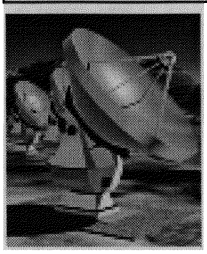
ALMA Computing Memo No. 5



EUROPEAN SOUTHERN OBSERVATORY

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

ALMA PROJECT

ATACAMA LARGE MILLIMETER ARRAY

ALMA Common Software Technical Requirements

Doc. No.: ALMA-TRE-ESO-XXXXX-XXXX

Issue: 1.0

Date: 2000-06-05

Prepared:	G.Raffi (ESO), B.Glendenning (NRAO)		2000-06-05
	Name	Date	Signature
Approved:			
	Name	Date	Signature
Released:			
	Name	Date	Signature

CHANGE RECORD

ISSUE	DATE	SECTION/PAGE AFFECTED	REASON/INITIATION DOCUMENTS/REMARKS
1.0/Prep1	2000-03-10	All	First version
1.0	2000-06-05	All	Reviewed version

1 Preface

1.1. Reference Documents

These Technical Requirements are based largely on the Feature List in the document referenced below. The Feature List is now being rewritten to take these Technical Requirements into account. The Feature List will emphasize the design and prototyping work being carried out in support of the ALMA Common Software. This separation of implementation aspects allows the Technical Requirements to be presented with a better feel for implications and feasibility issues.

ALMA Common Software Feature List, G.Chiozzi, Issue 1.0/Prep.2

1.2. Acronyms

ACS	ALMA Common Software
ACU	Antenna Control Unit
ALMA	Atacama Large Millimeter Array (http://www.eso.org/projects/alma/)
API	Application Programmatic Interface
BOA	Basic Object Adapter
CAN	Controller Area Network
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DBMS	Data Base Management System
DO	Distributed Objects
ESO	European Southern Observatory (http://www.eso.org)
FITS	Flexible Image Transport Format
GUI	Graphical User Interface
ICD	Interface Control Document
IDL	Interface Definition Language
M&C	Monitoring and Control
LAN	Local Area Network
NRAO	National Radio Astronomy Observatory (http://www.nrao.edu/)
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
POA	Portable Object Adapter
RTOS	Real Time Operating System
SW	Software
TAI	International Atomic Time
TBC	To Be Confirmed
TBD	To Be Defined
UML	Unified Modeling Language
UTC	Universal Time Coordinated
VME	Versa Module Eurocard
WAN	Wide Area Network
WS	Workstation
XML	Extensible Markup Language

2. INTRODUCTION

2.1. Scope

The purpose of this document is to define an initial set of technical requirements for the ALMA Common Software (ACS), in particular those related to control software aspects, to allow implementation of the first version of ACS.

These requirements are complementary to the high level scientific and technical requirements. The user of the ACS software is typically a developer, producing specific application software to satisfy those requirements. The requirements expressed here shall reflect the needs of developers and the constraints of the ALMA project.

2.2. Overview

The ACS software is located in between the application software (on top) and other commercial or shared software over the operating systems. ACS is meant to be the kernel of a larger system of software that relies on ACS. Any software that utilizes ACS, either as an extension to ACS or through use of the ACS primitives, must be tested and distributed as an integrated unit.
Along with Requirements for ACS there are also Requirements for Application software written using ACS. It is in fact one of the purposes and advantages of an ACS, to enforce standards simply by asking to use the ACS software in all applications. The requirements for application software are grouped together in a separate chapter.

Requirements are subdivided in different categories according to their **Priority**.
Priority can be: **Critical, Major, Desirable.**

This version of the requirements does not make use of priorities.
Requirements that are not fully clear are marked as TBD or TBC.

2.3. Reference Architecture

A reference layout for the system will be provided in the ALMA project book. For the purposes of this document a distributed architecture based on computers at the individual antennas and a number of central computers is assumed. It is also assumed that at the antenna there will be not only Local Area Network (LAN) connectivity but also a distributed Fieldbus (CAN).

3. General requirements

3.1. ACS scope

3.1.1. Scope. ACS shall contain software that is common and supporting the various applications (like antenna control, correlator software etc). It shall fulfil the technical and scientific requirements of ALMA.

3.1.2. Design. The design of ACS shall offer a clear path for the implementation of applications, with the goal of obtaining implicit conformity to design standards and the production of maintainable software.

3.1.3. Use. The use of ACS software is mandatory in all applications, except when the requested functionality does not exist. Exceptions to this shall be explicitly authorised at design review and shall be implemented with a design compatible with ACS.

3.2. User requirements

3.2.1. Users. These are Developers, Operators performing routine maintenance operations via standard tools of ACS, and Maintenance staff, who typically access low level functionality of the system.

3.2.2. Value retrieval. It shall be possible to retrieve locally and remotely information in read mode without interfering with control actions.

3.2.3. Value setting. This shall normally be done programmatically or via GUIs enforcing access to different kind of values (engineering GUIs for engineering values, user GUIs for normal users).

3.2.4. Local and central operation. Every software operation available shall also be available at the Control centre in San Pedro, without significant degradation in performances.

3.2.5. Remote access. Remote access from the US and Europe shall be possible and reliable within given rules and security limits (TBD). Performances will be on a best effort basis.

3.3. System requirements

3.3.1. Size. ACS shall be capable to cope with 100.000 Distributed Objects (extrapolating from the assessment of about 1000 I/O points/antenna, which will the basis for most distributed objects in the system and adding the need for purely logical objects on top).

3.3.2. Serialization. ACS shall provide a simple serialization mechanism to create on demand a persistent image of certain types of objects. Default persistent values will be applied at start-up, and default or other persistent values can be used for reinitialization.

3.3.3. Migration. It must be possible for special objects (e.g. observing blocks) to make use of Serialization to migrate across links (rather than being accessed where they are).

3.3.4. Simulation. ACS shall support distributed objects in simulation. This allows testing when some or all the hardware is unavailable.

4. Distributed object values

4.1. Value retrieval

4.1.1. Direct value retrieval. It shall be possible to retrieve values directly, typically for programmatic use.

4.1.2. Indirect value retrieval. This consists in retrieval of values that have been cached, to provide a mirrored view of the running system. This allows to keep system and network load under control.

4.1.3. Rate. Indirect value retrieval shall be available both in periodic form and on change.

4.1.4. Transparency. Applications should be the same for getting direct or indirect values. This could simply depend on start-up parameters, but the code should not need changing.

4.1.5. Values at given time. Time stamped data requests shall allow to retrieve values at a specified array time.

4.1.6. Data channel. Event/data channels to provide connections between producing and consuming objects shall be supported by ACS.

4.1.7. Quality. Retrieved values shall have an associated quality to state if the value is reliable.

4.2. Database

4.2.1. Configuration Database. Distributed object information shall be stored in a Database, to be downloaded at start time and to be accessed whenever needed to retrieve definitions.

4.2.2. Database design. The exact form of the database is TBD at Database design time.

4.3. Sampling

4.3.1. Sampling. It shall be possible to store samples for later analysis or plotting, sending buffered values to keep network load under control.

5. Tools and Libraries

5.1. Tools

5.1.1. Framework. The common software shall provide an application framework that allows an easy implementation of a standard application with all essential features and standard messages.

5.1.2. Procedures. Start-up and shut-down procedures shall exist in ACS, providing also a framework for applications.

5.1.3. Browser. There shall be a Tool to browse through, display and modify object values. This should know the configuration of the objects installed and their availability, including the fact that they are accessible or not to a given user.

5.1.4. Analysis tool. An existing engineering data analysis tool to display several object values at the same time shall be integrated in ACS.

5.2. Scripts

5.2.1. Scripts. A technical scripting language shall be part of ACS. It will be used for high level procedures such as coordination functions, prototyping and test scripts.

5.2.2. Functionality. The scripting language shall provide access to the basic features of ACS, having interfaces to CORBA as a pre-requisite.

5.3. Libraries

5.3.1. Time conversion, arithmetic and formatting (could be a Time class).

5.3.2. Mathematics (e.g. FFT library for correlator)

5.3.3. Astrometry

6. Control information flow

6.1. Messages

6.1.1. Messages include commands and normal or error replies. Command replies can return information along with command acknowledgement.

6.1.2. Commands. They are the way to perform actions or to request information to distributed objects. They normally shall have a set of associated input and output parameters.

6.1.3. Syntax Check. The syntax of commands shall not need to be checked for correctness when the message is sent programmatically. This does not exclude that there might be tools (e.g. GUIs, interactive commands, scripts) where it is desirable to do a syntax check also at the origin.

6.1.4. Validation. Range checking and full context validation of message syntax shall in any case be done at the receiving end.

6.1.5. Programmatic use. It shall be possible to invoke commands both programmatically and interactively, including a generic tool performing also syntax checking. There shall only be one file being used for the syntax and range checks mentioned under Syntax Check, Validation and Programmatic use.

6.1.6. Command delivery. Command delivery shall be reliable, i.e. commands shall always be acknowledged.

6.1.7. Mode. It shall be possible to send commands either synchronously or asynchronously.

6.1.8. Timeout. An adjustable timeout shall be supported for commands (so that users get a reply in any case).

6.1.9. Intermediate replies. These are allowed for asynchronous commands and there shall be a way to link replies to messages (e.g. headers) and to identify the final reply.

6.2. Logging

6.2.1. Logging. Applications shall be able to log information at run time according to specific formats in order to log a record of: execution of actions, status, anomalous conditions. All log records will have an associated array time.

6.2.2. Persistency. Short term persistency of logs shall be provided by having the total size and/or time span of such logs adjustable. Long term persistency shall be provided by a backup policy TBD.

6.2.3. Filtering. It shall be possible for log messages to be searched and filtered (both in input, e.g. to avoid flooding of identical messages and in output).

6.3. Errors

6.3.1. Definition. An Error is a failure being reported back either synchronously after a command or asynchronously.

6.3.2. Tracing. Errors shall be traceable at interesting levels.

6.3.3. Severity. Errors shall have a Severity attribute. For serious unrecoverable errors the subsystem concerned shall fall back to a safe state.

6.3.4. Presentation. A standard way to report errors to users shall be defined for all user interfaces.

6.3.5. Configuration. Error, help text and descriptions of manual recovery procedures, shall be predefined in error configuration files.

6.3.6. Scope. Error messages shall be reported only where they occur (normally to the action requester), so that a faulty subsystem cannot flood with errors other parts of the system.

6.4. Alarms

6.4.1. Definition. Alarms are events associated with subsystem anomalies, which rely on continuous retrieval of relevant distributed object values.

6.4.2. Behaviour. It shall be foreseen to categorise alarms, so that some alarms just disappear when the alarm condition is cleared, while others will require an explicit acknowledgement.

6.4.3. Severity. Alarms shall have a Severity attribute. For serious unrecoverable alarms the subsystem concerned shall fall back to a safe state.

6.4.4. State. Alarms should have a state, e.g., FIRST OCCURRENCE, MULTIPLE OCCURRENCE, TERMINATED, IDLE, etc.

6.4.5. Hierarchy. It shall be possible to define hierarchical alarms: by default only high level alarms are shown to the operator.

6.4.6. Alarm logging. All alarm transitions shall be logged.

6.4.7. Configuration. Alarms shall be predefined in alarm configuration files. These should also provide in addition help text and descriptions of manual recovery procedures.

6.4.8. Binding. It shall be possible to define actions to be started on the occurrence of alarms. In general event driven software shall be supported by ACS.

7. Astronomical data flow

7.1. Bulk data transfer

7.1.1. Image pipeline data transfer rates of 3 MB/sec sustained and 30 MB/sec burst shall be supported.

7.1.2. Transport mechanism. A bulk data transport mechanism shall exist within ACS (TBD)

7.2. Data format

7.2.1. FITS format shall be supported.

8. Timing signals flow

8.1. Time System

8.1.1. Standard. Array time shall be based on a standard TBD (UTC or TAI).

8.1.2. API. There shall be an API to access time information in ISO format (one time zone, e.g. UTC TBC). This shall make use of the Time library defined earlier.

8.1.3. Distributed timing. The distributed timing system consisting of a periodic pulse with a period of 48 millisecond. The leading edge of each pulse marks a Timing Event.

8.1.4. Services. ACS shall support attachment to the Timing Event via event synchronization.

8.1.5. Resolution. The API shall be the same for all platforms, but provides higher time resolution and performances on platforms that have specific hardware support.

9. Attributes

9.1. Security

9.1.1. Security. Access to the system shall be password protected in software and firewall protected in hardware (implications for ACS TBD).

9.1.2. Tracing. Personal login with password should allow access to the system but the access of each individual should be logged for security.

9.1.3. Booking. There shall be some protection against users who try to access configurations of Distributed Objects reserved by other users.

9.2. Safety

9.2.1. All human and machine safety will have mechanical or other systems that will operate even in the face of malfunctioning software. Feedback to the affected software shall be provided, except in case of catastrophic failures. (implications for ACS TBD).

9.2.2. The use of software limits shall be supported by ACS. These are used to allow the software to stop potentially dangerous actions before hardware limits are reached, under normal software operation conditions.

9.3. Reliability

9.3.1. Robustness. Should a subsystem fail, ACS shall allow the rest of the system to continue operation, supporting the possibility to reconfigure without shutting down the system.

9.3.2. Backup. Backup copies of configuration Database information shall be supported.

9.3.3. Roll-back. A 'breakpoint' system with a standard roll-back should also be considered.

9.4. Performance

9.4.1. Performance. A minimum goal rate of **1000 messages** per sec (TBC) shall be supported in communicating between any two CPUs of any of the considered OS/RTOS.

9.4.2. Command reception shall be acknowledged immediately, i.e. **within a TBD time.**

9.4.3. User replies. When actions get started from a user interfaces there shall be either a positive or an error reply returned to the user **within 2 sec**. For actions requiring longer time, there shall be a way to see that they are on-going within the time specified above.

9.4.4. Panel updates. Data refreshed on screens shall arrive to the user **within 0.5 s**. This extends to users of the Control centre, but not to remote users.

9.4.5. System down time. Restarts and reboots shall be contained within a goal maximum time of 10 minutes. This defines also the maximum time for remote access interruptions.

10. Standards

10.1. Standards and products

10.1.1. Official or de-facto standards shall be used whenever possible.

10.1.2. The choice of products to be used in ACS shall be based on their compliance with standards. Reference products are given below following the corresponding standards.

10.1.3. Open source products shall be preferred over commercial products, when functionally satisfactory and if they represent a real proven alternative to commercial ones. The first ones have the advantage of long term safety and portability across platforms (e.g. observing tools).

10.2. Computer hardware standards

10.2.1. Scalable CPU architecture families shall be chosen for future upgrading.

10.2.2. There shall be a standard for real-time computers. Non time-critical computers might be either workstations or PCs.

10.3. Software standards

10.3.1. RTOS. A standard real-time operating system shall be supported by ACS.

10.3.2. High level operating system. ACS shall support Posix compatible Unix.

10.3.3. Compiled Languages. ACS shall primarily be implemented in JAVA and C++. Limited low level parts of ACS may be implemented in C.

10.3.4. IDL. All Distributed Object interfaces shall be written in CORBA/IDL.

10.4. Communication standards

10.4.1. CORBA. Communication on LANs and remote links shall be based on CORBA.

10.4.2. ORB independence. ACS shall be as insensitive as possible to the choice of ORB vendor and therefore it shall be based on POA rather than BOA.

10.4.3. LAN. The LANs connecting to the central computers shall be suitable for a connection over distances up to 20 Km.

10.4.4. Backbone. The central computers shall be connected with a high speed backbone (TBD).

10.4.5. Field-bus. There shall be a standard field-bus to be used on ALMA antennas.

10.5. Reference products

10.5.1. Real-time computers will be based on the PowerPC model 26xx for VME in Phase 1

10.5.2. High level computers may be either Sun SPARC workstations or x86-based PC compatibles.

10.5.3. RTOS. The RTOS for Phase 1 is VxWorks.

10.5.4. OS. The reference Unix operating systems will be Linux and Solaris (TBC for Phase 2).

10.5.5. Configuration DB. The relational database mySQL will be evaluated. It could be also that information is kept in files (e.g. XML files).

10.5.6. Scripting language. TCL and JPython will be evaluated for use with ACS, with the goal to choose one.

10.5.7. Analysis and plotting. Labview will be evaluated (based on WinNT).

10.5.8. CORBA ORB. TAO (with ACE) and Orbacus will be evaluated as candidate ORBs for communication among Distributed Objects. Final choice of ORBs TBD for Phase 2.

10.5.9. Data transfer. There shall be a feasibility check of CORBA in relation to astronomical data transfer.

10.5.10. The XML format will be evaluated to implement Distributed Objects Serialization.

10.5.11. LAN. The LANs connecting to the central computers will be either (Fast) Ethernet or ATM.

10.5.12. Field-bus. The standard field-bus to be used on ALMA is CAN bus.

11. Life cycle aspects (TBC by Software Engineering)

11.1.1. Prototyping. Any technology presenting a certain risk shall be prototyped before a final decision on its use is adopted. This is valid also for the initial choice of CORBA, given that suitable ORBs with appropriate performances have to be chosen.

11.1.2. Performance tuning. ACS shall be performance tuned on a computer model and then by using performance metrics on an existing telescope.

11.1.3. Methodology. The ACS software shall be developed using the same Case Tools methodology, to be used in the development of ALMA applications.

11.1.4. Releases. A formal Release cycle of 6 months shall be started with ACS and encompass the whole common software.

11.1.5. Test. Automatic regression tests shall exist for all releases.

11.1.6. Applications. Application groups will submit code of general utility to ACS for consideration of inclusion. Developers are responsible for the maintenance of their code to be provided together with test procedures.

11.1.7. Procedures. Standard software management tools (e.g. CVS) shall be applied. The same applies to coding and documentation standards, as soon as they will be defined by the Software Engineering Team.

12. Interface Requirements

12.1. User interface

12.1.1. Portability. User interfaces shall be platform independent, so that they can be run both centrally, locally and remotely.

12.1.2. Extended portability. Packages requiring porting to platforms external to ALMA (e.g. observing tools to be run on any Laptops) shall be identified, if existing, within ACS.

12.1.3. Tools. A standard set of tools for the development of user interfaces shall be provided, including: an interactive GUI builder, a standard set of widgets, standard support libraries for the integration with applications and ACS.

12.1.4. Modularity. User interfaces shall not implement any control logic, to mark the difference with control applications, but might need to contain some interaction logic (e.g. a button being blocked while some action occurs).

12.1.5. ACS GUIs. Specific panels shall allow to display sampled data both in quasi-real-time or off-line, alarms and logs inclusive of filtering and sorting capabilities.

12.1.6. Location. All data, logs and alarms shall be available both centrally and locally, to allow use in the antenna area, at the control centre and at remote locations.

12.2. Hardware interfaces

12.2.1. Hardware interfaces. These will be provided as part of the M &C software development and will be integrated into the ACS releases.

12.3. Software interfaces

12.3.1. The ACS shall be integrated and tested with any common open-source and commercial software, off-the-shelf software, legacy and embedded software used for ALMA.

13.ACS Design requirements

13.1.1. Distributed Objects and Commands. Commands shall be implemented via remote method invocations of Distributed Objects, based on the CORBA technology .

13.1.2. Standard methods. There shall be a list of standard methods for Distributed Objects that is supported by ACS.

13.1.3. Groups. It shall be possible to logically group Distributed Objects.

13.1.4. Events. ACS should allow attaching callbacks to events.

13.1.5. Configuration. It shall be possible to use ACS in a development set-up and in the real control situation (e.g. control model and real telescope), even in parallel.

13.1.6. Modularity. It shall be possible to build only the modified components of the system, without having to rebuild the entire system.

13.1.7. Portability. ACS shall be designed to be portable across the RTOS, Unix and Linux platforms for what concerns its upper level components. There must be parts though which are only applicable to the RTOS.

14.Requirements for applications

This is a temporary section, meant to be replaced later by a style guide document.

14.1. Style guide requirements

14.1.1. Logging of commands. All user commands shall be logged (to correlate in time relevant events to user activities).

14.1.2. Normal commands. They shall not assume deep knowledge of system and cannot lead to situations requiring recovery actions.

14.1.3. Maintenance commands. Not required during normal operation. Require deep knowledge of the subsystem to which they apply and might need a certain sequence of actions. Normally to be used by maintenance staff only.

14.1.4. All subsystems shall be monitored in a standardised way.

14.1.5. Reliability. System reliability shall be improved by implementing error recovery whenever reasonable. A trace of the recovery attempts shall exist in the logs, even when recovery succeeded.

14.1.6. Subsystems shall be monitored when in operation to notify malfunctions (with alarms) when they occur.

14.1.7. Warnings. The number of low severity alarms (warnings) shall be small during normal operation.

14.1.8. Disabled subsystems. It shall be possible to globally inhibit actions on part of the ALMA software.

14.1.9. Dynamic configuration. It shall be possible to change all parameters dynamically without resetting the ALMA software.

14.1.10. Simulation. Control software must be able to run in simulation mode, without requiring any normally associated hardware

14.1.11. Stand-alone. Subsystem software must be able to run in stand-alone mode, meaning without the rest of the ALMA system (e.g. sub-reflector being tested without antenna). This might require that other control software (interfaced with the considered subsystem) runs in Simulation.

14.1.12. Additions. Existing software, whose interfaces don't change, shall not need to be modified when new software components are added to the existing system.

14.1.13. States. Distributed Objects may have states (like off, standby, online). When States are used they should have standard names and meaning, with standard methods supported by all objects.