A Reference Architecture Specification of a Generic Telescope Control System

JOAO S. LÓPEZ[1], RODRIGO J. TOBAR[1], TOMAS STAIG[1], DANIEL A. BUSTAMANTE[1], CAMILO E. MENAY[1], MAURICIO A. ARAYA[2] AND HORST H. VON BRAND[1]

[1]Computer Systems Research Group, Universidad Técnica Federico Santa María (UTFSM), Valparaíso, Chile
[2]Institut National de Recherche en Informatique et Automatique (INRIA), Nancy, France

ABSTRACT

A Telescope Control System (TCS) is a software responsible of controlling the hardware that an astronomical observation needs. The automation and sophistication of these observations has produced complex systems. Currently, a TCS is compound by software components that interact with several users and even with other systems and instruments.

Each observatory has successfully developed a wide spectrum of TCS solutions for their telescopes. Regardless the mount, there are common patterns in the software components that all these telescopes use. As almost every telescope is custom designed, these patterns are reimplemented again and again for each telescope. This is indeed an opportunity of reuse and collaboration.

The Generic Telescope Control System (gTCS) pretends to be a base distributed framework for the development and deployment of the TCS of any telescope, independent of its physical structure, the type of mount and instrumentation that they use. This work presents an architecture specification explained through two complementary approaches: the layers perspective and the deployment perspective. The first approach defines a set of layers, one on the top of the other, offering different levels of abstraction. Meanwhile the deployment perspective intends to illustrate how the system could be deployed, focused on the distributed nature of the devices.

The Generic TCS Problem

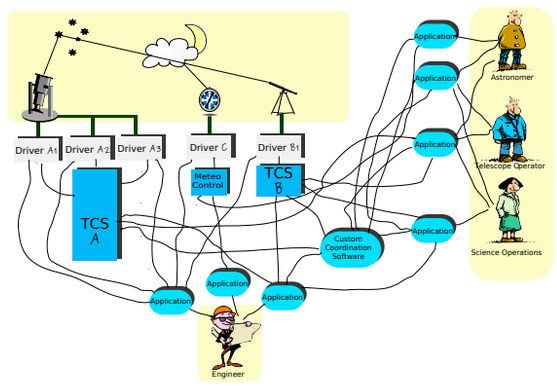
Overview

Control systems have two basic entry points: the *users* and the *devices*. In a TCS domain, users and devices are heterogeneous: users with various levels of expertise, and devices with different protocols and access levels. Therefore, to understand the problem we must identify the diverse nature of users and devices.

The Users

Users can be classified by the *usage* that they give to the system, or in other words, to which *profile* they belong. Profile examples:

- **Observation Control**: The control of the observation itself. Includes all the variables in the domain of the Astronomer, and all the technical and specific details of the telescopes are hidden.
- **Calibration and Startup**: The automatic or manual process of calibrating the telescope, also in high-level domain, but including the specific details of the telescope.
- **Maintenance**: The low-level variables of the telescope, with a detailed control of all the devices and software states.
- **Monitoring**: The summary of the telescope operations to audit, check the behavior of the devices, etc. This may be a mixture of specific logs with general information of the observation.



The problem is that several users could use more than one profile. Then, building an application for each profile is not the most desired approach. Therefore, the existing TCSs often build very complex user interfaces that have all the information that the user *may* need. This turns the application unmanageable for unexperienced users. Fortunately, defining these profiles helps to identify which is the scope of the TCS and select the features that the system will need in a user-independent fashion.

Devices Scope

The devices of a telescope are diverse. Only in the axis control domain each telescope mount/technology has different devices with different protocols and configurations. Even if two telescopes has the same hardware, the firmware or other vendor software could vary. If we add to the equation mirror control, active optics and meteorologic stations, the set of devices turns unmanageable. Therefore, defining the possible set of devices is not a practical approach. A simpler approach is to group the devices into *instruments* that do a specific task. In an observatory there are two general types of hardware devices:

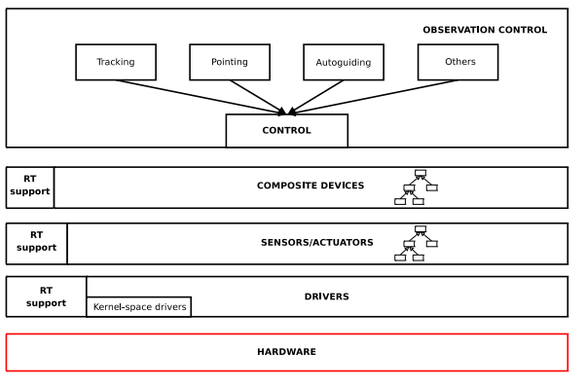
- **Technical Instruments**: All instrument that does not directly produce scientific data, such as the telescope, a meteorologic station, the active optics, an autoguiding CCD, etc.
- **Scientific Instruments**: All instrument that produces scientific data that the astronomer will use, such as the main CCD, a spectrograph, an interferometer, filter wheels, etc.

The scope of a TCS is limited to *technical instruments* and their devices. Also, a TCS must provide all the interfaces to connect the software in charge of managing the *scientific instruments*.

Proposed Architecture Specification

Layers perspective

The *layers perspective* presents different levels of abstraction (higher layers offer higher abstractions). This perspective can be seen as the network stack (such TCP/IP) where each layer offers services to the upper one, and the upper one only uses the lower one.
This view allow us to have an idea of which are the different abstractions that the system needs, and how to encapsulate this information in different layers.



Drivers Responsible of the communication of the system with external entities.

Sensors/Actuators Considering a device as a single monitor/control point associated with a unique property to be read/written, Sensors/Actuators provides software abstractions of physical devices, without considering any interaction with any other device.

Composite devices It represents a full device used on the astronomy domain that can be composed by a given number of sensors and/or actuators.

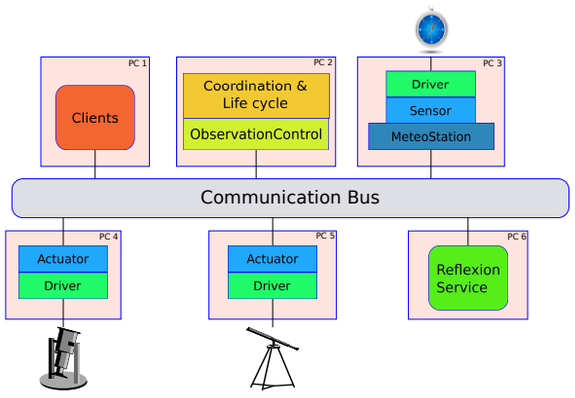
Observation control It offers the abstraction of a full equipped telescope. This is done by grouping different composite devices, and using them in an intelligent way, in order to obtain data, collaborate, and finally do control over the necessary composite devices.

Deployment perspective

The *deployment perspective* intends to illustrate how the architecture can be used in a distributed way, showing the geographical deployment of the system, and the bus that communicates its components.

Each computer will execute different parts of the system and we can distribute one layer through several computers. This is the capability of the system to have its components spread over a network of computers, while working together.

The Information Service can be seen as a software component accessible through any part part of the system. It has two main responsibilities: to know which software components are available in the system and to offer the possibility to retrieve information about the classes, interfaces, methods, parameters and related information from all the components of the system.



Reference technologies

The Control System for an Amateur Telescope [1] project, a TCS constructed over ALMA Common Software (ACS) [2,3] was used as an initial approach to the problem and reference architecture.
To exemplify the proposed reference, existing technologies that are implementing some aspects of the proposal are analyzed.

ALMA Common Software If we consider the Container/Component [4] model present on ACS, each composite device can be managed by an ACS Characteristic component. Through the use of states, exceptions and container lifecycle we are able to manage the lifecycle of the whole gTCS. Currently ACS uses CORBA, which is an example of a communication bus. The information service can be obtained through the Manager (responsible for the management of containers, components and clients) and the information provided by the Interface Repository (IR) and the Configuration Database (CDB).

VLT Common Software The VLT control software [5] uses software modules, sharing benefits of re-usability. These modules are a basic item for the detailed design, development and integration of software. The software architecture is distributed over several workstations that provide high level and coordination services. The communication is based on a message system and a distributed hierarchical database.

Java Remote Method Invocation The Java Remote Method Invocation (RMI) [6] is a Java approach to support a model of a distributed object application. It provides remote communication between programs written in Java. It allows applications to call methods located remotely, sharing resources and processing load across systems.

Acknowledgements

This work was supported by ALMA-CONICYT Fund project #31060008 *"Software Development for ALMA: Building Up Expertise to Meet ALMA Software Requirements within a Chilean University"*, under development at Universidad Técnica Federico Santa María.

References

[1] Tobar, R. et al., "An amateur telescope control system towards a generic telescope control model" in [*Proceedings of SPIE - Advanced Software and Control for Astronomy 2008*], (2008).
[2] Chiozzi, G. et al., "The ALMA Common Software: A developer friendly CORBA based framework" in [*Proceedings of SPIE - Advanced Software and Control for Astronomy 2004*], (2008).
[3] Chiozzi, G. et al., "Application development using the ALMA Common Software" in [*Proceedings of SPIE - Advanced Software and Control for Astronomy 2006*], (2006).
[4] Sommer, H. et al., "Container-component model and XML in ALMA ACS" in [*Proceedings of SPIE - Advanced Software and Control for Astronomy 2004*], (2004).
[5] Chiozzi, G., "An object-oriented event-driven architecture for the VLT Telescope Control Software" in [*Proceedings of ICALEPCS 1995*], (1995).
[6] Grosso, W., "Java RMI", O'Reilly & Associates Inc, (2002).

Atacama Large Millimeter Array

Contact e-mail: jslopez@csrg.inf.utfsm.cl / For more information about ALMA development and research at UTFSM Computer Systems Research Group, please visit our web site http://alma.inf.utfsm.cl