ADASS XIII
ASP Conference Series, Vol. XXX, 2004
F. Ochsenbein, M. Allen and D. Egret eds.

Generic Abstraction of Hardware Control Based on the ALMA Common Software

B. Jeram and G. Chiozzi

European Southern Observatory, Karl-Schwarzschild-Straße 2, D-85748 Garching b. München, Germany, Email: bjeram@eso.org

J. Ibsen

European Southern Observatory, La Silla Observatory

R. Cirami

INAF - Astronomical Observatory of Trieste, Via G.B. Tiepolo 11, I-34131, Trieste, Italy

M. Pokorny

NRAO, 949 N. Cherry Ave., Tucson, Az 85721

D. Muders

MPIfR, Auf dem Hügel 69, D-53121 Bonn, Germany

D. Wischolek

Ruhr-Universität Bochum, Universitätsstr. 150, D-44801 Bochum, Germany

Abstract. The ALMA Common Software (ACS) is a CORBA-based framework that provides a common and homogeneous infrastructure for the whole ALMA software, from high-level data flow applications down to instrument control [O8-1].

This paper focuses on ACS support for the development of Control System applications. In this domain, ACS provides a generic abstraction of hardware control and monitor points that is independent of the software underneath. This abstraction layer is coupled to the hadware using the DevIO (Device Input/Output) interface, based on the Bridge design pattern. Application developers have to implement DevIO classes that handle the details of the communication with the hardware.

ACS itself provides a default DevIO implementation which simply writes and reads into/from a memory location. Currently there are two other major DevIO implementations available: a CAN bus communication, used by ALMA, and a socket based implementation used by the Atacama Pathfinder EXperiment (APEX) project.

In spite of using different hardware and control electronics, the DevIO abstraction allows the ALMA and APEX projects to have the same device architecture down to the level of the DevIO implementation.

1. ACS Overview

The ALMA Common Software (ACS) is a set of application frameworks built on top of CORBA to provide a common software infrastructure to all partners in the ALMA collaboration.

ACS is located between the ALMA application software and other basic commercial or shared software on top of the operating systems and provides a generalized common interface between applications and the hardware in order to facilitate the implementation and the integration in the system of new hardware and software components.

ACS provides basic software services common to the various applications (like antenna control, correlator software, data pipelining) and consists of software developed specifically for ACS and as well of OS build and commercial device drivers. All code developed specifically for ACS is under the GNU Lesser General Public License. Commercial and off the shelf packages are subject to their specific license agreement.

ACS is designed to offer a clear path for the implementation of applications, with the goal of obtaining implicit conformity to design standards and maintainable software. The use of ACS software is mandatory in all ALMA applications, except when the requested functionality is not provided by ACS. Motivated exceptions (for example based on reuse considerations) have to be discussed and approved on a case by case basis.

The main users of ACS will be the developers of ALMA applications. The generic tools and GUIs provided by ACS to access logs, Configuration Database, active objects and other components of the system will be also used by operators and maintenance staff to perform routine maintenance operations.

2. High Level Architecture of an ACS Based Control System

The architecture of a Control System based on ACS can be represented by a multi-tier layout:

1. **Presentation tier:** it provides a GUI and a API interface for an application written using the ACS software to the end user, through the easy embedding of languages like Java and Python.
2. **Middle tier:** in this tier the devices, represented as Components, and their interactions with the rest of the system are modelled using the Component/Property/Characteristics design pattern (see section 3). The underlying infrastructure of this tier consists of C++, Java or/and Python Containers where Components/devices are deployed. One Manager manages the system with the help of the Containers.

3. **Data-access tier:** it is used for working with the information in the configuration database and other databases (e.g. the ALMA Archive which is where all monitor values are eventually stored).
4. **Hardware-access tier:** this tier is used for retrieving data for monitor points implemented in hardware, as well as sending commands to control points.

3. Component/Property/Characteristic Design Pattern

A Component is a CORBA object that corresponds to a physical/logical device, like a power supply, a vacuum pump, or a telescope mount. It is the most natural concept for modelling physical entities.

Each Component has a number of Properties, which represent monitor and control points, e.g. electrical current, status, position etc. They provide asynchronous (using the callback mechanism) and synchronous retrieval and setting of values, monitors and alarms. There are primitive property types (long, double, etc.) and sequences of those primitive data types (longSeq, doubleSeq, etc.). They all can be read-only or read-write objects.

Components and Properties also have a set of Characteristics, which are static data, that are usually - but not necessarily - read from a configuration database, e.g. name, unit, minimum/maximum, etc.

Property interaction with the hardware is performed using the abstract DevIO interface.

4. DevIO - Generic Abstraction of Hardware

As a framework, one goal for ACS is to enable an easy way of integrating new hardware into the application. This is done letting the high level application use the abstraction of Properties, while the Property itself interfaces to the hardware using a hardware specific DevIO implementation.

Properties - high-level abstraction of control/monitor points - provide the application with facilities to monitor and retrieve values from monitor points (get_sync, get_async), and to set the value of control points (set_sync, set_async), or to handle periodic telemetry logging and alarms. Internally, Properties use a DevIO implementation to access the actual hardware (Fig. 1). DevIO is a simple and generic abstraction of hardware monitor and control point, based on the Bridge design pattern. The hardware access points are abstracted with the DevIO parameterized (template) interface, which defines two methods:

- read
- write

These are the only abstract methods used by the Property to access the hardware. The implementations of the DevIO interface expose these two methods and hide the communication with the hadware (e.g. reading from and writing to hardware) and the conversion from raw values to engineer units and vice versa.

For example, the ACS high level code controlling a Power Supply is only aware of the SetPointCurrent control point and ReadbackCurrent monitor point.

Figure 1. DevIO class diagram.

If the actual Power Supply is controlled via an Analog I/O board, the properties are configured with a DevIO capable of reading and setting control registries of the board. If this power supply is replaced by a new one controlled via a serial interface, only the DevIO implementation needs to be replaced.

5. Implementations of DevIO

ACS itself provides a default DevIO implementation, which simply writes to and reads from a memory location. The aim of this memory DevIO is to represent values that are calculated by the software and not directly associated to hardware.

Currently there are two other major DevIO implementations available: a CAN bus communication, used by ALMA, and a socket based implementation used by the Atacama Pathfinder EXperiment (APEX) project:

- ALMA Antenna Test Facility (ATF): Controller Area Network (CAN) bus. All devices at the ATF (i.e. the ALMA prototype antennas) are attached to a CAN bus, through which they are controlled and monitored. The Test Interferometer Control Software (TICS) implements several DevIOs for the different devices used.
- APEX: socket (TCP and UDP). The APEX project uses on one side the TICS software. On the other side it interfaces the system to existing hardware. Those existing embedded systems mostly communicate via socket connection. Thus a DevIO based on a pure socket protocol has been implemented. The socket commands have been standardized using the SCPI syntax.

The Hexapod Telescope (HPT) of the U. of Bochum also uses ACS to interface the following hardware: Heidenhain Encoder board, Motor control unit, Shack-Hartmann sensing unit and Web camera.

References

Schwarz, J., Farris, A., & Sommer, H. 2003, this volume, [O8-1]