Mem. S.A.It. Vol. 78, 719
© SAIt 2007

Memorie della

An overview of the ALMA Common Software (ACS)

P. Di Marcantonio[1], R. Cirami[1], A. Caproni[1,2], G. Chiozzi[2], B. Jeram[2], H. Sommer[2], S. Harrington[3], K. Zagar[4], M. Plesko[4], and M. Sekoranja[5]

[1] Istituto Nazionale di Astrofisica – Osservatorio Astronomico di Trieste, Via Tiepolo 11, I-34131 Trieste, Italy
e-mail: `dimarcan@oats.inaf.it`
[2] ESO – European Southern Observatory, Karl-Schwarzschild Strasse 2, D-85748 Garching bei Munchen, Germany
[3] National Radio Astronomy Observatory, Socorro, N.M. 87801, USA
[4] CosyLAB, Teslova 30, SI-1000 Ljubljana, Slovenija
[5] Joseph Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenija

Abstract. The ALMA Common Software (ACS) is an application framework designed to provide a common and homogeneous software architecture and infrastructure, spanning the end to end needs of an Astronomical observatory, from the Telescope Control system to high-level data flow management.
ACS offers, at the lower level, several basic services needed for object-oriented distributed computing like transparent remote object invocation, object deployment and location, distributed error, alarm handling, logging and events. On top of this it provides an application architecture based on the Component/Container paradigm that fosters sharing and reusing of software components.
Although developed for the ALMA project, ACS is now used by several other projects worldwide, among which the Italian Sardinia Radio Telescope (SRT). Besides, there is an active community that shares ideas, concepts and actual software components.
Major drivers for this diffusion were the choice of adopting the LGPL public license and the adoption of CORBA, a free but reliable and widely used middleware software. In this paper we present an overview of the main features of ACS, emphasizing in particular the role of INAF-OAT in this project.

Key words. CORBA, common software, middleware, object-oriented

1. Introduction

The Atacama Large Millimeter Array (ALMA) is one of the largest ground-based astronomical projects of the next decade aiming at the construction of a complete astronomical and imaging instrument dedicated to millimeter/submillimeter astronomy. It will consist of an array of 50 antennas with reconfigurable baselines up to 14 km, located on the Chajnantor plain of the Chilean Andes at 5000 m above sea level and operated from the

Send offprint requests to: P. Di Marcantonio

Operations Support Facility (OSF) situated at an elevation of 2900 m.

ALMA will be therefore a truly highly distributed system, not only from an operational/control point of view, but also from the development perspective, being in fact an international collaboration between Europe, Japan and North America in cooperation with the republic of Chile.

To cope with such a complexity a new software infrastructure, the ACS (ALMA Common Software) has been implemented. The purpose of ACS is twofold: from the system perspective it provides the implementation of a coherent set of design patterns and services that will make the whole ALMA software uniform and maintainable; from the perspective of an ALMA developer, it provides a friendly programming environment in which the complexity of CORBA (the chosen middleware on which ACS is based) and other libraries are hidden and coding is drastically reduced.

It is important to underline that ACS is available under the LGPL public license and since the beginning it has been developed as a general purpose application framework (i.e. not specifically linked to ALMA needs). The net result is that after the first ACS presentation (2001), several projects adopted it (also outside the astronomical community) and among others also the Sardinia Radio Telescope (SRT).

2. ACS Main Features

2.1. Component Container Paradigm

One of the core elements of ACS is the Component/Container paradigm, whose main purpose is from one side to decouple technical and functional aspects in the applications development and from the other side to facilitate deployment/administration of the whole system. The whole ALMA software is written in terms of components, basically CORBA objects, which could model a physical device or an abstract entity (e.g. an algorithm, an object used to access a database or whatever). Within ACS, developers need only to implement the functional aspects in the Components (just as an example, implement the method "move" for a motor device), but they must not be experts in the technologies used underneath. It is the Container, implemented by the ACS team, which takes care of the technical concerns of the system, manages the lifecycle of the Components, activates and deactivates them when requested and manages their state according to a standardized state machine.

The deployment of the Components in the system is completely decoupled from the implementation of the Components themselves. This means that Components are unaware of the system where they effectively run. They are also unaware of the location where other Components they might need are deployed. In this way it is easier to change the system deployment to improve performance and to make it scale up while it grows.

The administration of the whole system is demanded to a Manager, which is the broker for the Components. It is capable of handling both quasi-static model of control system (i.e. where the numbers of devices/components are statically configured) and fully dynamics components whose number and configuration varies according to the usage of the system (e.g. computation components in an image pipeline, which could be called simultaneously by many users).

It is worth noting that the ACS Component/Container model was designed in 2000. Since then, the .Net and EJB model have become well established commercial implementations proving the validity of the chosen solution.

2.2. Support for multiple programming languages

The middleware beneath ACS is CORBA. It was chosen because it is a widely adopted standard, and as such it offers platform independence and abundance of existing services and solutions. Among various possible choices, three free CORBA implementations were chosen: ACE/TAO for C++ support, JacOrb for Java support and omniORB for Python support. ACS is capable therefore of supporting in a transparent way and providing most of the same features for all the three languages with the following usage:

- C++/C for high performance, algorithmic and numerical functions and real time applications in the Control System domain;
- Java for higher level and coordination applications, also in the Control System domain, and for GUIs;
- Python for prototyping, scripting, testing, small engineering user interfaces and in general as a glue language.

2.3. Real Time Support

In the last two years the ALMA project has carried on a major paradigm shift in the implementation of real time software. The real time part of the control system for the Test Interferometer was implemented using VxWorks. This implied the usage of Local Control Units (basically VME crates equipped with PPC CPU) running the VxWorks operating system. A complete implementation of ACS Containers in VxWorks was necessary to support the deployment of Components on the LCUs. Presently we use Real Time Linux (in the RTAI flavour) and the new implementation of the control system based on RTAI is currently being deployed on the test antennas. The decision to change the real time operating system has been the result of an evaluation of the two alternatives in terms of technical advantages and disadvantages, cost and future perspectives.

2.4. Distributed support

Beside the object deployment and location based on the Component/Container paradigm, ACS provides all the basic services needed for the object-oriented distributed computing:

- transparent remote object invocation;
- distributed error and alarm handling;
- distributed logging;
- distributed events (notification channel);
- threading support;
- synchronous and asynchronous method calls.

For detailed information on the specific aspect, the interested reader could consult the references or to the documentation available online (http://www.eso.org/~almamgr/AlmaAcs/index.html).

3. The role of INAF-OAT

The Astrophysical Technologies Group (ATG) of the INAF-AOT has joined the ACS project since 2002 and has significantly contributed to the project in various areas offering 2 FTE/year of manpower, with one person permanently at ESO headquarters. Among others the most significant contributions are described in the following subsections.

3.1. Interface to the hardware

As far as interfacing to hardware is concerned, ACS provides a generic abstraction of hardware control and monitor points which is independent of the software underneath and is based on the Component/Property/Characteristic design pattern. This abstraction layer is coupled to the hardware using the DevIO (Device Input/Output) interface, based on the Bridge design pattern. Application developers have to implement only the DevIO classes that handle the details of the communication with the hardware. In most cases this simply implies the implementation of a "read" and/or a "write" method(s) that goes down to the hardware level.

Several DevIO implementations are already at disposal of users such as DevIo for CAN bus, serial communication, encoder boards etc.

3.2. Bulk Data Transfer

One of the needs coming from the Correlator (the device responsible for the processing of raw digitized data from the antennas) is an efficient transfer of huge amounts of data. For example, a typical output data rate expected from the Correlator, when in operation, will be of the order of 550 Mbit/sec, and is well known that CORBA has some problems to meet such a QoS requirements. To overcome the CORBA potential bottleneck, we introduced in ACS a new transfer mechanism based on the ACE/TAO CORBA Audio/Video (A/V) Streaming service (the ACS Bulk Data Transfer). This architecture uses CORBA for handshaking (thus enabling ALMA applications to keep leveraging the inherent portability and flexibility benefits of the ACS middleware), but allows an efficient data transfer by creating out-of-bound stream(s) of data (i.e. bypassing the CORBA protocol and using TCP/IP instead).

On top of this, we designed and implemented also a Distributor model, which mimics a multicast behaviour. One or more receivers can subscribe to a common object (the Distributor) which receives data from one Sender (e.g. the Correlator), and dispatches them to all the subscribed Receivers (using out-of-bound connections). Preliminary performance tests showed that even by using three Receivers we are well inside the specs reaching a transfer rate of the order of 700 Mbit/sec.

3.3. Sampling System

We designed and implemented a sampling system which allows sampling of every Characteristic Component Property (e.g. the current of a motor, the brightness of a lamp or whatever) with a specific, user-defined, sustained frequency limited only by the hardware. Collected data are sent to various clients (one or more Java plotting widgets, a dedicated GUI or a COTS application) using the ACS/CORBA Notification Channel. The data transport is optimized: samples are cached locally and sent in packets with a lower and user-defined frequency to keep network load under control. Simultaneous sampling of the Properties of different Components is also possible.

3.4. Alarm System

An alarm system is a cornerstone service in every computer controlled environment. Its purpose is the notification of exceptional conditions in the system requiring an intervention from the staff. The philosophy of ACS is to look for existing implementation before writing a service from scratch. For the alarm system such an implementation, the CERN Laser Alarm System developed for the Large Hadron Collider, was fulfilling and exceeding our requirements. We have therefore started a collaboration with CERN and integrated the Large Hadron Collider Alarm Service (LASER) into ACS. Particular attention was given to the reduction mechanism for its importance in helping the operators find the real cause of each problem in a short time. The project implicitly showed that it is possible to integrate two different software systems if they are written with well defined interface and have a similar infrastructure.

4. Conclusion

We presented an overview of the ACS, framework for the development of highly distributed systems. During the last years it has become very mature and complete. Thanks to the adopted LGPL licence its usage has not been limited only to the ALMA project. ACS is currently used in several other worldwide projects one of which is the Sardinia Radio Telescope. Since the beginning INAF-OAT has played an important role in the overall project and its main contributions are outlined in the paper.

References

Chiozzi, G., et al. 2006, SPIE, Vol. 6274, pp. 627406

Di Marcantonio, P., et al. 2004, SPIE, Vol. 5496, pp. 402-410

Cirami, R., et al. 2006, SPIE, Vol. 6274, pp. 62741E

Caproni, A., et al. 2006, SPIE, Vol. 6274, pp. 627407-2