CSAT Deployment and Final Status
Rodrigo J. Tobar, Joao S. López, Mauricio A. Araya, Horst H. von Brand

Abstract

Control System for an Amateur Telescope (CSAT) is a Telescope Control System (TCS) designed to operate amateur telescopes. It is based on the ALMA Common Software (ACS), taking advantages of its Component/Component model, generic control design and CORBA encapsulation. CSAT is the first step towards a generic Telescope Control System, the gTCS, a framework capable to control any telescope, regardless its mount. Because of this, CSAT was designed with such a generality for amateur telescopes. Different deployments have been done for CSAT, testing its performance on various different telescope models. In this work we present CSAT, its main components, and our experiences on the usage of it on different scenarios.

1 Introduction

Control System for an Amateur Telescope (CSAT) is a Telescope Control System (TCS) designed to operate amateur telescopes, and built over the ALMA Common Software (ACS) [3, 2]. ACS is a CORBA-based distributed framework designed for controlling any kind of device in a generic fashion. Its Container/Component model [7, 8] eases the development of ACS components, encapsulating the CORBA communication inside the containers, and writing the logic of the problem in the components. CSAT, then, is compound by several ACS components that communicate between them through CORBA and CORBA services, to control an amateur telescope in a distributed and heterogeneous fashion.

The CSAT project was a first step towards a generic telescope control system [9], the gTCS. The idea behind the gTCS is to provide a framework with the ability to control any telescope, regardless its mount, in a generic way, and only writing the code needed to communicate the telescope with the system. CSAT provides some generality on its design, and so it helps as a first hand experience towards the gTCS.

The rest of this work is organized as follows: section 2 explains what is CSAT, which are its main components, and which is its final status. Section 3 shows the main deployments of CSAT on several environments, and which were the main conclusions for each of them. Future work on the CSAT project and its continuation, the gTCS project, is shown in section 4. Finally, section 5 illustrates the main conclusions of our work.

2 What is CSAT?

CSAT is a Telescope Control System. A Telescope Control System consists, mainly, on a piece of software that is able to control the mount of a certain telescope, being able to move the telescope to a given position, for example, or to move an axis at certain velocity. CSAT is aimed to control amateur telescopes, with simple interfaces, and not used for professional astronomy. Such telescopes may or may not contain a control logic inside of them. Then, and in order to be able to control them with CSAT’s intelligence, this embedded control logic should be disabled when needed.

CSAT is build over ACS, the ALMA Common Software. Therefore, it uses the Container/Component model of ACS to develop the software pieces. The distribution of the components through a network of computers is then totally transparent to the programmer, who only has to care about the logic of the component piece.

2.1 CSAT 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 compo-
nants 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).

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.

The very fist layer of CSAT is the hardware access layer. Here is where the DevIOs [5] 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 composed 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 composed 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 [4], 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 2.3. 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 maintenance or engineering GUI. Anyway, the ACS Object Explorer is also a valid client to CSAT.

2.2 Control Design

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 [5]. 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) [10]. 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 mayor 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) [6], 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 3) with both mounts (Equatorial and altitude/azimuth) with very good results.

2.3 External applications 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 4.

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 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 DevICs to access the telescope, along with the 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.
Figure 4: External client wrapper. The wrapper acts, to the external clients, as a telescope, while to CSAT as a client. Note that the wrapper can listen to clients located on the same machine as well as clients located on a second PC, regardless of its operating system.

2.4 CSAT Final Status

CSAT project is now finished. Its original purpose of being the predecessor of the gTCS project has been achieved, providing a first hand example on the usage of ACS for the building of such a system. This project, anyways, is not only for us, but for the community.

The actual state of CSAT makes it usable for a wide range of devices. Home-made telescopes that do not have any TCS could take advantage of the generic structure of CSAT and be controlled by it. Commercial amateur telescopes not supported by several astronomical applications can be plugged to CSAT, and the external applications wrapper can be used to control it, avoiding the development of a whole new GUI client for the new telescope mode. Besides these characteristic, CSAT offers, in its actual state:

- Control for Altitude/Azimuth mounted telescopes, as well as Equatorial mounted ones. The control schema is based on reading the current position, and setting velocities on both axis to reach a desired commanded position.
- Basic manual pointing model. This pointing model improves the observation precision when the telescope is not properly aligned.
- Tracking capabilities. Tracking, as its name indicates, is the ability to track the position of an object on the sky, compensating the Earth movement.
- Geographic information configuration. CSAT will perform all the calculations based on this information, so no other change will be needed.
- Current support for Nexstar 4 SE, Meade LX200 GPS and Meade LX200 EMC telescopes. Future support can be added easily by implementing the same operational DevTelescope interface.
- Hevelius GUI client for CSAT. Hevelius is a Java GUI client aimed for amateur observations, and with teaching purposes on the astronomical field.
- External applications wrapper, with the ability to simulate a Made LX200 basic telescope, and a Celestron Nexstar telescope.
• A simulator implementation in Java for the Celestron 4 SE telescope. This simulator can be used as a stand alone application, as well as an ACS component. This simulator helps to develop rapidly without the need of a physical telescope in place connected to a PC.

3 Experiences

3.1 Deployment methodology

Thanks to its architecture, CSAT can be easily extended in order to support a new telescope model. The DevTelescope interface and the Telescope component implementation allow us to use any implementation of the former in a transparent form through the last. The generic design of ACS also help us to encapsulate the communication code in the DevIO’s, making them reusable and useful for future work. With all these concepts in mind, the steps that must be given to deploy CSAT on new telescopes have been:

1. Study the communication protocol with the new telescope. In the three experiences that we have had, it corresponded to a set of commands sent through the serial RS232 interface, with a specific configuration.
2. In order to write some testing code, we developed simple test clients. This gave us an early feedback on our knowledge of the different protocols, refurbishing it when needed. At the end of this process, a full-operating communication code had been written.
3. Once the simple client was finished, we took the communication code from it and encapsulated it on the corresponding DevIO’s.
4. Finally, a DevTelescope interface implementation had to be written, using the corresponding DevIO’s.

The rest of the code (from the Telescope component and upper) did not need any change, theoretically. Our empirical experiments, anyways, shown that debugging was needed to have a generic code on the upper layers of the software.

3.2 UTFSM Deployment

CSAT initial development was totally done over a Celestron Nexstar 4 SE telescope. This is a widely used telescope that offers a serial RS232 interface to communicate with it with a PC. Thanks to this, its protocol has been implemented on several open-source astronomical software, like kstars and stellium. This allowed us to have a rapid development of the test client, in union of the corresponding DevIO’s and the component implementation. The first versions of this component and DevIO’s were only valid for altitude/azimuth mounted telescopes. Nevertheless, the equatorial mounted version of the telescope is actually supported as well.

In addition to this, a simulator of the Nexstar 4 SE telescope was developed. This simulator can be deployed as an ACS component, or as a standalone application, being able to listen commands through the RS232 port. When used in the first mode, it had to be accessed through ACS. For this we also developed an implementation of the DevTelescope interface that interacts with this ACS component through the corresponding DevIO’s. For the second mode, we just use the implementation for the real telescope. This way, we have a telescope simulator available, which can be run in the same machine, or in a separated one if needed.

On the other hand, we developed several other tools that aided us on the final development of CSAT on the UTFSM. We have developed a GUI client as an initial exercise to access our CCD. A snapshot of the main window, with a captured image can be seen in figure 5.

3.3 PUC Deployment

The Pontificia Universidad Católica (PUC) maintains the Observatorio UC [1], a small observatory used for teaching purposes in the University. In this observatory, a Meade LX200 GPS telescope was used to test CSAT. The adaptation of CSAT to this telescope model was done in two different steps.
The first step consisted on a summer job, and it mainly consisted on the first steps of the previous described process: study of the protocol, implementation of a simple client, and implementation of the DevIO’s and the DevTelescope interface. This summer job was the first exercise on using CSAT with another telescope than the Nexstar, which was traduced as bugs fixing, generic code reviewed and some improvements on the control code.

The second phase of this adaptation was done on a visit, after the summer job, to the observatory. The intention of this visit was to test the system in a real manner, making a real amateur observation and testing the whole system, with an attached CCD, and using the latest developments that we had done at the moment. The tests were successful, in terms that we were able to preset with precision to a given coordinate, and that the manual pointing was of help to keep aligned the telescope with the given star.

### 3.4 OCA Deployment

A second external deployment was done on the Observatorio Cerro Armazones (OCA), located at the North of Chile. This observatory is run by the Instituto de Astronomía of the Universidad Católica del Norte, in join with the University of Bochum, Germany. On this observatory, we deployed CSAT on two different telescopes, having very distinct results with each of them.

The first telescope model that we used was a Meade LX200 EMC, with an equatorial mount. It differed from the other Meade LX200 on some details on the protocol (and the mount). Therefore, we had to reimplement the corresponding DevIO’s, and write the component implementation that uses these DevIO’s. We had some problems, anyways, with this telescope, since different speed values could not be set independently for the two axis. This led us to develop a software solution on the component implementation that handled this problem, setting the highest velocity for the axes, but allowing only to one of them to move. Once the two axis receive the same velocity value, then the two of them were moved. This resulted on a good behavior of the telescope, including tracking of a star during a period of one hour, approximately.

The second telescope used at the OCA was the Bochum 84 [cm]. This telescope already counted with a small control software deployed in a Windows-installed machine. This machine was connected to the servo-motors power amplifier, and to the encoders communication board. Therefore, we did not had direct communication with the telescope, but instead had to interface it through the connected computer. The control software installed in the computer allowed external communication through the serial port, and with LX200-like commands, as well as ACL commands.
After some testing, we concluded that the only available method to command the telescope was to send a “goto” command to this software through the serial line. This did not fit with our designed control schema (read positions, set velocities), which disabled us to control the telescope with our software. We tried, anyways, to send a direct “goto” command, but the tests were unsuccessful.

4 Future Work

CSAT has been a first step towards the gTCS. Its development has been frozen, in order to concentrate all the efforts on the gTCS, which will count with support from all the community. Therefore, the next steps on the gTCS project have been defined. They are:

- Design a general architecture for the gTCS. This architecture specification will be published to the community in order to have some feedback from professionals working in the area, with real and long experience on telescope control software design. This specification of the architecture will be reviewed on several iterations, until a final specification will be released.

- After that, a gTCS workshop will be held at the UTFSM. At it will attend a group of professionals working in the area, but now with the goal of defining the main interfaces that should be present at the gTCS, in order to have a generic control specification.

- Finally, and for the rest of the next year, an implementation of gTCS will be developed. There will be several tasks that will be covered by different developing teams: communication layers, logical layers, system testing, GUI development, configuration and deployments, etcetera. These teams will be able to work in parallel thanks to the existence of a unified specification of the interfaces of the system.

5 Conclusions

CSAT was first thought as a first steps towards the gTCS. This idea has been maintained through all the development of the system, giving us direct feedback on our first design ideas of a generic control system.

Thanks to CSAT, we have learned several astronomical concepts that we did not have clear, such as the different coordinate systems used in the observations, time manipulation, physical telescope configurations and parts, general concepts like autoguiding, pointing, tracking, presetting, and so. All these concepts have been a great contribution for our knowledge of the problem that must be covered on the gTCS.

The development of different implementations of the same interface, and the usage of one of them through the configuration on the ACS CDB, has been a great example on how ACS provides us with tools on the development on such a generic system. Interface inheritance is a very interesting idea that must be explored on the gTCS project, and on which we shall take advantages on.

Our empirical experiences using CSAT on different telescopes helped us, also, to realize that our control schema is not the only one. This conclusion, then, must be reflected on the gTCS design. The gTCS should include several control schemes, depending on the characteristics of the telescope that is being plugged to the system.

The gTCS project is not a project aimed only for our group. It will be useful to all the community, specially for those observatories that do not count still with a TCS for their telescopes. Therefore, the community help will be an important factor that will affect the final result of the gTCS project.

Acknowledgments

This work has been supported by project ALMA-CONICYT 31060008 “Software Development for ALMA: Building Up Expertise to Meet ALMA Software Requirements within 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


