An Amateur Telescope Control System: Towards a Generic Telescope Control Model

Rodrigo J. Tobar\textsuperscript{a}, Horst H. von Brand\textsuperscript{a}, Mauricio A. Araya\textsuperscript{a}, Joao S. López\textsuperscript{a}

\textsuperscript{a}Departamento de Informática, Universidad Técnica Federico Santa María, España Avenue 1680, Valparaíso, Chile

ABSTRACT

Control System for an Amateur Telescope (CSAT) is a distributed telescope control system model for amateur telescopes with transparent interchangeable components, built using the ALMA Common Software (ACS) framework. The CSAT project has been thought as the first step towards a generic telescope control model, which will consist on a generic control framework for any telescope mount. With the ACS Container/Component model, a completely different hardware can be supported by just re-implementing the low-level components for the new setup. This way, CSAT becomes a very good example of all the features that ACS provides for building a generic telescope control framework.

Keywords: CORBA, ACS, TCS, Amateur telescope, Distributed systems

1. INTRODUCTION

Control System for an Amateur Telescope (CSAT) is a telescope control system developed by the ACS-UTFSM Group and designed for working with amateur telescopes, written over ACS, the ALMA Common Software. ALMA, in turn, stands for Atacama Large Millimeter Array and consists on an array of about 50 antennas working together to make observations of objects in the sky at millimeter/submillimeter wavelengths. ACS is the software platform that will serve as the base for the control of all the hardware in ALMA. It consists of a CORBA-based distributed system designed for the generic control of any kind of hardware. It uses the Component/Container model for the management of the software pieces. It handles three different programming languages (i.e., C++, Java and Python) for writing applications over it. The CSAT project has been thought as a first experience for the definition of the gTCS, a generic Telescope Control System framework that could be used to control any kind of telescope regardless its mount. The idea is that any telescope mount could use the gTCS just by writing the device specifics to connect the telescope to the TCS, without reimplementing the TCS common code.

In this document we present our current work on the CSAT project. We start presenting a general system description in section 2. Within this description, we show the system architecture, the control model and further internals information. In section 3 we describe an external applications wrapper developed in CSAT. Section 4 presents how this work contributes to the advance of the gTCS project, and which are the main features that make CSAT a good approach to what the gTCS pretends to be. Finally, section 5 contains the most important conclusions of this work.

Further author information: (Send correspondence to R.T.C)
R.T.C.: E-mail: rtobar@csrg.inf.utfsm.cl, Telephone: +56 32 2654562
H.v.B.: E-mail: vonbrand@inf.utfsm.cl, Telephone: +56 32 2654239
M.A.A.: E-mail: maray@csrg.inf.utfsm.cl, Telephone: +56 32 2654562
J.S.L.: E-mail: jslopez@csrg.inf.utfsm.cl, Telephone: +56 32 2654562
2. SYSTEM DESCRIPTION

CSAT is built over ACS. Therefore, it uses its Container/Component model to develop and deploy the software pieces that compound the system. Containers hosting components offer them several ACS services, and allow the programmer to forget about the CORBA encapsulation and just concentrate on the logic of the software. This CORBA encapsulation allows, also, the usage of remote objects over the network as it would be on the same process. Therefore, CSAT is constructed out of several ACS Components that interact over a network, or in the same machine, allowing to control an amateur telescope in a distributed environment with no complications.

The latest versions of ACS officially run on Scientific Linux 4. This distribution has an old kernel, and newer machines are not well supported on it, as well as many devices. An effort has been made by the “Repackaging of ACS for Embedded System” (RAES) project at the ACS-UTFSM Group to have the latest versions of ACS (i.e., versions 6 and 7) working on newer Linux Distributions,5 compiling them with the newest versions of the gcc compiler. CSAT is currently using the 6.0.4 version of ACS compiled with gcc version 4. Thanks to this, CSAT has been able to run over Fedora 6, Ubuntu 7.10 and 8.04 and Debian 4.0, with the newest kernels, giving us support to use, for example, the Meade LPI CCD, which could not be used on Scientific Linux 4.

Even when CSAT runs under Linux, there is a software piece developed on CSAT that makes it able to interact with many other operating systems. This will be detailed in section 3, where the external applications wrapper is detailed.

2.1 General architecture

CSAT architecture is composed by four software layers. Each layer has been thought to work only with the components placed in the present and below layers, being totally transparent to them which are the implementations of the components of the layers above them. This way we implement the components with only the tasks that they are supposed to be responsible of, leaving the rest to upper level components (see figure 1).

The very first layer of CSAT is the hardware access layer. Here is where the DevIOs6 are placed, being the base of the whole system. A DevIO is a special class that contains the code to read (and eventually write) a certain physical property of a device. Their implementation are totally independent of the rest of CSAT, and their only responsibility is to access the devices. In the same layer, creating, managing and destroying the DevIOs, are the DevTelescope, DevCCD and DevGPS components, which represent the physical devices corresponding to their names. Their only function is to hold inside them all the DevIOs that access a physical property of its corresponding device, representing finally a physical device with several physical properties, and where certain instructions can be given to it.

The second layer is the first one with intelligence in it, and it is composed of two main components types. The first one corresponds to all the components that do control over one of the Dev* components. For instance, the Telescope does control over the DevTelescope component (more details in subsection 2.2). This way, each component of this type contains one associated Dev* component, and controls it. The second type of component present at this layer are the logical components, containing the logic of the application on them. For instance, here are the Tracking and Pointing components.

The third layer is compound by two components: CSATStatus and CSATControl. These two components are the entry point for external applications, being the only interface to operate the whole system. The first component contains only “read-only” operations, like querying the current telescope coordinates, or the sidereal time of the system. The second one is intended to send commands to the telescope control system, like asking the telescope to go to a right ascension/declination position, or setting the tracking on/off. Hence, these two components manage references to almost all the components of the layers below, sending commands and reading responses from them.

Finally, the fourth layer is compound by all the clients that the system may have. At the time of writing, two different clients have been developed. The first one is Hevelius,7 which is an easy-to-use GUI designed for teaching purposes. The second client is a wrapper for external applications. This wrapper is explained in further detail in section 3. ACS provides built-in GUIs to access the functional interface of a component (such as the Object Explorer), therefore there is no need for an specific mainntaince or engineering GUI. Anyway, the ACS Object Explorer is also a valid client of CSAT.
Figure 1. Architecture of CSAT. There are four software layers, where the lower ones are not aware of the upper ones. The first layer is the Dev box. The second layer corresponds to the lower part (it is divided by a pointed line) of the amTCS box. The third layer consists on the upper part of the amTCS box. Finally, the last layer corresponds to the clients.

2.2 Control

Since CSAT is a first approach to the gTCS, one of the goals on its design was to be able to control different amateur telescopes with it, maintaining the same logic for all of them, while only changing the telescope device specifics. To meet this requirement, a generic interface has been written, representing a telescope device. This interface provides methods for controlling the speed of the telescope, while giving the opportunity of reading its current position. This information is used then by a higher level component responsible of the control itself of the device, but independent of which telescope it is controlling. This control structure (figure 2) works by setting a velocity on each telescope axis depending on the difference between the coordinates where the telescope is placed and the coordinates that is commanded to the telescope control system.

Here is the control structure: First of all, there is a DevTelescope interface that provides the definition of the methods that need to be implemented to communicate with the telescope. After the compilation of the IDL file for this interface, the DevTelescope abstract class is created. This abstract C++ class contains within it four ACS Properties, which are special classes representing hardware physical properties, and that are accessed for read/write operations through their corresponding DevIO. The DevTelescope class is inherited by NexstarImpl and lx200Impl, two different implementations that we have developed for the Celestron Nexstar 4 SE and a Meade LX200 telescopes, respectively. This allows us to use the default DevTelescope object from the Telescope
component, independent of its implementation. A given `DevTelescope` implementation can be set as default by writing the proper configuration into the ACS Configuration DataBase (ACS CDB). Once we have defined which is our default implementation, it is easy to find it (and use it) through ACS, regardless the language we are working with. If at a given point in time we want to change our default implementation for a given interface, it can be done by assigning the new implementation as the default one on the CDB, and reloading the CDB DAL with the new information. With this control schema we are able to have interchangeable components that obey to the same interface, having only one control mechanism over all of them.

Note that this control schema is not generic enough. For example, it cannot be applied to equatorial mounted telescopes, since all the structures and values refer to velocities on the altitude and azimuth axes, and the control loop shown in `Telescope` works only for this mount type. In order to have a more generic control over the telescopes, and to include equatorial telescopes as well in the presented control schema, three major changes need to be done to extend the presented control structure. The first change is just a rename of all the axis-related variables. This involved the change of all the `alt*` and `azm*` variables to `axis1*` and `axis2*` (an alternative could be `horizAxis*` and `vertAxis*`). In the case that both axes are referenced by the variable name, then `axis` should do it. This is important since they will represent coordinates or velocities on each axis, independent the telescope mount. The second necessary change is to have two new interfaces that inherit the `Telescope` interface: `EquatorialTelescope` and `AltazTelescope`. These two interfaces will have each an implementation, where the control loop will work with velocities on the corresponding axes. Finally, the third change is the inclusion of a `mount` method on the `DevTelescope` interface, which should return, on each implementation, the mount type of the telescope. With this new control structure (figure 3), when a telescope is plugged into the system, and the corresponding ACS component is activated, CSAT will know its mount type and will put the corresponding control component into work.

With these changes, CSAT allows to control horizontal and equatorial mounted telescopes. This covers a wide range of the existing telescopes today. Anyways, this control process cannot be applied, for example, to the HexaPod Telescope (HPT), a telescope mounted over 6 legs, since CSAT is clearly designed to control 2-axis mounted telescopes. Anyways, this problem is not covered by CSAT since it is aimed to amateur telescopes, which are mostly 2-axis mounted, and therefore can be controlled by it.

The two explained control mechanisms are already present in CSAT, in different source code branches at the time of writing. They have been successfully tested with several telescopes (see section 4) with both mounts (Equatorial and altitude/azimuth) with very good results.
2.3 Coordinate systems

CSAT has three different coordinate systems implicit in it. The first two coordinate systems are the altitude/azimuth and right-ascension/declination real word coordinate systems. These coordinate systems are used over the whole control system and by the external applications. Anyways, the telescope device does not necessarily know about these two coordinate systems, and in the worst case, the device will only understand its custom encoder units. Therefore, the responsibility to translate these encoder counts into useful information rests on its DevIO. Since the information that we need from the telescope is its actual position in both axes, then the encoder counts are translated into how many degrees the telescope is rotated on each axis. These rotations conform the telescope self coordinate system, which is the third coordinate system used in CSAT.

How the telescope coordinate system is translated into real-world coordinate systems depends on which mount type the telescope has. In the case of an altitude/azimuth mount, the telescope self coordinates correspond exactly to the altitude/azimuth coordinates, as the horizontal axis rotation corresponds to the azimuth coordinate, and the vertical axis rotation corresponds to the altitude coordinate. In the case of equatorial mounts, we must consider the following: The zero point for the vertical axis is declination zero, and the zero point for the horizontal axis is the sidereal time right ascension. With this taken into account, the vertical axis rotation of the telescope corresponds to the declination coordinate (if we are in the south hemisphere it must be inverted), while the horizontal axis corresponds to the right ascension coordinate minus the sidereal time (if we are at the north hemisphere, we must invert this coordinate).

The three coordinate system are not used by the whole system. First of all, and as we said before, if only
encoder counts can be retrieved from the telescope, then its DevIO will manage this information and translate it into telescope coordinates. If direct telescope coordinates can be retrieved from the telescope, then there is no need for the DevIO to translate the value, but only to retrieve them. The only component that makes use of this information is Telescope. This component is in charge of taking the telescope position in its coordinate system and translate this position to real-world coordinates as we explained before (see figure 4).

Figure 4. The three different coordinate systems used in CSAT. The telescope component is the most critical one, since it manages the three of them within it. The DevIOs that access the telescope might have to translate encoder counts into the telescope coordinate system if necessary.

3. EXTERNAL CLIENTS WRAPPER

As said in subsection 2.1, CSAT counts with two special ACS components. These two components (CSATControl and CSATStatus) communicate the clients of the system with the rest of the components, so the clients need to handle only the references of these two mentioned components.

We have developed two different CSAT clients. The first one is Hevelius, which consists on a Java GUI program thought for teaching purposes. This client connects to CSAT and sends commands to it, while reading its status. The idea under Hevelius was to have a simple-to-use client for inexperienced people. It includes, among other things, a 3D telescope model, a weather panel and a CCD image panel.

The second client that we have developed is a wrapper for external applications that actually connect to a telescope (e.g., kstars or stellarium). By external applications we understand all the software that is outside of the scope of CSAT. The idea of this wrapper is to reuse all the possible code by means of replacing the “controlled telescope” of the external application by our wrapper. The wrapper, then, listens for all the commands that are sent to the telescope by the external application, and it translates into CSAT’s commands. It executes the command on CSAT, and finally it sends the response to the external client. In summary, the wrapper behaves like a telescope to the external application, while like a client to CSAT.

The communication medium between the wrapper and the external applications is a serial line. If the external application and the wrapper are running in the same computer, then the serial line corresponds to a master/slave pseudo-tty pair, where the external application writes and reads the commands on the slave device, while the wrapper reads/writes them from the master one. If the external client is on a separated computer, then the serial lines corresponds to a physical cable connected between the serial ports of both computers. A graphical representation of this schema is presented in figure 5.
Since the idea of the wrapper is to be able to connect not to one, but to many external applications, the support of various telescope models is a requirement. Therefore, the wrapper has been constructed in such a way that it can be launched by specifying which telescope model it should simulate. At the time of writing, both Nexstar and LX200 telescopes have been implemented and tested with the \textit{kstars} application.

It is important to stress the main aspect of the wrapper. The external applications will always see the telescope model that it controls. Regardless of this telescope model, CSAT may be controlling any other telescope, with any other mount, and the external application will not note it, since it will see always the same telescope. Thanks to this, if we already have a wrapper for a given application, the only thing that we need if we want to use this external application with it is to write the \texttt{DevIOs} to access the telescope, along with the \texttt{DevTelescope} interface implementation for it. Once done, we will be able to control the telescope with any of the supported external applications without problems.

This wrapper represents an enormous gain and can have a wide application spectrum. For example, let say that an educational institution owns a telescope, and that it already has a telescope control system that controls it and a GUI application that the students know how to use and run. If they are used to it, and a new telescope comes to the institution, CSAT could be plugged into it and, by only writing the proper wrapper support for the old telescope, the old GUI can be used, so there is no need of learning how to use a new system. In general, the wrapper gives us the advantage of reusing all the possible code of many telescope controller applications, regardless of their operating system, their programming language, or its GUI look-and-feel.

\section*{4. TOWARDS THE $gTCS$}

CSAT has been thought as a first exercise for the construction of the $gTCS$. The architecture of interchangeable components is the most relevant aspect of CSAT, and is the main idea behind the $gTCS$. Hence, if CSAT works
properly using this architecture with different telescopes, then it should be a good sign that it is possible to control different telescopes with the same code.

CSAT has been tested with several telescopes. The initial telescope that we had to do all the tests was a Celestron Nexstar 4 SE, horizontally mounted. We wrote the access code for the telescope, and, afterwards we developed the telescope control code. A Nexstar 4 SE simulator has been developed as well, and is also being used through CSAT. A summer job in the Santa Martina Observatory of the Pontificia Universidad Católica de Chile added support for the Meade LX200 GPS horizontal-mounted telescope for CSAT. After this support was added, it was successfully tested on the same observatory. The support of the Nexstar telescope has been also tested at the UTFSM. Two equatorial telescopes mounts have also been added to CSAT, which correspond to the same Nexstar and LX200, but with equatorial mounts. The Nexstar was used to test the implementation of the control for equatorial telescopes, while a visit to the Observatorio Cerro Armazones (OCA) allowed us to test CSAT with an equatorial mount LX200.

Nevertheless, not all the software tests have been successful. At Cerro Armazones, we tried to test our software on a 84 [cm] telescope without success, since we didn’t have direct access to the encoder board and the servo motor amplifier board. Also, the available communication methods with the computer hosting these boards were not useful for CSAT, so clearly the architecture of CSAT should be improved towards the gTCS. Then, the gTCS architecture should develop a more flexible communication layer, giving the opportunity to make the control at low level, or at an upper levels; this is, instead of making the efforts to access directly to the necessary hardware, offer the opportunity of sending a target object to the hardware instead of velocities.

These CSAT experiences and tests, including the unsuccessful testing over the OCA 84 [cm] telescope, are just the first steps towards a generic telescope control system. The main problem of the gTCS project is the definition of generic interfaces to represent a variety of devices. We believe that the usage of inheritance is very important to achieve a good design of the gTCS. The APEX project has already tried to do some work in interface generalization, designing their interfaces as generic as possible. The HPT project contains also reused code from the ALMA project for its telescope mount, which indicates the big necessity (and feasibility) of reusing code between observatories. It also has aimed to the generalization of a device by its hptCCD implementation through a CCD factory.

The gTCS project is already running. Conversations, meetings, discussions and idea recollection are already being held. A workshop at the UTFSM is being prepared which will be dedicated exclusively for this topic, inviting software engineers from different observatories in Chile, in order to have different points of view, different experiences, and several groups working around the gTCS idea.

5. CONCLUSIONS

This work has been a very effective step for introducing us to a telescope control system. CSAT has been a first step to the gTCS, and its capability to control different telescopes with the same control structure, and without altering the software components above the telescope communication, is the main characteristic that must be rescued from it.

Our recent tests with CSAT have been of great importance for its development, since they have provided us a valuable feedback on our work. We think that it is of high level importance for the gTCS’ development the in-site testing of the produced software. CSAT has been a good example of this, since the laboratory development of it is not enough to corroborate how work some parts of the system (as the tracking). Because of this, the gTCS needs to be tested periodically in a set of representative telescopes in order to really have a generic control over them.

This set of telescopes can be already defined: the Nexstar telescope will obviously be one of them; Santa Martina’s LX200 can be used for testing, as well as the ESO 50 [cm] located in the same observatory (which is not working yet, but we hope it will soon). Telescopes located at Cerro Armazones may be used as well. With these (and maybe more) telescopes we hope to develop a good start point for the gTCS. It is important to note that it is impossible for us to really produce a software piece that could interact automatically with all the telescope around the world, but a good approach can be done by controlling this set of standard telescopes. If we probe that the system is generic enough to control this set of telescopes, then this will be a clear evidence
that extending it into more telescope models should not be a hard task, and that the system should be capable to control these new models without problems.

The gTCS is a very ambitious idea towards software reuse in astronomical projects. The general trend in this area is covered by Chiozi et al. on these proceedings. The gTCS project, even when it is being initialized by the ACS-UTFSM Group, and its first developments are going to be done by us, is a project that belongs to the whole community, and that must be kept alive by the collaborations between different observatories and the contributions of people all around the world.

ACKNOWLEDGMENTS

This work is supported by project ALMA-CONICYT 31060008 “Software Development for ALMA: Building Up Expertise to Meet ALMA Software Requirements withing a Chilean University”. The development over the Meade LX200 GPS telescope on the Santa Martina Observatory could not have been done without the collaboration of the Pontificia Universidad Católica. The development over the Equatorial Meade LX200 and the testing over the 84 cm telescope on Cerro Armazones could not have been done without the collaboration of the Universidad Católica del Norte and Roland Lemke, who has also given us support with his participation on the gTCS discussions. Jorge Ibsen has been of great help during the whole process.

REFERENCES