBA, provides general purpose utility libraries and tries to hide the complexity of CORBA to the developer reducing the time devoted to coding. It supports C++, Java and Python. ACS is supported by a team of developers from ESO and NRAO for the Atacama Large Millimeter Array (ALMA) and its lifetime matches that of the OAN radiotelescope. The first ACS public stable version (3.1) was available in October 2003 when the OAN began searching for software tools to develop the control of the 40m radiotelescope. The 40m radiotelescope is foreseen to last for 25 years, a similar period to that of ALMA. These facts guarantee that ACS will be regularly updated and maintained and that the OAN small software team gets a good suport from the ACS team. In the last year a non-ALMA ACS community is growing among other observatories which extends the support and eventually will allow the reuse of ACS software code reducing the amount of work for our team.

Experience with ACS at the OAN

We first tried ACS version 3.1 on November 2003. ACS is oficially supported on the RedHat Enterprise Linux distribution, however the standard Linux distribution in the OAN is Debian. The first task the OAN software team performed was to install ACS on Linux computers running Debian and use as many tools provided by the distribution as possible. The small tuning needed to install in Debian was reported back to the ACS team who modified some script files in later versions making

easier the installation on different Linux distributions. The installation instructions for ACS in Debian can be found on the ESO Alma Common Software Twiki and they are regularly updated to the latest Debian and ACS versions by one of the authors of this report.

ACS is compiled at the OAN on one Linux computer after a stable realase. We keep an updated copy of the latest stable ACS version from the ESO public CVS. Once compiled it is transferred to 6 Linux computers with the same distribution and packages as the first one. One of these computers runs the Manager and the ACS services. Development is performed in three different computers (one per team member) and when the code compiles and is functional it is commited to a local CVS repository. If we believe that the code may be useful for other users we commit the code to the public CVS contrib area in the ESO. All our code is licenced LGPL to keep it compatible with the ACS license.

Although ACS hides the complexity of CORBA, developers still need to devote much time to learn how to program under this environment. Not having followed a trainning course all our experience comes from self learning, reading documentation and unvaluable help from the ACS discussion users mailing list.

Up to now we have used the basic ACS functionalities: container/component model, DevIOs for accessing hardware, error system, notification channels and logging system. In the future as our needs and the complexity of the control system increases we intend to use the archiving system, the time system, the alarm system and the bulk data transfer system. We believe that a negative consequence of self learning and being a small software development team is that we will need to modifiy and eventually rewrite part of our code to include some of the ACS services that we do not use at this moment.

Typical developing strategy using the ACS

As already mentioned before, the radiotelescope is composed of several instruments. Each instrument or device is to be controlled by one ACS component. For each component that we develop we follow the same basic procedure:

1. create a directory structure for the component using the `getTemplate` ACS tool.
2. write a pure (non-ACS) C++ class which implements all the funcionality of the equipment and uses libraries to manage the hardware interfaces if necessary.
3. write an XML file with the exceptions we intend to throw.
4. include the ACS exceptions defined in the XML file in the pure C++ class.
5. create the IDL interface for the component defining its properties and methods.
6. run the HPT code generator which creates templates for the implementation files, the Makefile, the XML schema and the XML instance files.
7. develop the implementation files (header and cpp).
8. write the DevIO files which usually call the methods of the class defined in step 2.
9. Fill in the characteristics (maximum value, minimum value, default value, description, ...) of each property defined in the IDL interface and located in the XML instance file and place it in the central Component DataBase (CDB).
10. Define which container will manage this component.
11. Develop a graphical client.
12. Commmit the code to the local CVS server.

In some cases, like when using the notification channel we do not follow all these steps since, for example there is no need to define propeties.

This process, including testing and debugging may take a minimum of one week for simple components and may last several weeks for complex components like the ACU. The minimum time devoted for the component, not taking into account the graphical client may be of 3 days. Automatic generation of code is very important to reduce the coding time and programming errors.

Components are always developed in C++, and clients are written in Python or Java. For graphical clients in Python we use the Qt toolkit which is not included in the tools distributed by ACS but used

by other non-ALMA ACS teams. Qt Designer is an excellent tool for creating widgets and layouts.

Components are grouped in containers which manage its lifecycle and provide additional services. From the point of view of the developer there is no need to develop any code for the containers. We usually group components by relationship and create one container per computer, although sometimes we place several containers on one computer. For example the 40m radiotelescope will have an holography system which will be used to determine the quality of the surface of the antenna during commisioning. The holography system is composed of one downconverter, one radiofrequency synthesizer, two temperature probes, one continuum detector, and one FFT analyzer. Each device is implemented by one ACS component and all of them are managed by a single container running in a devoted computer.

The ACS logging system is a very useful system used in conjunction with "jlog" the logging client provided by ACS for debugging purposes. The generic client, "Object Explorer" is also an extremely useful tool for testing and debugging the component prior to developing a graphical client.

Overview of the software architecture for the radiotelescope

We intend to work in two consecutive stages, the testing stage and the production one. This will allow us to be able to make simple observations during the antenna comissioning. During the testing stage the radiotelescope will be commanded and monitored in an on-line mode; users will control in real time the antenna and equipment necessary for observations inside the OAN LAN. In the final production one, the users will be able to schedule and queue observations from outside the OAN LAN and it will not be necessary to be on-line while observing.

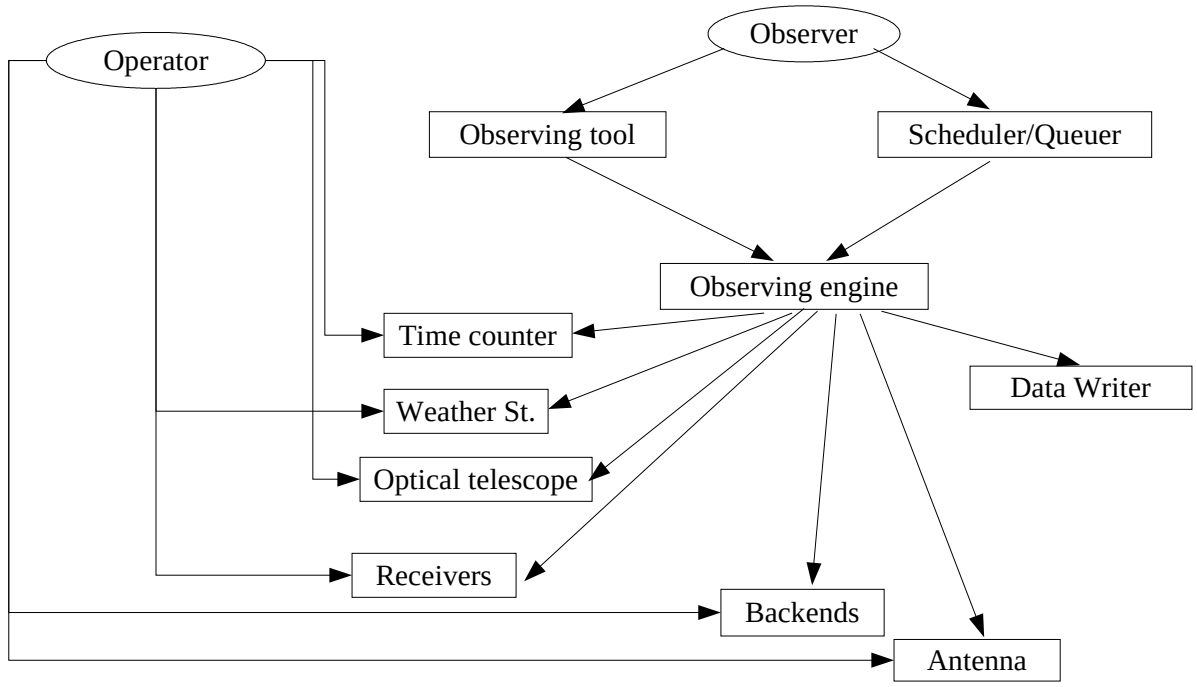

Figure 1. Overview of the foreseen architecture as viewed from the point of view of the users.

Figure 1 shows a simplified diagram of the control system architecture from the point of view of the users: the observer (astronomer) and the operator/engineer. The observer will be able to make scheduled or on-line observations using an observing tool. The observing engine will command and monitor the receivers, backends, antenna and the data acquisition and archiving. The operator and the engineers will be able to control directly the receivers, backends , antenna and auxiliary equipment.

The central module will be the observing engine implemented in Python. This component will connect to a generic antenna component which commands the antenna and monitors the azimuth and elevation and its errors. The observing engine will also connect to the ephemeris module to obtain the coordinates of the source to be observed and to the backends and receivers components and the FITS writer component. Using the azimuth and elevation errors and the UTC time it will synchronize the acquisition of data by the backends and the generation of a FITS data file. The observing client will be accessed by the user through a graphical observing tool in Java and a command line shell in Python.

The Antenna component has four properties to monitor the commanded azimuth and elevation, and the difference between the commanded and current azimuth and elevation, methods to track a source on the sky, and to perform several type of strokes while tracking it ('on-off', on the fly maps and raster maps). It will also provide a method to construct an observing table with a stepwise or continuous interpolation which allows to move the telescope in an arbitrary way around the tracked source.

We will develop a component common for all receivers. Currently we are defining a generic interface with properties and methods common to all of them. We plan to have 11 cooled receivers and one non-cooled receiver in the long term. Currently only the non-cooled receiver has been built. Receivers are to be controlled from PC104 CPUs running an embedded Linux version. We will try to install ACS on these CPUs to run a container which manages the receiver component. If this strategy is not possible we will run the component-container on another host and talk to these CPUs using the socket library developed in home.

We intend to develop components for two backends already built: a continuum detector and an autocorrelator. The third available backend is the VLBI one which uses an independent software system. In order to connect this system to the ACS we will eventually modifiy an ACS antenna client being developed by the Istituto di Radioastronomia (INAF) at Italy. This client will allow to command the antenna from the Field System.

Data from the backends will be transferred using the ACS Data bulk transfer system and written to the FITS format. We will posibly use the ACS FITS data writer from APEX which may need some modification to adapt it to our needs.

The observing tool using the command line in Python will also be adapted from APEX. No work has been performed in this direction up to now.

Other systems are the weather station, a Global Positioning System (GPS) receiver, the optical telescope used to test if the antenna tracks and points correctly, a time interval counter for comparing the maser signal to the GPS system and temperature probes in different places. All of them will be available to the Observing engine and provide important information for the observations.

Components developed with ACS

Most of the equipment used for the radiotelescope connect to the computers via serial, ethernet or GPIB ports. We have developed C++ libraries for each of these cases.

Since connection through serial ports usually requires one computer per two ports (unless using a multiport card), we have started to use comercial serial to ethernet converters. These converters are small, may managed remotely using the LAN and provide 4 or 8 serial ports. It is possible to address each serial port using the converter IP address followed by one port address. The control and monitoring is done using a socket library we have written for that purpose. These devices are specially useful for the holography system and the optical telescope since they are located on the subrflector of the antenna. Both systems require a total of 7 serial ports on a location which is usually unaccessible, moves constantly and exposed to extreme temperatures.

GPIB devices are controlled using the public binary drivers available for Linux from National Instruments. The GPIB devices are physically connected to PCs with PCI cards from National Instruments.

Up to now we have developed 7 components each for one different equipment: the weather station, a time interval counter, an holography downconverter, a frequency synthesizer for the holography

receiver, a continuum detector, an optical focuser for the optical telescope and the ACU. These components represent our starting development on the ACS and many more components should follow in the near future. These components have provided us insight to different services provided by the ACS. For example the weather stations uses the ACS notification channel to provide weather information to any client which suscribes to that channel.

The most complex component we have developed up to now is the ACU one. It uses the ACS error system, ACS logging system and ACS threads. Its IDL interface provides the same methods as those described in the Interface Control Document provided by BBH. The C++ pure class uses the socket library developed in home. We have recently started to develop a Java graphical client which simulates the console of the antenna and which should be an useful tool for the engineers.

We intend to distribute this tool and other Java clients using the Java Web Start technology which allows any user to install it on its computer without installing ACS previously.

ALMA uses astronomical libraries which are not GPL licensed. We have developed one module made up of three components for computing astronomical ephemeris. An AstroTime component computes time ephemeris and provides methods for conversions between calendar and time to and from julian day and to and from Greenwich sidereal time. The AstroLocation component provides properties and methods to convert from geocentric coordinates to geodetic ones on the surface of the Earth. This component also uses the AstroTime component to obtain the local sideral time on a given location on the surface of the Earth. The AstroSource component provides properties for storing the coordinates of astronomical sources and methods to compute mean coordinates, apparent coordinates, convert from different reference systems: galactic to/from equatorial, horizontal to/from equatorial and compute the radial velocity of the sources referred to different reference systems. The AstroSource component uses the AstroTime and AstroLocation components internally. This module has been checked against the JPL ephemeris available in Internet using a graphical PyQt client.

Status of the control system and future tasks

The control system for the 40m radiotelescope is at the present time in an early development stage. Currently only some instruments can be controlled and monitored remotely.

We have completed the ACU component which should move and monitor the antenna mount but it has only be tested against the simulator and it may suffer a partial rewrite yet. A Java client is being written. Currently the generic antenna component which hides the ACU complexity and uses the ACU component internally is being developed.

In the near future we will develop components for the optical telescope system composed of a remote focuser, a CCD camara, and a remote cover. This system has priority because it will be used in the commisioning of the 40m radiotelescope before the receivers are available.

The first available receiver will be a non-cooled one for holography observations for which we have already developed some components. Our main effort in the near future will be devoted to the transfer of data from the continuum detector to the FITS writer. Components for other receivers will follow later.

The component to control and monitor the autocorrelator backend and the observing tool will be delayed until other components are finished. We intend to reuse code from other projects like APEX and the INAF modifying the code to fit our needs.

References

[1] G. Chiozzi. ALMA Common Software: a developer friendly CORBA based framework. Paper 5496-23. Glasgow, Scotland: SPIE, 2004.

[2] G. Chiozzi. ALMA Common Software (ACS): status and developments. Geneva, Switzerland: ICALEPCS, 2005.