ACS, a CORBA-based Common Software for ALMA and other projects

G.Chiozzi*, B.Gustafsson*, B.Jeram*, P.Sivera* M.Plesko**, M.Sekoranja**, G.Tkacik**, K.Zagar**, D.Fugate***

*ESO, ** CosyLab, *** NRAO

ABSTRACT

The Atacama Large Millimeter Array (ALMA)[1] is a joint project between astronomical organizations in Europe and North America. ALMA will consist of at least 64 12-meter antennas operating in the millimeter and sub-millimeter range, with baselines up to 14 km. It will be located at an altitude above 5000m in the Chilean Atacama desert.

The ALMA Common Software (ACS)[2][3] provides a software infrastructure common to all partners and consists of a documented collection of common patterns and of components implementing those patterns.

The heart of ACS is an object model based on Distributed Objects (DOs), implemented as CORBA objects. The teams responsible for the control system development use DOs as the basis for components and devices, like an antenna mount control.

ACS provides common CORBA-based services like logging, error and alarm management, configuration database and lifecycle management. A code generator creates a Java Bean for each DO.

ACS is based on the experience accumulated in the astronomical and particle accelerator contexts, reusing and extending concepts and components.

Although designed for ALMA, ACS has the potential for being used in other new control systems, since it implements proven design patterns using state of the art, stable and reliable technology. Like all software developed for the ALMA project, ACS follows the GPL licensing scheme to foster reuse and collaboration in the scientific community.

This paper presents status and architecture of ACS.

1. THE RATIONALE FOR A COMMON SOFTWARE

Since its beginning, the ALMA project [1] has been characterized by complexity due to the wide geographical distribution of its development teams and their diverse development cultures.

To alleviate these problems, we have introduced a central object oriented framework, the ALMA Common Software (ACS)[6]. It is located in between the ALMA application software and other basic commercial or shared software on top of the operating systems. It provides a well-tested platform that embeds standard design patterns and avoids duplication of effort. At the same time it is a natural platform where upgrades can be incorporated and brought to all developers. It also allows, through the use of well-known standard constructs and components, other team members who are not authors of ACS to easily understand the architecture of software modules, making maintenance affordable even on a very large project.

In order to avoid starting from scratch, we have evaluated emerging systems that could provide a good basis for ACS and would bring to the project CORBA[13] and other new technological know-how. We then began a fruitful collaboration between ESO and JSI that, through exchange of experience and ideas from our previous projects has brought us to the concepts and implementation of ACS. For more details on the considerations that have led to the concepts behind ACS see [2], [7] and [8].

During the first phases of the project, we have concentrated on the aspects related to the Control System for ALMA, driven by the requirements of the ALMA Test Interferometer[5]. Now the ACS team is working with the ALMA Architecture team to seamlessly extend ACS to cover the needs of the higher-level software sub-systems[4].

2. ACS ARCHITECTURE

ACS is based on the object oriented CORBA middleware, which gives the infrastructure for the exchange of messages between distributed objects and system wide services [9]. Whenever possible, ACS features are implemented using off-the-shelf components; ACS itself provides in this case the packaging and the glue between these components. At the same time, whenever convenient ACS hides all details of the underlying mechanisms, which use many complex features of CORBA such as queuing, asynchronous communication, thread pooling, life-cycle management, etc.

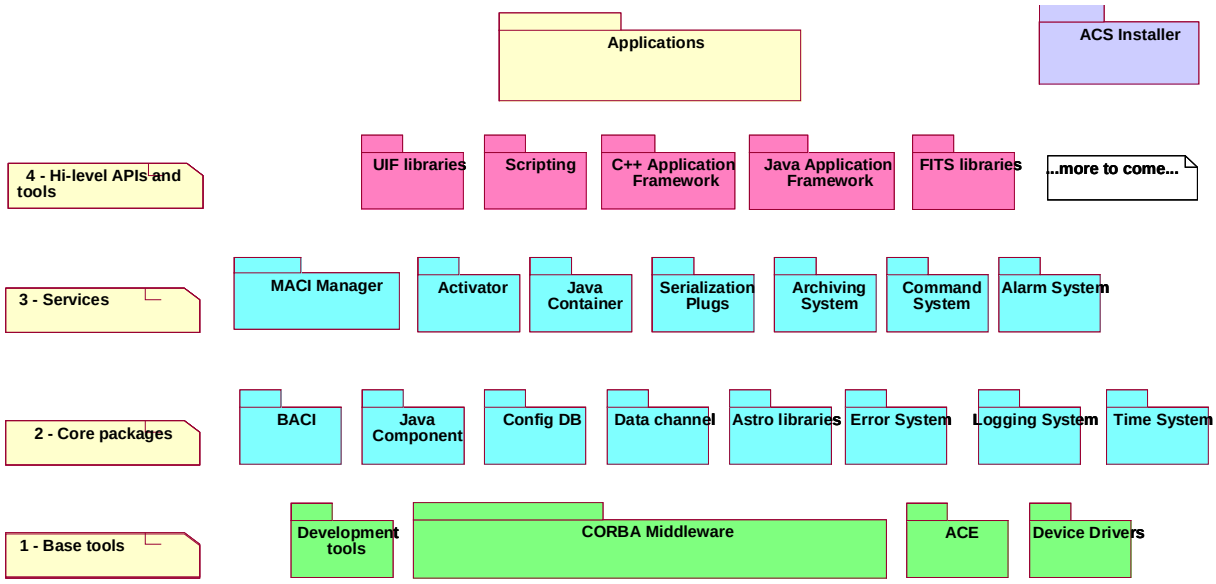

Fig. 1: ACS Packages

2.1 Component-Container model

The ACS Architecture is founded on the Component-Container model [15].

Containers provide an environment for Components to run in, with support for basic services like logging system, configuration database, persistency and security. Developers of Components can focus their work on the domain-specific "functional" concerns without having to worry about the "technical" concerns that arise from the computing environment in which their components run.

The division of responsibilities between components and containers enables decisions about where and when individual components are deployed to be deferred until runtime. If the container manages component security as well, authorization policies can be configured at run time in the same way.

Commercial implementations of the Component-Container model are quite popular in industry at present, with Sun's Enterprise Java Beans and Microsoft's .NET being the prime examples. A vendor-independent specification, the Corba Component Model (CCM), is under development, production implementations do not yet exist. These are rather comprehensive systems, and require a wholesale commitment from developers to use the languages and tools supplied. For our application domain we do not need all the power offered by such systems and ACS implements an infrastructure that is more lightweight. We are anyway committed to move to the CCM as soon as we find a suitable free implementation.

2.2 ACS Packages

The UML Package Diagram in Fig. 1 shows the main packages in which ACS has been subdivided. For more details, refer to the ACS Architecture, available on the ACS Web Page[2].

ACS has a classical layered architecture, where packages are allowed to use services provided by other packages on lower layers and on the same layer, but not on higher layers.

Each package provides a basic set of services and tools that shall be used by all ALMA applications.

2.2.1 Base Tools

The bottom layer contains base tools that are distributed as part of ACS to provide a uniform development and run time environment on top of the operating system for all higher layers and applications. These are essentially off-the-shelf components and ACS itself simply provides packaging, installation and distribution support. This ensures that all installations of ACS (development and run-time) will have the same basic set of tools.

2.2.2 Core Packages

This second layer ensures standard interface patterns and implements essential services, necessary for the development of any application. Among these:

- **Basic Access Control Interface (BACI)**
  We expect that the ALMA control system will be mainly implemented in C++, to satisfy performance and real time requirements. This package provides the implementation of C++ Components, introducing the concepts of Distributed Object, Property and Characteristic [10]. The object paradigm of CORBA is fully applied: each entity in the control system is defined as a Component type and is represented by

one specific CORBA interface that subclasses the base Distributed Object (DO). Each DO is further composed of Properties that correspond to what are called controlled points, channels or tags in Supervisory Control and Data Acquisition systems (SCADA). Each Property is an object too, described by Characteristics such as min/max values or units.

- **Java Component**
  Higher-level software, in particular data-flow applications, will be instead mainly developed in Java. This package provides a Java implementation for the Component model. The Java Component package is currently under design.
- **Configuration Database**
  This package addresses the problems related to defining, accessing and maintaining the configuration of a run-time system. For each Component in the system, there are configuration parameters that must be configured in a persistent store and read when the Component is started up or re-initialized.
- **Data Channel**
  The Data Channel provides a generic mechanism to asynchronously pass information between data publishers and data subscribers, in a many-to-many relation scheme. It is based on the CORBA Notification Service [13].
- **Error System**
  API for handling and logging run-time errors, tools for defining error conditions; tools for browsing and analyzing run-time errors.
- **Logging System**
  API for logging data, actions and events. Transport of logs from the producer to the central archive. Tools for browsing logs. It is based on the CORBA Telecom Logging Service [13].

2.2.3 Services

The third layer implements higher-level services. Among these:

- **Management and access control interface (MACI)**
  Design patterns, protocols and meta-services for centralizing access to ACS services and Components, to manage the full life cycle of Components, including persistence, and to supervise the state of the system [11]. This has been split in three packages:
  - **MACI Manager** contains the higher-level system supervisor (Manager).
  - **Activator** implements the C++ Container.
  - **Java Container** implements the Java Container for higher-level application components. The Java Container is currently under design.
- **Archiving System**
  API tools and services for archiving and monitoring data and events.

2.2.4 Application Frameworks and High-level APIs

The fourth and last layer provides higher-level APIs and tools. The main goal of these packages is to offer a clear path for the implementation of applications, in order to obtain implicit conformity to design standards. Among these, we mention:

- **UIF Libraries**
  Development tools and widget libraries for User Interface development. Java user interfaces are based on the ABeans library that wraps CORBA objects within Java Beans, which are then connected with commercial data-manipulation and visualization Beans using visual tools or programmatically[12].
- **ACS C++ and Java Application Frameworks**
  Implementation of design patterns and to allow the development of standard applications.

3. ACS DEVELOPMENT STATUS

The development of ACS is driven by the needs of the teams developing higher-level software, and in particular now the ALMA Control System.

3.1 ACS Release Policy

Our development cycle foresees one major release every year, with an intermediate, bug-fix release after six months. The complete ACS SW Development Plan, is available on the ACS Web Page [1].

ACS 0.0, released in September 2000 was essentially a concept demonstration prototype. With the support of some components taken from the VLT Control Software it has been used to develop a prototype control system for the 12m Kitt Peak antenna. This was successfully tested in December 2000.

ACS 1.0, released in September 2001, was the first “production release”. It was followed in April 2002 by ACS 1.1. ACS 1.1 includes an essentially complete implementation of the BACI, MACI and Abeans packages, together with Error System, Logging System, the first elements of the C++ Application Framework and prototypes for many other packages. It still relies on some components of the VLT Common Software including, in particular, the VLT Real Time Database as configuration database engine.

The next major release, ACS 2.0, is foreseen for September 2002. The main objectives for this release are to become fully independent of the VLT Common Software and to provide a Configuration Database

Engine based on an XML file hierarchy. The Time System and Notification Channel packages developed by the TICS team will be integrated into ACS. All existing components will be extended and stabilized.

We have started to work in parallel on the evolution of ACS to accommodate the needs of the ALMA Data Flow Subsystems (Archiving, Scheduling, Observation Tools, Pipeline, etc.). After ACS 2.0 the first prototypes for the Java Component-Container packages will be made available. Future releases will concentrate more on performance and scalability.

3.2 ACS Supported Platforms

ACS is supported for ALMA on Linux and VxWorks. An MS Windows version is running at the ANKA Synchrotron. A Solaris version is used internally at ESO for testing purposes. Other platforms, like Real Time Linux, are being investigated, but the porting is expected to be very easy thanks to the ACE [14] operating system abstraction layer.

3.3 ACS Installations

ACS 1.1 is used in more than 10 sites worldwide for different projects mostly, but not exclusively, related to ALMA:

- For the development of TICS, the ALMA Test Interferometer Control System[5]. The first test antenna is being installed and tested at the VLA site.
- For the development of the APEX (Atacama Pathfinder Experiment), a radio telescope built by the Max Planck Institute for Radioastronomy in Germany, with the collaboration of ESO.
- For the development of the Japanese ALMA Antenna Prototype.
- It is in operation at the ANKA Synchrotron in Karlsruhe, (Germany), in its Windows NT porting.

Some ACS components, like the Abeans libraries are used by other projects as well and we are discussing the possibility of collaboration with other groups.

4. CONCLUSION

ACS has been developed keeping in mind the needs of a wide range of astronomical and accelerator control projects. It runs readily on many platforms and operating systems and is open source. The complete code will be available under GNU Public License, is compiled with the standard GNU gcc compiler, and includes the sources of the underlying CORBA implementation, TAO[14], which is also open source. A part of the service client applications are written in Java, using a free Java ORB. We are therefore convinced that many other projects can use ACS. At the same time, we think that a wider user base can provide us with very valuable feedback.

ACKNOWLEDGEMENTS

The ACS project is managed by ESO in collaboration with JSI. This work is the result of many hours of discussions, test and development inside our groups and in the various ALMA centers at NRAO, IRAM and Bochum.

REFERENCES

1. ALMA Web page, http://www.mma.nrao.edu/
2. ACS web page and online documentation, http://www.eso.org/~gchiozzi/AlmaAcs
3. G.Chiozzi et el., CORBA-based Common Software for the ALMA project, SPIE conference 4848, Kona, Hawaii, Aug. 2002
4. J.Schwarz, G.Raffi, “ALMA Software Architecture”, these proceedings
5. R.G.Marson, B.Glendenning, “ALMA Test Interferometer Control Software”, these proceedings
6. G.Raffi, G.Chiozzi, B.Glendenning, “The ALMA Common Software (ACS) as a basis for a distributed software development”, ADASS XI, Victoria, BC, Canada, Sep. 2001
7. M. Plesko, “Implementing Distributed Controlled Objects with CORBA”, PCaPAC99, KEK, Tsukuba, Jan. 1999
8. B. Jeram et al., “Distributed Components in Control”, ICALEPCS 1999, Trieste, Nov. 1999
9. G. Milcinski et al, “Experiences With Advanced CORBA Services”, ICALEPCS 2002, San Jose, CA, Nov. 2001
10. G. Tkacik et al., BACI specs, see [2]
11. K. Zagar et al., MACI specs, see [2]
12. G. Tkacik et al., “Java Beans of Accelerator Devices for Rapid Application Development”, PCaPAC99 workshop, KEK, Tskukuba, January 1999
13. CORBA Home Page at OMG: http://www.corba.org/
14. ACE/TAO Home Page: http://www.cs.wustl.edu/~schmidt/TAO.html
15. M.Völter, A.Schmid, E.Wolff, *Server Component Patterns*, Wiley, Summer 2002