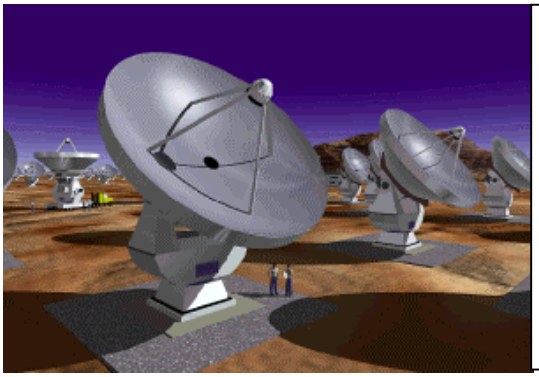

Atacama Large Millimeter Array

ALMA-SW-0016

Revision: 2.0

2001-09-26

ALMA Common Software Architecture

ALMA Common Software

G.Chiozzi, B.Gustafsson, B.Jeram
ESO

Keywords:	
Author Signature:	Date:
Approved by:	Signature:
Institute:	Date:
Released by:	Signature:
Institute:	Date:

Change Record

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
1.0/Prep.1	1999-11-20	G.Chiozzi	All
	First revision for working group internal review This first issue (Issue 1.0) of this document, called at the time *ALMA Common Software Feature List*, has been written before the *ALMA Common Software Technical Requirements* document[RD01] and has been used as an initial input for it.		
1.0/Prep.2	2000-01-15	G.Chiozzi	All
	Updated after group internal review		
1.1/Prep.1	2000-05-31	G.Chiozzi	All
	Updated after discussions with NRAO and meetings with G.Chiozzi, G.Raffi, and B.Glendenning. Document renamed from "ALMA Common Software Feature List" to "Architectural Discussion on ALMA Common Software Framework. Comparison with ESO Common Software moved in appendix. "Devices" renamed "Distributed objects" in order to keep them distinct from Control System devices.		
1.1/Prep.2	2000-06-10	G.Chiozzi	All
	Document renamed ALMA Common Software Architecture and converted into Architectural description. Updated after official release of ALMA Common Software Technical Requirements. Explicit definition of requirements has been removed assuming the *ALMA Common Software Technical Requirements* document[RD01] as an applicable document. Added traceability matrix with requirements document		
2.0/Prep.1	2001-04-11	G.Chiozzi et al.	All
	Updated including all comment to issue 1.1/Prep.2 and results of Kitt Peak test and the feedback from the first experiences in the usage of the ACS 0.0 prototype. Applied ALMA Documentation template.		
2.0/Prep.2	2001-09-10	G.Chiozzi et al.	All
	Updated taking into account document's review.		
2.0	2001-09-26	G.Chiozzi	Headers and footers
	Assigned document number and released officially		

Table of Contents

1 Introduction

1.1 Scope

This document proposes an architecture for the ALMA Common Software (ACS), taking as applicable the requirements specified in the *ALMA Common Software Technical Requirements* document[RD01].

This list of requirements is discussed in order to identify a possible architecture for a common software implementation able to satisfy the given requirements. This revision of the document takes into account the experience collected with the ACS V.0.0 prototype and the Kitt Peak 2000 test[RD21].

This document provides a complete picture of the desired ACS functionality for the entire development phase, but individual concepts and features will be developed incrementally over a number of releases, according to the Software Life Cycle described in [RD17]. For each release, a detailed plan will be developed, identifying the components to be added or revised. Development priorities will be discussed with the community of users during the planning phase of each release.

The current Architecture describes Common Software to be used mainly for writing control software. There will be most probably other need for data flow software, pipeline or proposal preparation that we do not understand yet, because not enough work has been done in that areas. We will then have to extend this document to cover also that needs in future versions.

1.2 Overview

ACS is located in 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)[RD01 - 3.1.1. Scope] and consists of software developed specifically for ACS and as well of OS builds and commercial device drivers. All code specifically developed for ACS is under the GNU General Public License (GPL). 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[RD01 - 3.1.2. Design]. The use of ACS software is mandatory in all applications, except when the requested functionality is not provided by ACS [RD01 - 3.1.3. Use]. 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[RD01 - 3.2.1. Users].

This document identifies the main packages that will be part of ACS and their high level interrelations. For each package, a section in this document respectively discusses the requirements to clarify them and presents an architectural concept.

Requirements are traced back to the *ALMA Common Software Technical Requirements* document[RD01] whenever they are referenced in the document and a Requirements Traceability Matrix provides a summary. Also requirements on ACS expressed in the TICS Design Concept document[RD26] are summarized in a Requirements Traceability Matrix.

The concept illustrated here is based on the use of CORBA and takes into account knowledge of various control software projects based on CORBA in the astronomical and High Energy Physics communities, like SOFIA[RD10], GTC-Spain[RD11], ESRF-Grenoble[RD03], ANKA-Kalsruhe[RD04] etc. It has been an initial and explicit decision of the ALMA project to use CORBA technology and at the same time to share software rather than to re-invent it. It is up to documents like this to provide elements to confirm the initial choice of CORBA as adequate. It is up to a later comparison among existing systems to find the right way for ALMA, which we assume in any way will imply a number of specific additional developments.

The reasons for using CORBA are in short: Object Orientation, support for distributed systems, platform independence, it is a communication standard, it provides a variety of services.

1.3 Reference Architecture

A reference layout for the system shall be provided by a planned Design Concept document[RD01 - 2.3. Reference Architecture]. The Architecture of the Test Interferometer is described in the TICS Design Concept document[RD26].

For the purposes of this document a distributed architecture based on computers at the individual antennas and a number of central computers, connected by an high speed backbone[RD01 - 10.4.3. LAN] [RD01 - 10.4.4. Backbone] [RD01 - 10.5.11. LAN], is assumed[RD02].

At both the antenna and the central control building there will be not only Ethernet LAN connectivity but also a Field-bus[RD01 - 10.4.5 Field-bus] (the AMB) [RD01 - 10.5.12. Field-bus] connected to various intelligent devices. The fact that the Antenna controller and all or part of these Devices is on Field-bus or LANs shall not make any difference in terms of the architecture proposed here.

1.4 Reference Documents

The reference documents contain background information required to fully understand the structure of this document, the terminology used, the software environment in which ALMA shall be integrated and the interface characteristics to the external systems.

The following documents are referenced in this document.

[RD01] ALMA Common Software Technical Requirements, ALMA-TRE-ESO-XXXXX-XXXX, G.Raffi, B.Glendenning, Issue 1.0, 2000-06-05

[RD02] ALMA Construction Project Book, Version 5.00, 2001-08-01 (http://www.mma.nrao.edu/projectbk/construction/)

[RD03] TANGO - an object oriented control system based on CORBA - J.M.Chaize et al., ICALEPCS'99 Conference, Trieste, IT, 1999 (http://www.elettra.trieste.it/ICALEPCS99/proceedings/papers/wa2i01.pdf)

[RD04] Implementing Distributed Controlled Objects with CORBA - M.Plesko, PCs and Particle Accelerator Control Workshop, DESY, Hamburg, 1996 (See http://kgb.ijs.si/KGB/articles.htm for this and other related papers).

[RD05] SOSH Conventions for Control - F.Di Majo C.Watson, Software Sharing (SOSH) for Accelerators & Physics Detectors (http://www.jlab.org/sosh/)

[RD06] Java Home Page - (http://java.sun.com/)

[RD07] Real-time CORBA with TAO (the ACE ORB) - (http://www.cs.wustl.edu/~schmidt/TAO.html)

[RD08] ObjectStore home page - (http://www.odi.com/objectstore/)

[RD09] MySQL home page - (http://www.mysql.com)

[RD10] SOFIA home page - (http://sofia.arc.nasa.gov/)

[RD11] GTC home page - (http://www.gtc.iac.es/)

[RD12] ALMA Monitor and Control Bus, Interface Specification, ALMA Computing Memo #7, M.Brooks, L.D'Addario, Rev.B 2001-02-05

[RD13] National Instruments LabVIEW - (http://www.ni.com/labview/)

[RD14] CORBA Telecom Log Service - (http://www.omg.org/technology/documents/formal/telecom_log_service.htm)

[RD15] omniORB Home Page - (http://www.uk.research.att.com/omniORB/)

[RD16] IBM VisualAge - (http://www-4.ibm.com/software/ad/)

[RD17] ALMA SE Practices - Software Development Process Methodology and Tools, G.Chiozzi, R.Karban, P.Sivera - (http://www.mma.nrao.edu/development/computing/docs/joint/draft/SE-SwDev.pdf)

[RD18] eXtensible Markup Language Home Page - (http://www.w3.org/XML/)

[RD19] Orbacus Home Page - (http://www.ooc.com/ob/)

[RD20] IBM DB2 Home Page - (http://www-4.ibm.com/software/data/db2/)

[RD21] ALMA ACS and AMS Kitt Peak 2000 Test , G.Chiozzi et al. (http://www.mma.nrao.edu/development/computing/docs/joint/notes/2000-12-KP.pdf)

[RD22] Design and Initial Implementation of Diagnostic and Error Reporting System of SMA, SMA Technical Memo 132, Q.Zhang.

[RD23] The Adaptive Communication Environment (ACE) home page - (http://www.cs.wustl.edu/~schmidt/ACE.html)

[RD24] Python language home page - (http://www.python.org/)

[RD25] Home Page for the Official Tcl/Tk Contributed Sources Archive - (http://www.neosoft.com/tcl/)

[RD26] Test Interferometer Control Software Design Concept, B.Glendenning et al., DRAFT 2001-02-15

[RD27] Advanced CORBA Programming with C++, M.Henning S.Vinoski, Addison-Wesley, 1999

[RD28] ALMA Software Glossary
(http://www.alma.nrao.edu/development/computing/docs/joint/draft/Glossary.htm)

[RD29] AMI/ACS Report, R. Lemke, G. Chiozzi 2001-03-20
(http://www.eso.org./~gchiozzi/AlmaAcs/examples/amitest/AmiReport.pdf)

[RD30] ALMA Memo #298, Timing and Synchronization, L. D'Addario, 2000-03-09
(http://www.alma.nrao.edu/memos/html-memos/alma298/memo298.pdf)

[RD31] GNU General Public License (GPL) (http://www.fsf.org/copyleft/gpl.html)

1.5 Glossary

An extended list of glossary definitions, abbreviations and acronyms is part of the main ALMA Software Glossary [RD28], available online at the following URL: http://www.mma.nrao.edu/development/computing/docs/joint/draft/Glossary.htm.

The following list of abbreviations and acronyms is aimed to help the reader in recalling the extended meaning of the most important short expressions used in this document:

ABM	Antenna Bus Master
ACE	ADAPTIVE Communication Environment (http://www.cs.wustl.edu/~schmidt/ACE.html)
ACS	ALMA Common Software
ACU	Antenna Control Unit
AIPS++	Astronomical Information Processing System (http://aips2.nrao.edu/docs/aips++.html)
ALMA	Atacama Large Millimeter Array (http://www.eso.org/projects/alma/)
AMB	ALMA Monitor and Control Bus
ANKA	Synchrotron Radiation Source ANKA (http://www.fzk.de/anka)
API	Application Programmatic Interface
CAN	Controller Area Network
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
ESO	European Southern Observatory (http://www.eso.org)
FITS	Flexible Image Transport Format
GUI	Graphical User Interface
GTC	Gran Telescopio CANARIAS (http://www.gtc.iac.es/)
HW	Hardware
IDL	CORBA Interface Definition Language
IIOP	Internet Inter-ORB Protocol
ISO	International Standardization Organisation
JDBC	Java Database Connectivity
LAN	Local Area Network
LCU	Local Control Unit
M&C	Monitor and Control
N/A	Not Applicable
NRAO	National Radio Astronomy Observatory (http://www.nrao.edu/)
OMG	Object Management Group (http://www.omg.org/)

ORB	Object Request Broker
OSI	Open Systems Interconnection
OVRO	Owens Valley Radio Observatory (http://www.ovro.caltech.edu/)
RDBMS	Relational Data Base Management System
RPC	Remote Procedure Call
SLA	Subprogram Library A (Positional Astronomy Library)
SW	Software
TAO	The ACE ORB (http://www.cs.wustl.edu/~schmidt/TAO.html)
TBC	To Be Confirmed
TBD	To Be Defined
TCL	Tool Command Language (http://www.scriptics.com/resource/)
TCL (CORBA)	CORBA Trader Constraint Language
TICS	ALMA Test Interferometer Control Software
TPOINT	Telescope Pointing Analysis System
UML	Unified Modeling Language
URI	Uniform Resource Identifier (http://www.w3.org/Addressing/)
URL	Uniform Resource Locator (http://www.w3.org/Addressing/)
UTC	Universal Time Coordinated
VME	Versa Module Eurocard
VLT	Very Large Telescope
WS	Workstation
XML	eXtensible Markup Language (http://www.w3.org/XML/)

2 ACS Basic Architecture

2.1 Overview

The ALMA Common Software (ACS) is located in between the ALMA application software (Applications) and other basic commercial or shared software on top of the operating systems. In particular, ACS is based on CORBA (CORBA Middleware), which provides the whole infrastructure for the exchange of messages between distributed objects. Whenever possible, ACS features will be provided using off the shelf components and ACS itself will provide the packaging and the glue between these components.

The ACS is also based on an Object Oriented architecture [RD01 - 13.1.1 Distributed Objects and commands].

The following UML Package Diagram shows the main packages in which ACS has been subdivided.

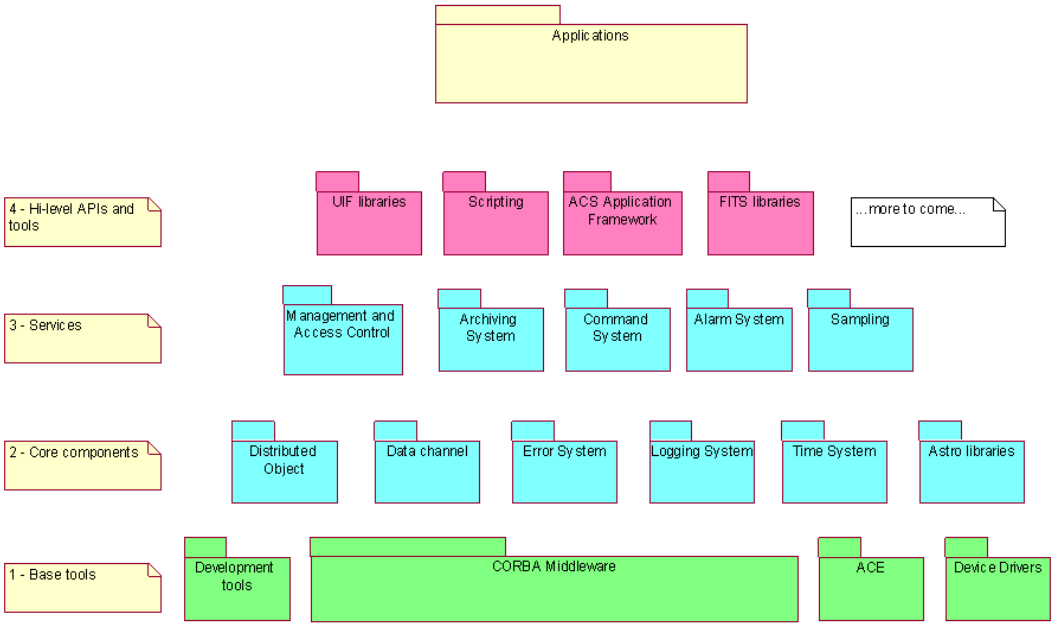

Figure 2.1: ACS Packages

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

Packages have been grouped in layers. Packages are allowed to use services provided by other packages on the lower layers and on the same layer, but not on higher layers.

A brief description of the layers and the packages if provided hereafter, while the next chapter will contain a detailed description of the feature included in the packages.

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 just provides packaging and installation and distribution support. This ensure that all installations of ACS (development and run-time) will have the same basic set of tools with versions kept under configuration control.

Three main packages have been identified in this layer:

- **Development tools**
  Software development tools (compilers, configuration controls tools, languages, debuggers, documentation tools).

- **CORBA Middleware**
  Packaging of off-the-shelf CORBA implementations (ORB and services) to cover the languages and operating systems supported by ACS.

- **ACE**
  Distribution of the Adaptive Communication Environment[RD23].

2 - Core components

This second layer provides essential components that are necessary for the development of any application

- **Distributed Object**
  Base interfaces and classes for Distributed Object, Properties and Characteristics are implemented in this package.

- **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.

- **Time System**
  Time and synchronization services.

- **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 of data, actions and events. Transport of logs from the producer to the central archive. Tools for browsing logs.

- **Astronomical libraries**
  Libraries for astronomical calculations.

3 - Services

The third layer implements services that are not strictly necessary for the development of prototypes and test applications or that are meant to allow optimization of the performances of the system:

- **Management and access control**
  Design patterns, protocols and high level services for Distributed Object's life-cycle management.

- **Archiving System**
  API and services for archiving monitoring data and events from the run time system. Tools to browse, monitor and administer the flow of data toward the archive.

- **Command System**
  Tools for the definition of commands, API for run-time command syntax checking, API and tools for dynamic command invocation.

- **Alarm System**
  API and tools for configuration of hierarchical alarm conditions, API for requesting notification of alarms at the application level, tools for displaying and handling the list of active alarms.

- **Sampling**
  Low level engine and high level tools for fast data sampling (virtual oscilloscope).

4 - API and High-level tools

The fourth and last layer provides high level APIs and tools. More will be added in the future. The main goals for this packages is to offer a clear path for the implementation of applications, with the goal of obtaining implicit conformity to design standards and maintainable software[RD01 - 3.1.2. Design].

- **UIF Libraries**
  Development tools and widget libraries for User Interface development .

- **Scripting**
  Scripting language and access libraries for the integration with ACS core components.

- **ACS Application Framework**
  Implementation of design patterns and to allow the development of standard applications.

- **FITS libraries**
  Support for the handling of FITS files is just an example of other high-level components that will be integrated and/or distributed as part of ACS.

2.2 Deployment

The choice of CORBA for the implementation of Distributed Objects and of all services that are part of the previously described packages makes it possible to have every software operation available in a transparent way both locally and at the Control center in San Pedro. This applies also to all data, logs and alarms[RD01 - 12.1.6 Location]. The choice for the LAN and access optimization mechanisms, described in the following sections, will ensure that no significant degradation in performances will take place [RD01 - 3.2.4. Local and central operation].

In principle, this same mechanism allows a reliable remote access from the US and Europe, although with reduced performance. It is anyway necessary that applications are designed in order to prevent unauthorized access and undesired side effects on the performance of the control system [RD01 - 3.2.5. Remote access]. ACS provides the basic building blocks for the implementation of these mechanisms.

All components in the lower ACS layers will be available for developing code both for the Linux[RD01 - 10.5.4 OS] and the VxWorks[RD01 - 10.5.3 RTOS] platforms. The usage of ACE allows writing portable code that can migrate from Linux to VxWorks and vice versa according to development and run-time needs.

Higher-level components will be often available only for Linux, since they are not needed on the real-time platform.

Some development tools can be required to run on Windows platforms (like Visual Age for Java [RD16]).

The usage of platform independent UIF tools allows executing user interfaces on multiple platforms, including Windows.

Using CORBA, all objects publishing an IDL interface will be available to any environment, host and programming language where a CORBA implementation is available. In particular it will be possible to write client applications for Distributed Objects in any CORBA-aware platform. ACS explicitly supports C++, Java, C and

Python (selected by TICS with preference to TCL) [RD01 - 10.3.3. Compiled Languages] [RD01 - 10.5.6. Scripting language].

On the other hand, most servant-side ACS libraries will be developed first in C++, with the assumption that servants for Distributed Objects will run typically on VxWorks and that servants have anyway higher performance requirement than clients. It is possible to implement servants in other languages or for other platforms, but this requires a complete servant side implementation of the ACS basic IDL interfaces for the Distributed Object - Property - Characteristic pattern. This can be handled at application level or, better, a specific extension of ACS to provide support for servants in specific languages other than C++ can be requested when the need for it will be clearer.

The ACS installation procedures will allow selecting the installation platform and will allow selecting between development and run time installations. For a development installation, all development tools will be installed, including compilers and debuggers, while a run-time installation will be much lighter and include only the libraries and components needed at run-time.

Per each package, it will be specified at design time what components will be available on each platform and for run-time and development installations.

We foresee 6 different types of deployment nodes (connections in the diagram show the foreseen communication paths among node types, for example a Remote User Station is allowed to communicate only with a Linux run-time workstation):

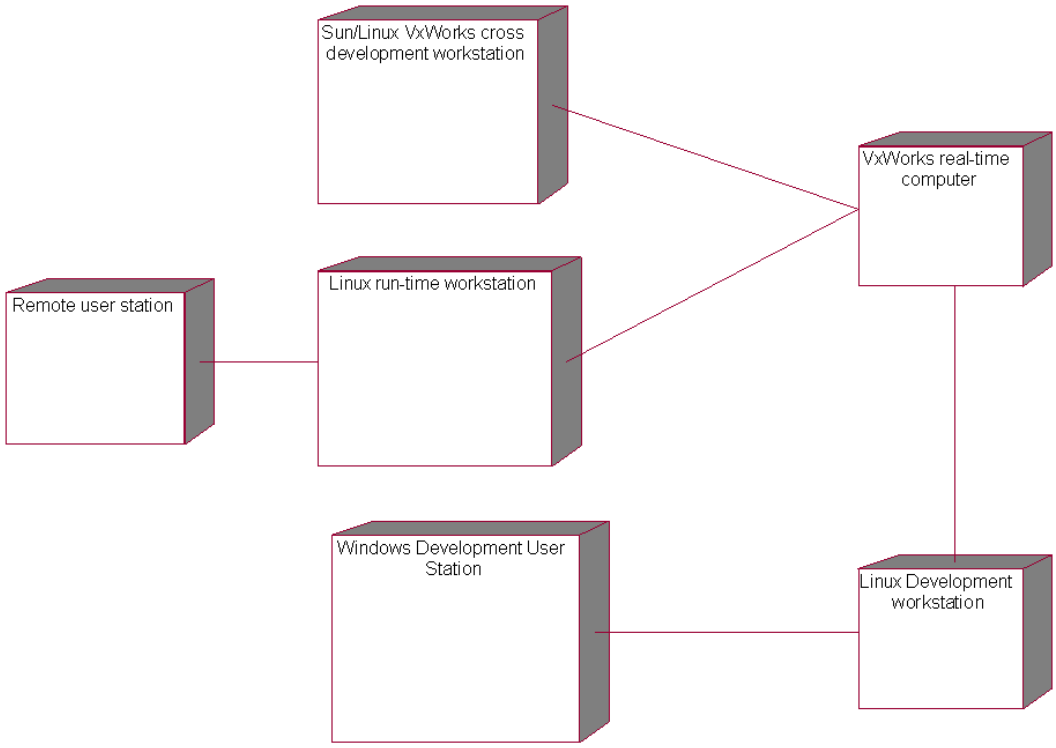

Figure 2.2: ACS Deployment

- **Linux Development workstation**
  Linux is the main development and run-time platform.
  This installation includes all ACS components necessary for development and at run-time.

- **Windows Development User Station**
  This is the console where a software developer is working to develop UIF applications using Windows-based tools.
  Development of user interfaces is generally more efficient on Windows systems and development tools are more reliable and stable with respect to Linux.
  An ACS Windows Development installation will allow installing UIF development tools and UIF libraries for development, including CORBA ORB and services.
  We assume that an ACS developer will have a Windows desktop used for UIF development and as a terminal to connect to a Linux Development Workstation. This does not exclude user having access only to Linux workstations but limit their capability of developing UIF applications.

- **VxWorks real-time computer**
  VxWorks computers are used only as run time platforms and a cross-development computer is necessary to develop code.
  ACS will deploy on VxWorks only run time libraries, which are downloaded from a file server at boot time or when needed.

- **Sun/Linux VxWorks cross development workstation**
  The VxWorks cross development environment is installed on a UNIX (Sun or Linux) workstation. This includes all development tools and ACS components. Typically the same development workstation should be used for Linux and VxWorks development.

- **Linux run-time workstation**
  When disk space is an issue for small run-time only boxes, it will be possible to deploy on ACS runtime libraries and components. The main limitation of this configuration is that it offers very poor debugging capabilities in case of problems, with respect to a full development installation

- **Remote user station**
  A very light ACS installation will be provided for users that need only to access remotely other ALMA subsystems using only UIF applications. This will allow installing only UIF run time libraries and applications (including CORBA ORB and services) on Linux (or Windows) operating system.

3 ACS Packages

This chapter contains one section per each ACS package, describing the features provided and its high level architecture.

The packages are described, looking at the package diagram in Chapter 2, starting from the lower layer and from left to right.

For some packages, just a brief description is given. In particular this applies to packages that are just the integration of off the shelf components.

3.1 Development tools

Packaging of tools necessary for the whole life cycle of software developed with ACS. This includes for example compilers, configuration controls tools, languages, debuggers, documentation tools. The complete list will be defined at each release. ACS will assume a specific set of supported Operating Systems and versions.

3.2 CORBA Middleware

Packaging of off-the-shelf CORBA implementations (ORB and services) to cover the languages and operating systems supported by ACS.

This includes also all CORBA Services used by ACS.

3.3 ACE

Distribution of the Adaptive Communication Environment[RD23]. This C++ class library provides support for cross-platform portability.

3.4 Device Drivers

3.4.1 Device drivers for all standard boards (motor control, analog, digital, serial…) and network protocols (serial communication, CAN bus[RD01 - 10.5.12 Field-bus],…) are provided as part of the ACS delivery, if they are not already part of the operating system, although development of device drivers is not direct responsibility of ACS. They will be typically provided as part of M&C software development and integrated into the ACS releases[RD01 - 12.2.1 Hardware Interfaces]. This integration must be taken into account both in the design of ACS and of Device Drivers.

3.4.2 A communication library working over CAN is already existing as part of the M&C work[RD12] and has been harmonized with these concepts. For every CAN device there must be one or more Objects on the CAN bus master CPU, which are the abstract Distributed Object(s) corresponding to the physical CAN device.

3.4.3 Low level access to the CAN bus nodes will be completely hidden to the general user (engineering applications can directly access the CAN nodes).

3.5 Distributed Object

The requirement document [RD01] specifies as a basic design requirement the adoption of an Object Oriented architecture based on distributed objects [RD01 - 13.1.1 Distributed Objects and commands]. This concept is the basis of the architecture and all services provided by ACS are designed around Distributed Objects.

3.5.1 The ALMA system (and the control system in particular) is described using a 3 tier naming for the logical model [RD03] [RD04] [RD05]:

- Distributed Object
- Property
- Characteristic

3.5.1.1 Distributed Object - Instances of classes identified at design level in the ALMA system, with which other components of the system interact, are implemented as Distributed Objects. In particular, at control system level, Distributed Object is the base class used for the representation of any physical (a temperature sensor, a motor) or logical device in the control system.

3.5.1.2 Property - Each Distributed Object has 0..n Properties that are monitored and controlled, for example status, position, velocity and electric current.

3.5.1.2.1 Properties can be read-only or read/write. If a read/write property cannot read its value back (for example it is associated to a write only physical device), it caches the last written value and returns this upon read request. This implementation is mandatory and must be documented in the property documentation.

3.5.1.2.2 Properties can represent values using a limited set of basic data types:

- Long for integers
- Double for floating point numbers
- String for strings.
- Boolean for TRUE/FALSE conditions
- Bitfield to handle patterns of bits, typically from hardware devices
- Complex for handling complex numbers
- Enum for enumerations like states.
- Sequence<scalar> of one of the previously defined scalar types. A Sequence<scalar> is a set of properties of a given scalar type, i.e. each item in the sequence is a complete property of the given scalar type. It is implemented as an IDL Sequence of the scalar property type. For example a Sequence<Long> allows manipulating a group of properties of type Long. Each item in the list can be assigned to a Long property object and manipulated (reading characteristics and value) independently from the others.
- Array<scalar> of one of the previously defined scalar types. An Array<scalar> is a property type that contains as value a fixed length array of values handled by the corresponding scalar type. For example, an Array<Long> is a property type with a unique set of characteristics that apply to an array of integers. It is a unique property and its value is an array of values. With respect to Sequence<scalar>, Array<scalar> is much more efficient for transporting big tables of data.
- Structures built with properties of the other basic types. Since structures introduce a significant increase of complexity in the handling libraries, they will be implemented last and only if a clear need arises.

3.5.1.2.3 The selection of a limited set of type is motivated by the need of avoiding implementing the same code for many different types and conversion problems between similar types (like short, int and long). Also, nowadays saving a couple of bytes using a short instead of a long usually does not pay:

- it introduces performance problems (CPUs now always works with longs and every operation on a short requires a conversion to long)
- it introduces alignment problems in memory and when transferring structures from different platform

3.5.1.3 Characteristic - Static data associated with a Distributed Object or with a Property, including meta-data such as *name, description, version* and *dimensions*, and other data such as *units, range* or *resolution.* Each Distributed Object or each Property has 0..n Characteristics. An initial list of Characteristics for Distributed Objects and Properties will be agreed at design time. For the time being we have identified the following candidates (more details will be given in the design documentation):

- Distributed Objects and Properties: Name, Description, Version and URI of extended documentation, where the last is optional and would point to documentation generated automatically from the source code.
- Read-only and Read/Write Properties: default values, range, units, format, resolution

3.5.1.4 The following diagram shows an architectural class diagram for the Distributed Object (DO class) - Property - Characteristic pattern

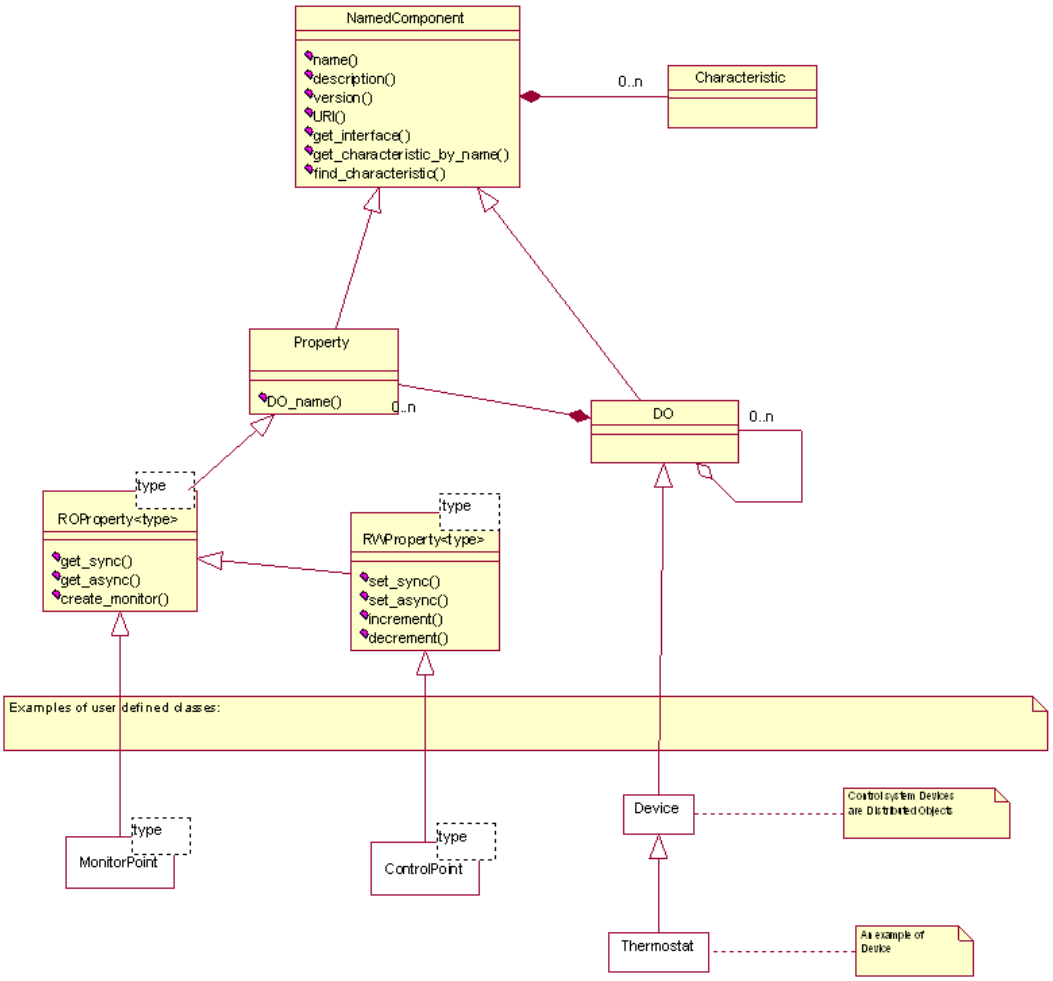

Figure 3.1: Distributed Object - Property - Characteristic class diagram

- A **NamedComponent** base class groups methods and attributes common to both Property and DO. In particular, both have a common set of Characteristics and provide the related access methods.

- A **DO** can reference other **DOs**, to build a hierarchical structure

- **Properties** are always contained into a **DO**. This means that a **DO** can contain 0 or many **Property** instances, while a property is always contained in one and only on **DO**. The **Property** class provides a method to retrieve the reference to the DO that contains it.

- From the base **Property** class, subclasses for each read only and read/write types are derived. This is represented in the diagram by the ROProperty<type> and RWProperty<type> parametrized classes. From an architectural point of view, RWProperty<type> classes are subclasses of the corresponding ROProperty<type>.

- The lower part of the diagram (white class boxes) shows how applications will inherit from the base classes provided by ACS. The example shows classes used for the implementation of the control system.

- This diagram is sufficient and correct at architecture level. At design level, it will probably be more convenient to introduce some intermediate class levels to improve the modularity of the code. It is also under discussion if the RWProperty<type> classes are better implemented as subclasses of ROProperty<type> or parallel to it in the hierarchy. These intermediate classes will be in any case hidden to the users, making the actual structure between the Property class and the implementation ov RO and RW properties an implementation detail.

3.5.1.5 The Distributed Objects - Properties - Characteristics 3 tier logical model is very well established in the community of Control Systems for Physics Experiments [RD03] [RD04] [RD05], where the name Device is used to identify what we call here Distributed Object. We prefer to adopt the more generic name, as specified in [RD01 - 13.1.1 Distributed Objects and Commands], because the usage of the ACS is not limited to the realm of the Control System, as in the case of the mentioned references. It provides instead generic services for the development of the whole ALMA software. Proper Device classes will be implemented by the Control System development team based on Distributed Objects.

3.5.2 The architecture of the system will be based on CORBA. There are various CORBA-based implementations of the 3 tier logical model that can be reused.

3.5.2.1 A Distributed Object will be a CORBA object

3.5.2.2 A Property will be CORBA object. A class hierarchy with Property class as root implements the basic read-only and read/write versions for the predefined types. This hierarchy provides standard IDL interfaces that shall be used by all clients to access Properties. On the implementation (servant) side, specific subclasses will provide polymorphic access to specific implementations like 'logical', 'simulated', 'CAN', 'RS232', 'Digital IO' and so on.

3.5.2.3 Public interfaces to Distributed Objects and Properties will be defined as CORBA IDL.

3.5.2.4 Characteristics of Distributed Objects and Properties can be accessed through access methods (as shown in figure) and through a generic value = get_characteristic_by_name(name) type of interface at run time. The interface of properties is defined by their IDL and the IDL is the same independently from the implementation (logical, CAN...). But specific implementations will have also specific characteristics. For example a CANLong property has a CANID characteristic. This means that from the property's IDL there is no way to retrieve the CANID using CORBA calls. We provide then a generic interface that can be used to retrieve any characteristic just querying by name. This allows accessing specific Characteristics, like the CAN ID for properties related to CAN monitor points that are not defined in the generic property IDL but are instead implementation specific.

3.5.3 The configuration parameters for all Distributed Objects, i.e. the initial values for Properties control values and all Characteristics for Properties, are persistently stored in the Configuration Database [RD01 - 4.2.1. Configuration Database].

3.5.3.1 When a Distributed Objects is instantiated, it configures itself according to the configuration stored in the Configuration Database[RD01 - 3.3.2. Serialization].

3.5.3.2 The implementation of the Configuration Database is hidden in a Configuration Database Access API to allow switching among alternative implementations[RD01 - 4.2.2 Database Design]. The reference implementation will be based on an RDBMS. IBM DB2[RD20] is the platform that will be tested for TICS development instead of mySQL[RD01 - 10.5.5 Configuration DB].

3.5.3.3 All DOs (or, more precisely, all NamedComponents) in the same process space will have access to a global reference to the Configuration Database they will use at instantiation time to retrieve their configuration information. At a higher level, the Activator responsible for the DOs (see Management and Access Control package) will provide an interface to set the reference to the configuration database used. In this way it is also easy to switch between different databases at startup time.

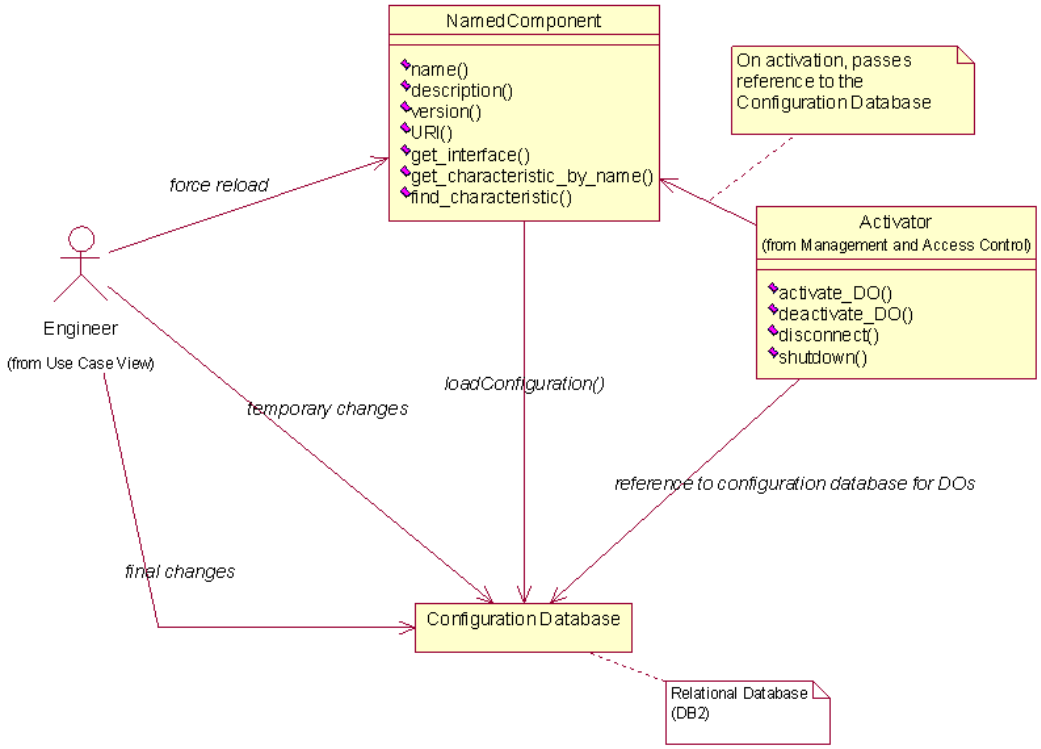

Figure 3.2: Configuration Database

3.5.3.4 An approach that satisfies the TICS' requirements of cold, warm or hot boot is:

- The system configuration comes from a reference configuration database.
- An application listens to the monitor stream and creates a snapshot configuration database that contains current monitor values.
- When a computer reboots, the reference DB and the snapshot DB are merged according to if it is a cold, warm, or hot boot. To configure the degree, to which defaults are overridden, another DB can keep a list of values that are overridden from the snapshot.

- To allow for user modified values during testing and tuning; a third DB with user preferences is kept.
- The scenario to boot a configuration is to merge the snapshot with the user preferences, and then this is merged with the reference DB according to the type of boot requested. The final configuration database is then passed to the Activator responsible for the DOs (see Management and Access Control package).
- A database administration tool allows also generating copies of parts of the database to be used for testing or when disconnected from the central systems, like when doing maintenance on a single disconnected antenna.

3.5.3.5 We define Characteristics as statically defined in the Configuration Database. This means that is possible to change their value only by changing the configuration database and reloading just the corresponding Distributed Objects[RD01 - 14.1.9 Dynamic configuration]. This means that they cannot be used for, e.g., calibrations that change at run time. With this definition, calibration values that change at run time should be implemented as Properties, not as Characteristics. "Static" calibration values, for example measured by an engineer that are not supposed to change for months can be Characteristics. Characteristics can change with time or can change with the context. We will have to verify if some more dynamic mechanism is necessary. This can be done transparently at a later time extending the Property class with methods to change the value of characteristics but will not be considered in the first design to avoid increasing the complexity of the system. This point will be reviewed after the TICS development.

3.5.4 Distributed Objects may have a state. Specific distributed objects can have additional sub-states. [RD01 - 13.1.2 Standard Methods] [RD01 - 14.1.13 States]. A standard state machine and standard state transition commands will be defined at a later stage, based on the TICS experience.

3.5.5 Means will be provided to have serializable Distributed Objects for creating a persistent image of the system. CORBA defines a persistency service, but few ORBs implement it. A persistent object oriented database can be used, like ObjectStore[RD08]. Another possibility would be to use XML to serialize objects [RD01 - 10.5.10 XML]. Other possibilities have to be investigated. The usage of a persistency service, of an object oriented database or of other solutions would be hidden inside the implementation of the Distributed Object sub-classes. [RD01 - 3.3.2. Serialization].

3.5.6 The Serialization mechanism will also be used to support migration of Distributed Objects from one application/sub-system to another one in different phases of their lifetime [RD01 - 3.3.3. Migration]. Inheritance will be used to add this capability for serializable DOs when needed.

3.5.7 JavaBeans wrap CORBA objects on the client side. Standard Rapid Application Development (RAD) tools like IBM VisualAge[RD16] are used to handle them. Given the IDL interface of a DO, a code generator produces automatically the corresponding JavaBean. In this way the developer has libraries that provide him direct support for ACS concepts like DO/Property/Characteristic, Monitors, Data Channel and so on.

3.5.8 ORB independence and interoperability [RD01 - 10.4.2 ORB Independence] is ensured basing the Distributed Object implementation on CORBA Inter-ORB Protocol (IIOP) and

Portable Object Adapter (POA) and not allowing the use of any ORB-specific feature. Interoperability between ORBs has been demonstrated on the Kitt Peak Test with TAO[RD07] and Orbacus[RD19] [RD01 - 10.5.8 CORBA ORB]. The selection of the final ORBs needed for ACS is not part of this Architecture document. The current baseline includes TAO[RD07] for C/C++, ORBACUS[RD19] for Java and omniORB[RD15] for Python bindings.

3.5.9 ACS provides support for simulation[RD01 - 3.3.4. Simulation] at property level. All properties that access hardware can be switched in simulation by setting TRUE a simulation characteristic in the configuration database. After this, they behave like "logical properties". This provides basic simulation capabilities

3.5.10 If an application wants to provide a more sophisticated level of simulation (for example simulating interrelations between the values of properties), a specific simulated device should be implemented in parallel to the real device. Switching from the real to the simulated device is handled in the configuration of the Manager (see Management and Access Control section), telling it to start a different implementation of the same device's CORBA interface.

3.5.11 Direct Value Retrieval

3.5.11.1 The Property classes provide get() and, in case of writeable Properties, set() methods that can be used to directly access the value of the property from clients [RD01 - 4.1.1 Direct value retrieval]. Both synchronous and asynchronous get() and set() methods are provided.

3.5.11.2 Value setting is done using set() property methods. These methods can be called by applications or by specifically designed GUIs. CORBA Dynamic Invocation Interface allows to write generic applications and GUIs (like the *Object Explorer*) that are capable of resolving dynamically at run time the structure of Distributed Objects and call set() methods to set the value of Properties [RD01 - 3.2.3. Value setting].

3.5.11.3 Inheritance will be used to implement property types accessing specific hardware devices, like CAN, RS232, GPIB. CAN properties will always directly access the hardware on the CAN bus at direct value retrieval and not use cached values.

3.5.12 Value Retrieval by Event

3.5.12.1 The Distributed Object will provide a method to create a monitor object for a Property, able to trigger events on Property change or periodically. A callback will be connected to the event and will be called by the monitor object when the specified event occurs[RD01 - 13.1.4. Events]. Triggered events are delivered directly to the registered object via the callback mechanism in a point-to-point fashion. The value retrieval by event is then very well suited for providing timely feedback to control applications, as described also in the TICS design document.

3.5.12.2 Events can be generated on any change of value[RD01 - 4.1.3 Rate]. Other conditions, for example any write, value increase/decrease, value less or greater than setpoint could also be included at a later stage.

3.5.12.3 Timed or periodic events can be generated as follows:

- Periodic, at a specific interval rate [RD01 - 4.1.3 Rate]

- Periodic, at a specific interval rate, synchronized with an absolute array time [RD01 - 4.1.5 Values at given time]. This also allows periodic events aligned with the monitoring rate. For example, a 1-second rate generates events on the 1-second mark, 5-second rate on the 5-second mark and so on.

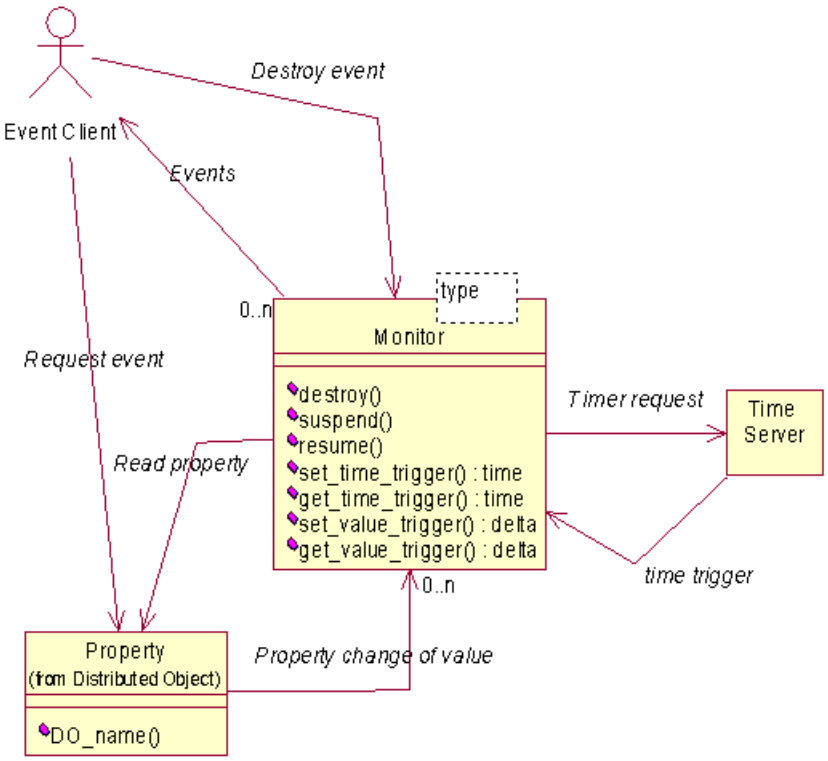

Figure 3.3: Value Retrieval by Event: ACS Monitors

3.5.12.4 All events will be time stamped with the time the value has been acquired (as opposed to the time of delivery of the event). Timers will warranty the time of acquisition of the value and not the time of delivery, that depends on the network characteristics.

3.5.12.5 The monitor class provides methods to suspend, resume and destroy the monitor itself

3.5.12.6 CAN-Properties will have to implement notification on change also for CAN monitor points although CAN monitor points do not provide a specific support via HW or in the drivers. This can/should be done via polling. If there are clients registered on events on change, an ACS monitor is used to poll and to generate events in case of change of the value of the monitor point. The poll rate is defined by the characteristic change frequency. The polling frequency determines the time resolution of the event-on-change.

3.5.12.7 For performance optimization, the final implementation will not leave to the single Distributed Object Properties the responsibility of managing timers, but a local centralized manager will take care of that, transparently to client applications. More details will be given in the ACS design.

3.5.12.8 A particular case is a Distributed Object State Machine. A State Machine class is a Distributed Object and the current state is represented by a State Property. This State Property can fire events whenever the state changes to allow external objects to monitor it.

3.6 Data Channel

3.6.1 The Data Channel provides a generic mechanism to asynchronously pass information between data publishers and data subscribers, in a many-to-many relation scheme.

3.6.2 With the Data Channel:

- the data producer *publishes* its data *pushing* it on the channel, completely unaware of clients getting access to the data, i.e. the data producer decides how and when data is going to be published
- data consumers *subscribe* to data sets on the channel without establishing any direct communication with the data producers.

3.6.3 The Data Channel is the basic mechanism for Indirect Value Retrieval [RD01 - 4.1.2 Indirect value retrieval] providing mirroring of data on other computers than where the data are produced. This makes it possible to randomly access data without interfering with the control process [RD01 - 3.2.2 Value retrieval] and without knowing if the data is directly available on the client's machine or if it is a mirrored copy[RD01 - 4.1.4 Transparency].

3.6.4 The CORBA Notification Service provides the infrastructure for the Data Channel implementation:

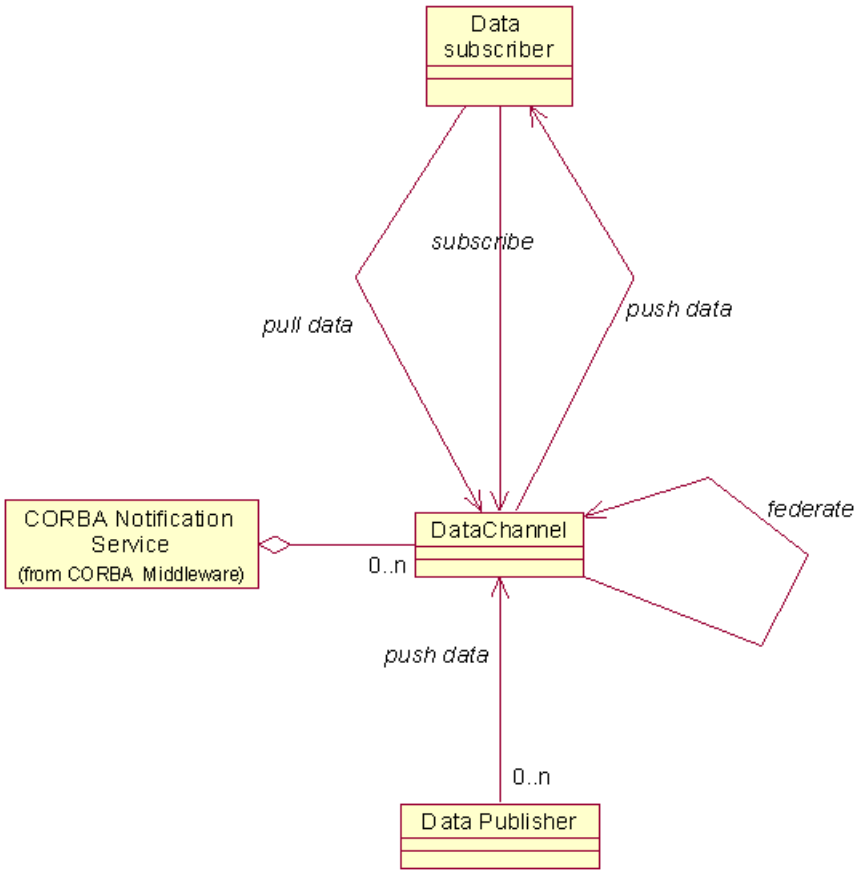

Figure 3.4: Data Channel

3.6.4.1 An ACS API provides a simplified client and server API to connect to the Notification Service and to create/open/close Data Channels. This does not hinder direct access to the CORBA Notification Service to access features not implemented by the API.

3.6.4.2 CORBA TCL (Trader Constraint Language) query language is used to allow filtering of messages from clients

3.6.4.3 Notification Service servers can be federated to guarantee system redundancy and to provide higher reliability. Federated Notification Service servers allow:

- Load balancing. Clients access can be split among different servers
- Security. Just specific servers, with a reduced set of published data (defined using filtering), can be allowed access from remote sites. This can be used to allow remote monitoring of ALMA from Europe and USA without exposing to the Internet confidential data.

3.6.4.4 The data channel shall also support the transport of scientific data. The performance of the data channel for transport of bulk data needs to be tested to verify if the requirements for scientific data movements are fulfilled.

3.6.5 The Data Channel is a process separated both from publisher and subscriber. It also optimizes data transfer by implementing caching to reduce network traffic.

3.6.6 Applications can create specific Data Channels to publish-subscribe to application specific data. ACS itself uses the Data Channel to provide basic services like logging and archiving.

3.6.7 Comparison between Data Channel, Direct Values Retrieval and Data Retrieval by Event:

3.6.7.1 The 3 data access mechanism provided by ACS have different characteristics and are meant to be used in different situation.

- With the Data Channel, data subscriber and data publisher are completely de-coupled.
- With Direct Value Retrieval, the client needs the reference to the servant object to call get() methods that directly return the requested value
- With Value Retrieval by Event, the client establishes a callback-based direct communication with the servant that asynchronously delivers data to the client in a point-to-point communication scheme.

3.6.7.2 The Data Channel pattern is convenient when:

- the client is interested in receiving events of a certain type (for instance event logs, monitor point values or alarm events) and handling them regardless of their source. Since the potential number of sources is very large, it becomes very inefficient if the client must establish a connection to each potential source. In this case a Data Channel is necessary as the mediator between publishers and subscribers. Publishers push data in the channel while the channel efficiently multicasts events to all subscribers that are interested in receiving them.
- many clients (in particular remote clients) are interested in the same data. With a point-to-point communication, the data producer would have to deliver data to each of many clients, with a potentially heavy impact on servant performances. With the Data Channel pattern, the servant pushes the data only once on the channel that will efficiently multicast the events to the interested clients allowing to keep constant the load on the servant.

3.6.7.3 Direct Value Retrieval and Value Retrieval by Event are convenient when the client needs to be in control of the rate by which it receives data from the servant (asking directly for the values, when required, or establishing monitors with a specific data rate or triggering condition.

3.7 Error System

The Error System provides the basic mechanism for applications to handle internal errors and to propagate from server to client information related to the failure of a request. An error can be notified to the final user and appear in a GUI window if an action initiated by the user fails (for example a command activated from a GUI fails).

3.7.1 The basic error reporting mechanism is to throw an exception.

3.7.2 Errors can be propagated through the call chain as exceptions and through objects over the network to be reported to the action requester. [RD01 - 6.3.6 Scope]

3.7.3 The lowest level of the call chain where the error occurs will add an entry in the error stack. When the error stack is propagated back through the call chain any level can add entries to the stack to provide a trace of the error. [RD01 - 6.3.2 Tracing]. Not all levels need to add an entry in the stack, but only the ones that provide useful information.

3.7.4 At any level it is possible to:

3.7.4.1 Fully recover the error. The error stack is destroyed and no trace remains in the system.

3.7.4.2 Propagate the error to the higher level, adding local details to the stack trace (including object and line of code where the error occurred). The higher level has the responsibility of handling the error condition

3.7.4.3 Close the error, logging the whole stack in the Logging System. The error could not be recovered at the higher level responsible for handling it and it goes in the log of anomalous conditions for operator notification and later analysis. This typically happens for unrecoverable errors or when an error has to be recorded for maintenance/debugging purposes. This option is used also to log errors that have been recovered but that we want to record in the log system, for example to perform statistical analysis on the occurrence of certain recoverable anomalous conditions.

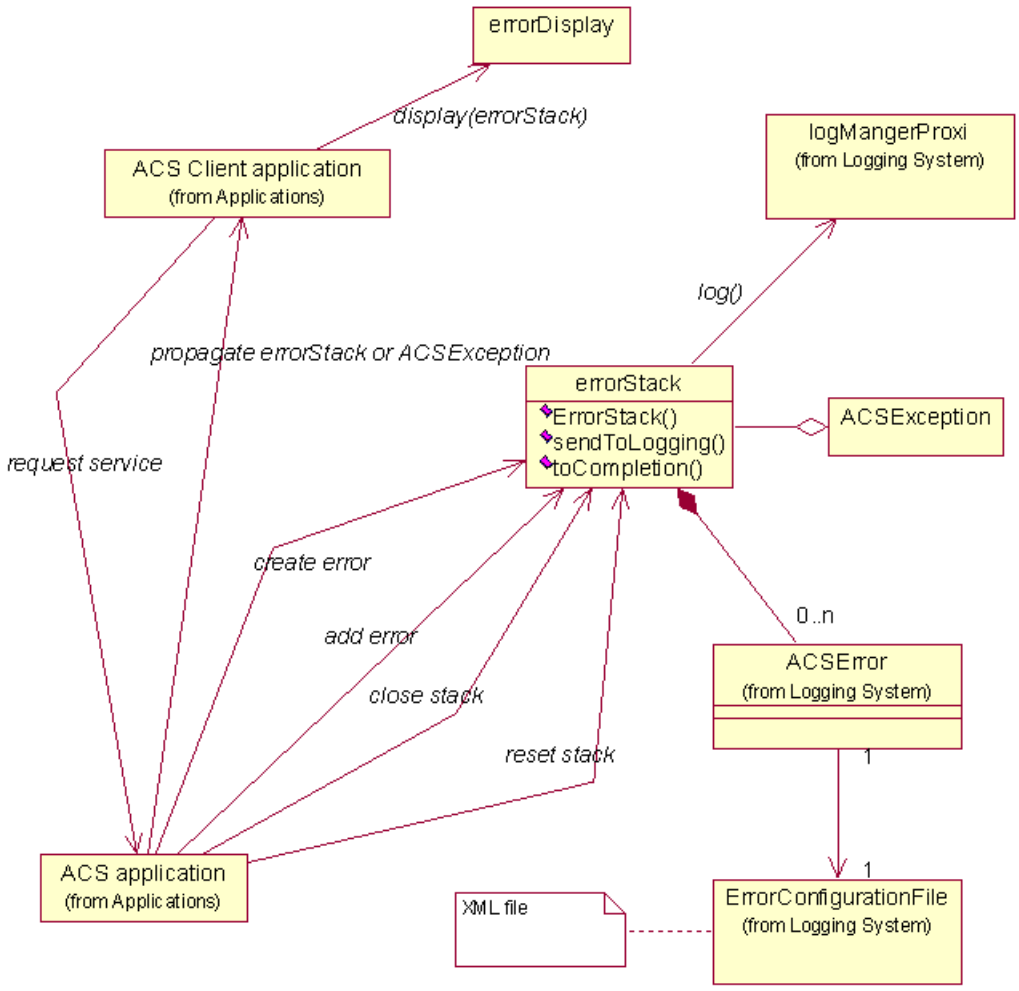

Figure 3.5: Error System

3.7.5 An errorStack class has to be implemented for handling the addition of error information while it is passed from one application layer to the higher one or from a server to its client. The errorStack class contains zero or more ACSErrors. The errorStack class will also have a method to check the stack for a specific error

3.7.6 Errors are propagated via exceptions containing an errorStack. Exceptions would probably be CORBA exceptions, which are propagated through the network in a transparent way.[RD01 - 6.3.1 Definition]

3.7.7 A specific base class for exceptions must be developed. It has to provide support for error stacks, help handling and definition of exceptions.

3.7.8 User interface tools allow navigating through the error logs and through the levels of the error stack [RD01 - 6.3.4 Presentation]. The logging display GUI is used to browse errors and a specific Error display tools is used to browse the error stack corresponding to a single error.

3.7.9 Error conditions are predefined in error configuration files that provide help description of the error and of the recovery procedures to be taken[RD01 - 6.3.5 Configuration]. Error

configuration files are based on XML. Help handling is implemented as links to XML help pages.

3.7.10 Errors have a severity attribute[RD01 - 6.3.3 Severity]

3.7.11 It is important to take into account that exceptions can skip several layers in the call tree if there is no adequate catch clause. In this case the error stack will not have a complete trace of the call stack.

3.8 Logging System

Logging is a general mechanism used to store any kind of status and diagnostic information in an archive, so that it is possible to retrieve and analyze it at a later time.

3.8.1 ACS Logging System is based on CORBA Telecom Log Service[RD14] and on the ACS Data Channel architecture

3.8.2 Applications can log information at run time according to specific formats in order to record[RD01 - 6.2.1 Logging]:

- The execution of actions
- The status of the system
- Anomalous conditions

3.8.3 Logging includes for example:

- Device commands - reception and execution of commands from devices [RD01 - 14.1.1 Logging of commands]
- Debugging - Optional debugging messages, like notification of entering/leaving specific code sections.
- Unrecoverable programmatic errors
- Alarms - change of status in alarm points
- Miscellaneous log messages. Applications can log events regarded as important to archive, for example receivers changing frequency, antennas set to new targets etc.

3.8.4 Each log consists of an XML string with timestamp[RD01 - 6.2.1 Logging], information on the object sending the log and its location in the system, a formatted log message.

3.8.5 The logging system is centralized so that eventually the logs are archived in a central log.

3.8.6 Log clients can subscribe to the Log Data Channel. The permanent Log Archive [RD01 - 6.2.2 Persistency] (an RDBMS) is essentially such a client.

3.8.7 Logs can be cached locally on the machine where they are generated and transmitted to the central log on demand or when the local buffer reaches a predetermined size. High priority logs are not cached but are transmitted to the central log immediately. The main purpose of caching is to reduce network traffic.

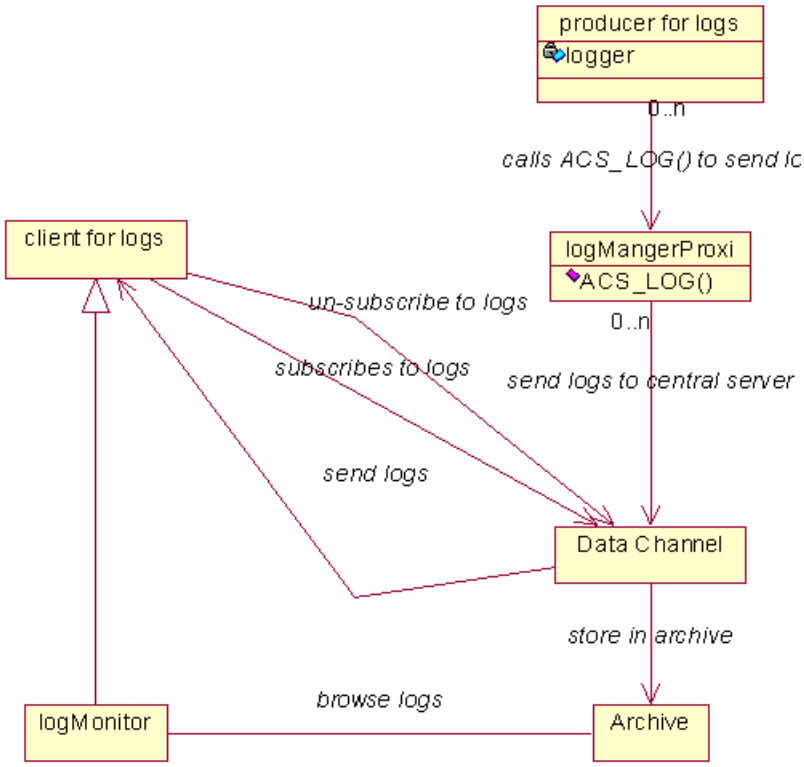

Figure 3.6: Logging System

3.8.8 Applications log data using the API provided by ACS. This API provides methods for logging information of the different kinds specified above.

3.8.9 The log API allows for filtering, so that log entries with low priority do not get logged. The filter can for example be set to log or not log debug messages. The filter level can be set through a command.

3.8.10 The API for error logging must be done such as to prevent message flooding. It shall handle repeated error messages - possibly a mechanism to not report the same error more than once, unless reset.

3.8.11 A user interface (logMonitor) allows to monitor the logs while they are produced, in quasi-real-time, and to browse the historical database offline. It provides filtering and sorting capabilities [RD01 - 6.2.3 Filtering]. It is useful to filter logs by inclusion and exclusion and logical combinations based on:

- time
- process that generated the log

- name of item logged
- all things from a device node down to the leaves
- type of item logged

3.8.12 The logMonitor will be implemented as a Java application and it will use standard beans based on JDBC (Java Database Connectivity) for accessing the long term relational database and standard XML parsers to access the short term logs.

3.8.13 If the Logging System crashes or cannot keep up with log requests, subsystem performance should not be affected but the log system should record that logs have been lost. The log system will run with lower priority than the control system.

3.8.14 The logging system is NOT meant to allow "re-run" observing sequences from log messages, but to allow analyzing a posteriori the exact behavior of the system. Since the system will never be the same twice it will never be possible to execute twice exactly the same set of actions. The task of making observing sequences reproducible must be assigned to the higher level observation scheduler and the logging system has to be used to understand what eventually did not go as planned in the observing sequence and to fix the identified problems.

3.8.15 Debugging logs are always present in code. Multi-level output of debugging logs can be switched on/off by setting the priority for log filtering. No compile time debugging flags are necessary, except when strict real time concerns apply.

3.9 Time System

3.9.1 ALMA will have a distributed HW pulse, interval is 48 ms, that the hardware will synchronize to[RD01 - 8.1.3. Distributed timing] [RD30]. Also devices on the CAN bus will be connected to the time bus to allow synchronization. ACS will provide the possibility to send time-tagged commands to allow synchronous actions.

3.9.2 All time data in ACS (APIs, timestamps….) will be based on the array time [RD01 - 8.1.1. Standard]

3.9.3 The Time System provides basic service for time handling. It includes:

3.9.3.1 Abstract API to access time information in ISO format [RD01 - 8.1.2. API] and to perform time conversion, arithmetic and formatting [RD01 - 5.3.1. Time conversion] to/from various time systems (UTC, TAI…).

3.9.3.2 An abstract API for synchronization between applications based on array time and on the 48 ms HW pulse.

3.9.3.3 Timer classes to trigger events based on time [RD01 - 8.1.4 Services]:

3.9.3.3.1 Applications can create instances of the timer class and register callbacks to be executed when the timer event is triggered

3.9.3.3.2 Timers can be *periodic* (retriggerable) or *oneshot* events

3.9.3.3.3 Timers can be configured to trigger events:

- at a specific array time. For *periodic* timers the time of the first event is specified together with the frequency for the following events).
- after a given number of 48ms pulses (plus an optional offset to allow phasing). For *periodic* timers the number of pulses between to events defines the event repetition rate.

3.9.4 ISO format only will be used "internally". At User Interface level, more formats might be required. We will start providing only ISO format and allow users to request more formats via the Change Request SPR mechanism.

3.9.5 The API is the same for all platforms, but provides higher time resolution and performances on the platforms that have specific HW support (48 ms event distributed by hardware..) [RD01 - 8.1.5 Resolution]

3.10 Astronomical Libraries

The common software provides and integrates also class libraries for astronomical calculation routines (SLA, TPOINT, AIPS++)[RD01 - 5.3.3 Astrometry].

They are off-the-shelf packages only integrated and distributed within ACS.

3.11 Management and Access Control

3.11.1 The Management and Access Control package implements design patterns and high level services to manage the life cycle of distributed objects[RD01 - 5.1.2. Procedures]:

- **Instantiation and de-instantiation of Distributed Objects.**
  Instances of Distributed Objects are created when needed or when requested and destroyed when not needed any more.
- **System startup and shutdown.**
  At system startup all needed Distributed Objects are created and initialized in the proper order.
- **Location of Distributed Objects in the system**.
  A client does not need to know where a Distributed Object resides and objects can be relocated when necessary, for example to recover from failure, for testing purposes or for load balancing.
- **Access granting.**
  A client can gain access to a desired Distributed Object, provided that its access rights are adequate
- **Version control.**
  If a newer version of a Distributed Object is available, or if its configuration data changes, it must be possible to upgrade or reconfigure

the service without interfering with its clients.
Different versions of the same Distributed Object can be loaded or relocated for testing purposes without requiring changes in clients.

- **Administration control.**
  Administrative users must have an overview and control over the current state of the system, including existing services and clients, and must have the possibility of manually manage services.

3.11.2 Distributed Objects will be registered in the CORBA Naming Service. In the first implementation Properties will not be registered in the naming service, but will be accessed retrieving their CORBA reference from the Distributed Object that contains them. This scheme will be reviewed after the TICS development.

3.11.3 Access to the CORBA Naming Service and Distributed Object's life cycle is handled by a Management and Access Control Interface. A Manager is the communication entry point for clients: it provides access to the CORBA Naming Service (with added security) and delegates to Activators the task to manage the life cycle (code loading, creation and destruction) of DOs, based on the request of services from the clients.

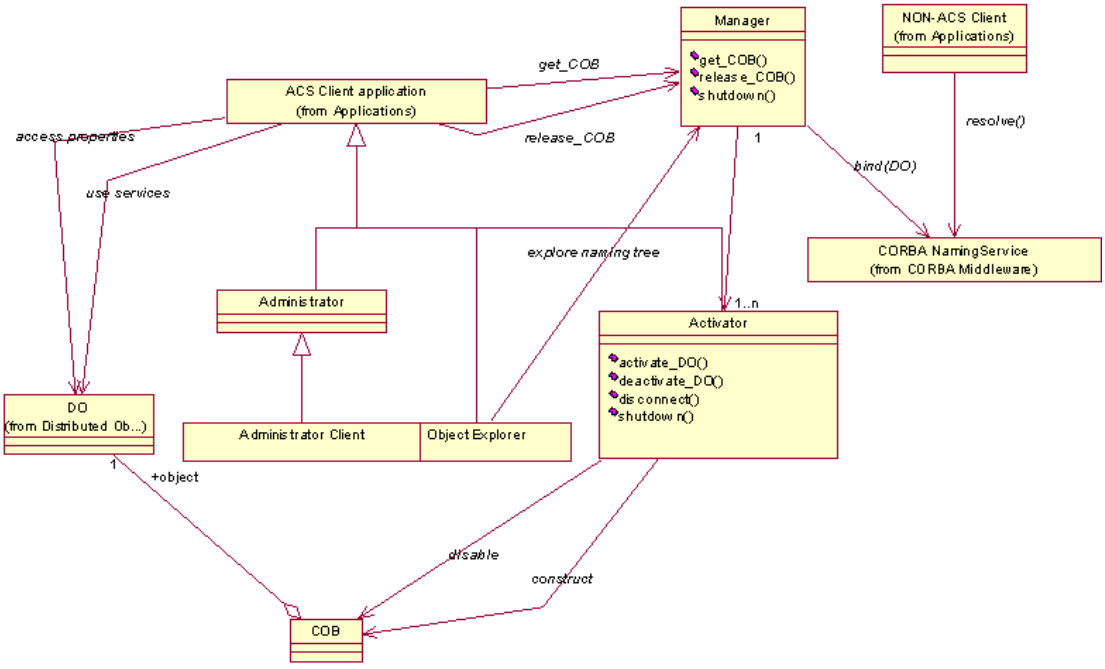

Figure 3.7: Management and Access Control architecture

3.11.4 The basic entities managed by the Management and Access Control interface (MACI) are `CORBA Objects (COBs).`

3.11.4.1 To the management system, the COB is an entity with a lifecycle that has to be managed. COBs can be instantiated in three different ways:

- **regular COBs** are instantiated on demand when they are first requested, and released when they are no longer needed.

- **startup COBs** are instantiated when the system is brought online. They cannot be shut down, unless the entire system is shut down.

- **immortal COBs** are instantiated when they are needed for the first time, but cannot be shut down afterwards. They are similar to startup COBS, but being activated only when needed they allow shortening the time required starting the system.

3.11.4.2 A COB implements the functionality that allows the interaction between the MACI infrastructure and an underlying object. The underlying object can be a Distributed Object or other type of objects, for example CORBA services, which have to be managed by MACI.

3.11.4.3 Every COB has a unique designation. Well-formed COB designations are COB URLs or *CURL*s. A `CURL` is a hierarchical name implemented as a string of arbitrary length that consists of static prefix `curl:`, domain identifier and the COB name. An example of a `CURL` might be `curl://alma/antenna1/mount`, representing the mount COB for ALMA antenna number 1. MACI provides name resolution that from a well-formed COB designation generates the COB reference that can be returned to the client

3.11.5 `Manager` is the central point of interaction between the COBs and the clients requesting their services.

3.11.5.1 Manager has the following functionality:

- It is the communication entry point. A client requesting a COB service can do so by querying the Manager. Security is tied to this functionality by requiring every client to pass the authorization protocol and subsequently tag every request to the manager with a security token generated by the manager upon successful authentication. Manager can serve as a broker for objects that were activated outside of MACI (non-COBs). It provides a mechanism for binding and resolving references to such objects.

- It performs as a name service, resolving CURLs into object references. If a CURL is passed that is outside the current Manager's domain, the Manager forwards the request to the Manager closest to the destination domain to resolve.

- It delegates the COB life cycle management to the Activator object and therefore creates no COBs directly. However, it does maintain a list of all available Activators.

- The Manager uses the configuration database to retrieve relevant configuration for individual COBs in the domain, as well as locations of other Managers.

3.11.5.2 Manager is the only interaction that clients have with MACI. Thus, neither COB implementers nor GUI client developers need concern themselves with aspects of MACI other than the Manager.

3.11.5.3 Manager implementation is based on CORBA Naming Service and all references to COBs are available to clients not aware of MACI functionality through the CORBA Naming Service. A CURL-to-Naming Context mapping allows a one to one correspondence between COBs retrieved using MACI services or from the Naming Service.

3.11.6 Every client of a COB service that is not itself a COB may implement an interface called `Client`.

- The Client interface allows the client to act as a secure party in the communication with the COBs, to receive general-purpose string messages from the MACI components and to be notified when any change happens to the COBs that the client utilizes.
- Each Client logs in to the MACI system before any other requests are made, and in turn it obtains a security token, which it must use in every subsequent request to the MACI.
- The log in and other requests are issued to the Manager, which serves as a portal to other services.

3.11.7 An `Activator` serves as an agent of MACI that is installed on every computer in the control system.

- Every Activator runs in its own process.
- `Manager` sends the `Activator` the request to construct a specific COB by passing it the name, type and the path of executable code of the COB.
- The `Activator` loads the executable code and begins executing it. If the dependant executables are not loaded automatically by the operating system (as is the case on the VxWorks platform), Activator loads them prior to executing any code.
- The `Activator` also deactivates COBs, when so instructed by the `Manager` and is able to shutdown by disabling all COBs.
- `Activator` maintains a list of all COBs it has activated and is able to return information about individual COB's implementation (like file path of the loaded code, version and build date).
- Activator provides the COBs it hosts with an interface through which COBs can perform their MACI-related tasks, such as issuing requests to the Manager and activating other CORBA objects.

3.11.8 `Administrator` is a special-purpose client that can monitor the functioning of the domain that it administers. Monitoring includes obtaining the status of the MACI components as well as notification about the availability of the COB components.

3.11.9 MACI allows organizing Distributed Objects hierarchically and handling startup and shutdown dependencies between objects.

- Whenever a client needs a CORBA reference for a DO, a request to Manager is done for the corresponding COB.
- If the object is not already instantiated, the Manager asks the Activator to create it.
- When an object contains hierarchical references to contained objects, the dependency is expressed via CORBA references and resolved through requests to the Manager. In this way, the Manager can automatically achieve instantiation of not already active nested objects. This guaranties that all objects are automatically created in the right order and when needed.
- Some objects are needed immediately at bootstrap. They are directly specified in a Manager table and the Manager instantiates them as soon as it is bootstrapped.
- If there is a root top-level object, just putting this object in the Manager table will trigger a cascade instantiation of all dependent objects.

3.11.10 An Object Explorer User Interface tool is provided to navigate the hierarchy of distributed objects based on the naming hierarchy. All objects in the system can be reached by navigating the hierarchy and all object information can be retrieved and edited, including accessibility for a given user [RD01 - 5.1.3 Browser]. For example, it will be possible to graphically browse the hierarchy of distributed objects in the system, based on the naming hierarchy, reach every single distributed object and view/edit all values of properties and characteristics. The Object Explorer uses the CORBA Interface Repository to retrieve information on the interfaces provided by the Distributed Objects in the system.

3.11.11 An Administrator Client User Interface tool is provided. The Administrator Client:

- displays the information about CORBA objects on the local network. This includes COBs, Managers and Activators. Both currently active COBs and potentially active COBs (i.e. COBs that the Manager is able to bring online on request) are displayed.
- interacts with the Manager through IDL Administrator interface, by receiving notifications about other clients and activators in the system from the Manager.

3.12 Archiving System

3.12.1 The archiving system provides archiving of monitor point values. ACS provides only the data collection mechanism, but not storing of data. Data are made available from the archiving data channel.

3.12.2 The value of each property in the control system can be archived. Archiving is enabled/disabled and configured on per-property basis.

3.12.3 ACS Properties publish their value on the archiving data channel. The parameters for data publishing are defined in the following Property's Characteristics:

`ArchivePriority`: The priority of the log entry that will carry the information required for archiving the parameter's value. If the priority exceeds the value specified in the `MaxCachePriority`, the archiving data will be transmitted immediately. If it is below `MinCachePriority`, the data will be ignored. If it is somewhere in-between, it will be cached locally until a sufficient amount of log entries is collected for transmission.

`ArchiveMaxInterval`: The maximum amount of time allowed passing between two consecutive submissions to the channel. If the time exceeds the value specified here, the entry should be submitted even though the value of the parameter has not changed sufficiently.

`ArchiveMinInterval:` The minimum amount of time allowed passing between two consecutive submissions to the log. If the time is smaller than the value specified here, the value is not submitted, even though the value of the parameter has changed.

`ArchiveDelta` (same type as parameter): Defines what a change in parameter value is. If the value changes for less than the amount specified here, no value is submitted.

3.12.4 Clients can subscribe to the Archiving Data Channel using filters to get events when archiving data is available. ACS provides an API that allows parsing the Notification Channel messages containing monitor point archiving data.

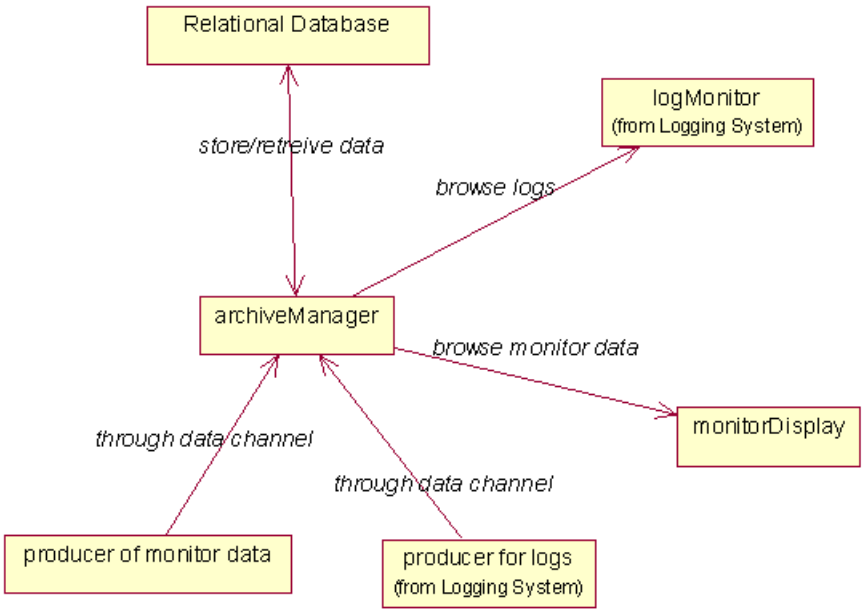

Figure 3.8: Archiving System architecture

3.12.5 Periodic data collection of properties can be synchronized to the 48 ms hardware tick

3.12.6 Individual properties data are cached locally before they are sent to the central archive

3.12.7 Each property can be uniquely identified by a name, composed of the device followed by the property name.

3.12.8 The archiving system supplies a mechanism for retrieving the historical value of any property, provided that the value has been archived.

3.13 Command System

A command is the basic mechanism for communication from users to Distributed Objects and between Distributed Objects. A command is actually a DO method.

3.13.1 Commands are sent to Distributed Objects [RD01 - 6.1.2. Commands] using remote method invocation. It is based on CORBA[RD01 - 10.4.1. CORBA][RD01 - 13.1.1. Distributed Objects and Commands].

3.13.1.1 CORBA provides full support for inter-process communication.

3.13.1.2 CORBA objects have a public interface defined with the IDL language[RD01 - 10.3.4. IDL]

3.13.1.3 CORBA objects can be remotely accessed by creating stubs and invoking the defined IDL interface.

3.13.1.4 Any language supported by CORBA can talk to any remote object, independently from implementation language and architecture. The Object Request Broker (ORB) does mapping of calls and marshalling.

3.13.1.5 CORBA defines a standard Internet Inter-ORB Protocol (IIOP) that guarantees interoperability between any CORBA implementation and vendor based on TCP/IP. Any implementation must comply with IIOP, but a vendor can choose to additionally implement high performance transport protocols. For example there are native ATM implementations. Same-process messages are usually implemented on direct function calls while same-CPU messages are based on operating system message queues.

3.13.1.6 A call to a method of a CORBA Distributed Object, based on its IDL interface, is what can and has to be mapped into the concept of Commands (the method call concept is very similar to RPC).

3.13.2 A command has a well-defined syntax and set of call and return parameters. Full context validation and syntax check of commands is always done when the command is received by the server application[RD01 - 6.1.4. Validation]. A command can also be checked by the sender, but this is not mandatory except in the case of generic command-sending GUIs[RD01 - 6.1.3. Syntax Check]. The syntax check would check that the command is valid and that the parameters are within the static ranges.

3.13.3 Commands are synchronous (the caller blocks and waits for a return reply, up to a defined timeout) [RD01 - 6.1.7. Mode] [RD01 - 6.1.8. Timeout]. Applications should take care of the fact that a synchronous call can block the whole application unless it is issued from a dedicated thread. Replies can be normal replies or error replies[RD01 - 6.1.1. Messages].

3.13.4 CORBA Asynchronous Method Invocation (AMI) [RD29] can be used to implement asynchronous calls on the client side using synchronous methods on the servant side. AMI is only supported by few ORBs, for example TAO, but not by Orbacus.

3.13.5 Asynchronous command handling using synchronous methods on the servant side can also be done by starting a separate thread, which sends a synchronous command. This way the main thread is not blocked.

3.13.6 Using synchronous commands, time-outs and intermediate replies are not handled by ACS, but must be taken care of by the application. ACS cannot the warranty that requirements [RD01 - 6.1.6 Command delivery] and [RD01 - 6.1.9 Intermediate replies] are satisfied. This is let to applications

3.13.7 Commands can be invoked from generic applications, that are able to browse the objects in the systems, show the available commands with the corresponding syntax, check dynamically the syntax of commands and send them[RD01 - 6.1.5. Programmatic use].

3.13.8 A server sub-system handling a command continues to operate if the client that has issued the command disappears, e.g. between a command being initiated and completed. In this case the server logs a non-critical error, since a client should always wait for replies to initiated actions, and continues.

3.13.9 Objects publish their interfaces via IDL interfaces. IDL allows use of inheritance to build new types. IDL allows only defining the pure interface in terms of parameters and types; it does not allow specifying range checking for the parameters. This checking has to be performed by the applications. IDL interfaces are registered in the CORBA Interface Repository (IREP) and made public.

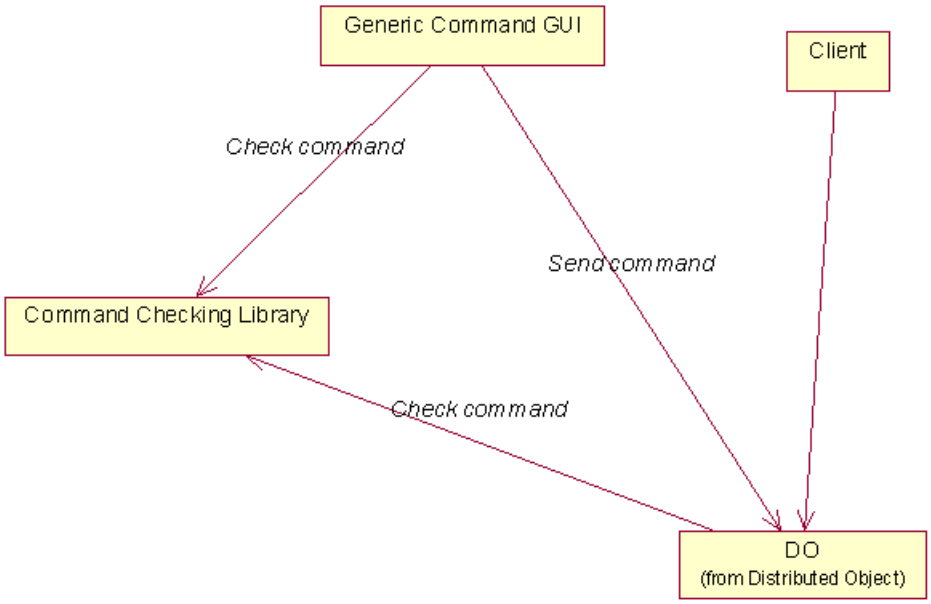

Figure 3.9: Command System architecture

3.13.10 ACS will provide checking functions and tools to implement command syntax checking both at the sender and receiving side of commands. Checking of commands on the sender side is for example required for generic tools like a command line tool to send commands from the shell.

3.13.11 Property classes implement a Check() method, that can be invoked to verify compliance of a value with what is specified in its Characteristics. Characteristics are used to perform range and other run-time checks.

3.13.12 A Distributed Object receiving a Command performs range checking and validation of the whole command syntax by calling transparently a Command Parameter Checking library, which has to be designed.

3.13.13 CORBA introspection is used to develop generic tools able to analyze at run time the IDL interfaces of available CORBA objects to provide a list of available "commands" with their signatures and to execute on the fly commands. Alternatively, XML definitions of the commands in the Configuration Database can be used, see below.

3.13.14 Commands and their Parameters are defined in the Configuration Database[RD01 - 6.1.5. Programmatic use] [RD01 - 4.2.1. Configuration Database]. The Configuration Database of a Distributed Object contains, together with Property definitions, one entry per command. The description of a command parameter has the same syntax as a Property of the same basic type; i.e. is defined by the same set of Characteristics. In an XML implementation of the Configuration Database, a COMMAND tag is used to describe commands and PROPERTY sub-tags to define each parameter in a hierarchical structure. Using the same syntax to describe Properties and Command Parameters allows a seamless and coherent mapping between the two types of entities. A good solution to guarantee alignment between Configuration Database and IDL definitions would be to generate IDL from XML definitions, but this has to be investigated.

3.13.15 Clients can also call the Command Parameter Checking library prior of invoking a command, but are not bound to do it. This will be typically done by interactive applications to validate human user input. The usage of the Command Description in the Configuration Database also allows to implement client-side context sensitive help and command completion.

3.13.16 In case of crash of the remote application, the client ORB notifies the caller with proper CORBA exceptions.

3.14 Alarm System

The Alarm System provides a mechanism for notifying asynchronously operators and registered applications of the occurrence of important anomalous conditions that can disrupt the observation or be potentially dangerous for the system[RD01 - 6.4.1 Definition].

What is described below is still mostly TBD and needs prototyping to try out the concepts.

3.14.1 Properties are able to trigger alarms. The simple alarm is triggered when the value of the property exceeds or falls below the alarm limits defined for the property. Hysteresis is provided for the alarm limits.

3.14.2 Clients can be notified on alarms through callbacks. Notification is done when the alarm is set and when it is cleared.

3.14.3 Whenever an alarm is set or cleared, the event is logged in the Logging System [RD01 - 6.4.6. Alarm logging].

3.14.4 More complex alarms will be based on objects. They will have the following properties

- An enumerated state property with the values: GOOD, WARNING, BAD
- An enumerated occurrence property (FIRST OCCURRENCE, MULTIPLE OCCURRENCE....)[RD01 - 6.4.4. State]
- A timestamp property describing when the alarm status changed
- A property holding the status of user acknowledge, when required
- A set of description characteristics [RD01 - 6.4.7. Configuration], including (on top of the standard ones like description, URL for extended documentation...) a severity [RD01 - 6.4.3. Severity]
- Some alarms just disappear when the alarm condition is cleared, but others require an explicit acknowledgement from the operator[RD01 - 6.4.2 Behavior]

3.14.5 Hierarchical alarms can be constructed by a client on top of the available alarm system. Such an implementation would be based on the OVRO hierarchical alarm system [RD22][RD01 - 6.4.5. Hierarchy].

3.14.6 It is possible to define actions that have to be started on the occurrence of a given alarm condition [RD01 - 6.4.8. Binding]. An API is provided for applications, so that they can request a notification to be sent via a callback of the occurrence of specific alarms.

3.14.7 An alarmDisplay GUI provides an operator user interface to the alarms. Using this application the operator can browse through the currently active alarms, request for detailed information and help about a specific alarm, acknowledge alarms when this is required.

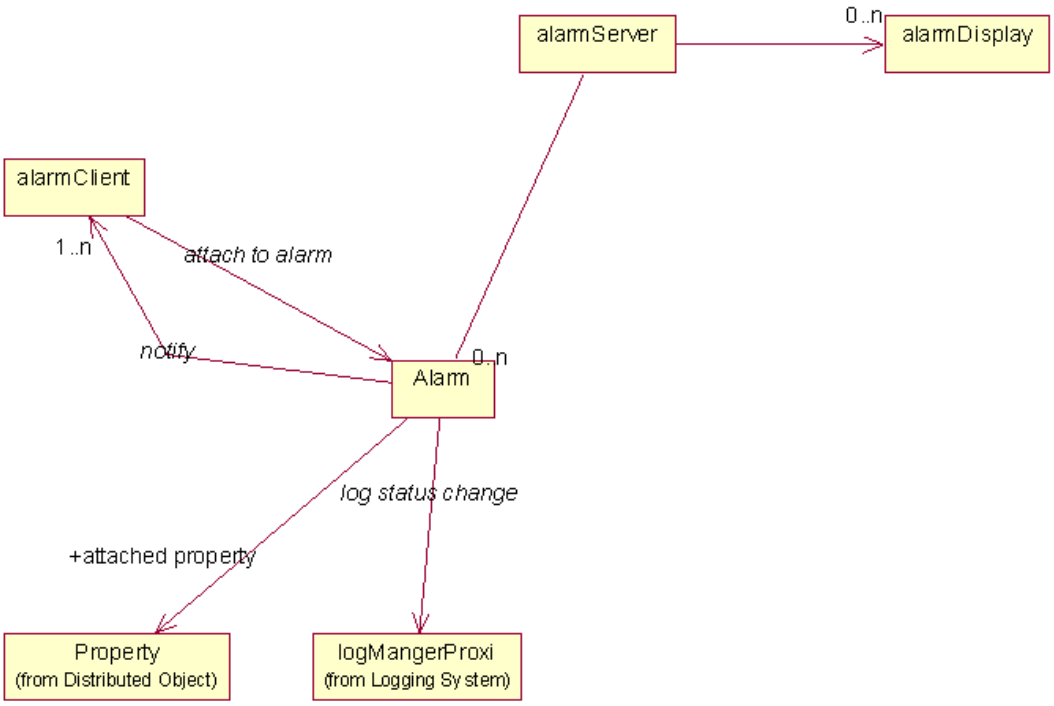

Figure 3.10: Alarm System architecture

3.15 Sampling

3.15.1 Every Property can be sampled at a specified sustained frequency, up to 200 Hz, limited by hardware.

3.15.2 A sampling frequency of 1KHz for a limited number of Distributed Object Properties must be allowed in order to implement a "virtual oscilloscope" whereby, for example, noise sources in the IF chains can be correlated and identified.

3.15.3 The data channel transports sampling data.

3.15.4 The samples are stored in a data file for later analysis or to be passed to applications like LabView for analysis of engineering data. Every sample is time-stamped to allow correlating data from different sources. [RD01 - 4.3.1. Sampling]

3.15.5 The samples can be plotted in near real-time on a user interface on the operator terminal or any other user station. The plotting application subscribes to the sampling data from the data channel, which is fed by the sampling manager, and periodically updates the running plot with the new data [RD01 - 5.1.4 Analysis tool]. The plotting application can also display sampling data from data files. The plotting application can be:

- A Java application that uses Java plotting widgets and in particular widgets aware of the Property internal structure to define plotting scales and units.

- A COTS application, like LabView[RD13] [RD01 - 10.5.7 Analysis and plotting], with advanced plotting and analysis capabilities.

3.15.6 Multiple samples can be super-imposed on the same plot with the same time base, even if the data are received from different distributed objects, possibly on different antennae. Also, it should be possible to plot samples against one another, an example being VI curves (voltage vs. current) used in biasing mixer junctions.

3.15.7 Data transport is optimized. The samples are NOT sent one by one to the central control computer responsible for storing data, but are cached on the local computer and sent in packets (for example with 1 Hz. Frequency). This is of primary importance to keep network load under control. [RD01 - 4.3.1. Sampling]

3.15.8 There will be a sampling engine implemented as a distributed object:

3.15.8.1 It can be activated on any local computer

3.15.8.2 Can be configured to sample one or more Properties at given frequencies.

3.15.8.3 It sends periodically at low frequency packets of data to a sampling server that stores the data for later analysis or provides them to near-real-time plotting applications.

3.15.8.4 Sampling must not reduce the performance of the control system or the network. The caching of data will reduce network traffic while assuring that the sampling processes runs with lower priority than control processes will reduce the impact on the performance of the control system.

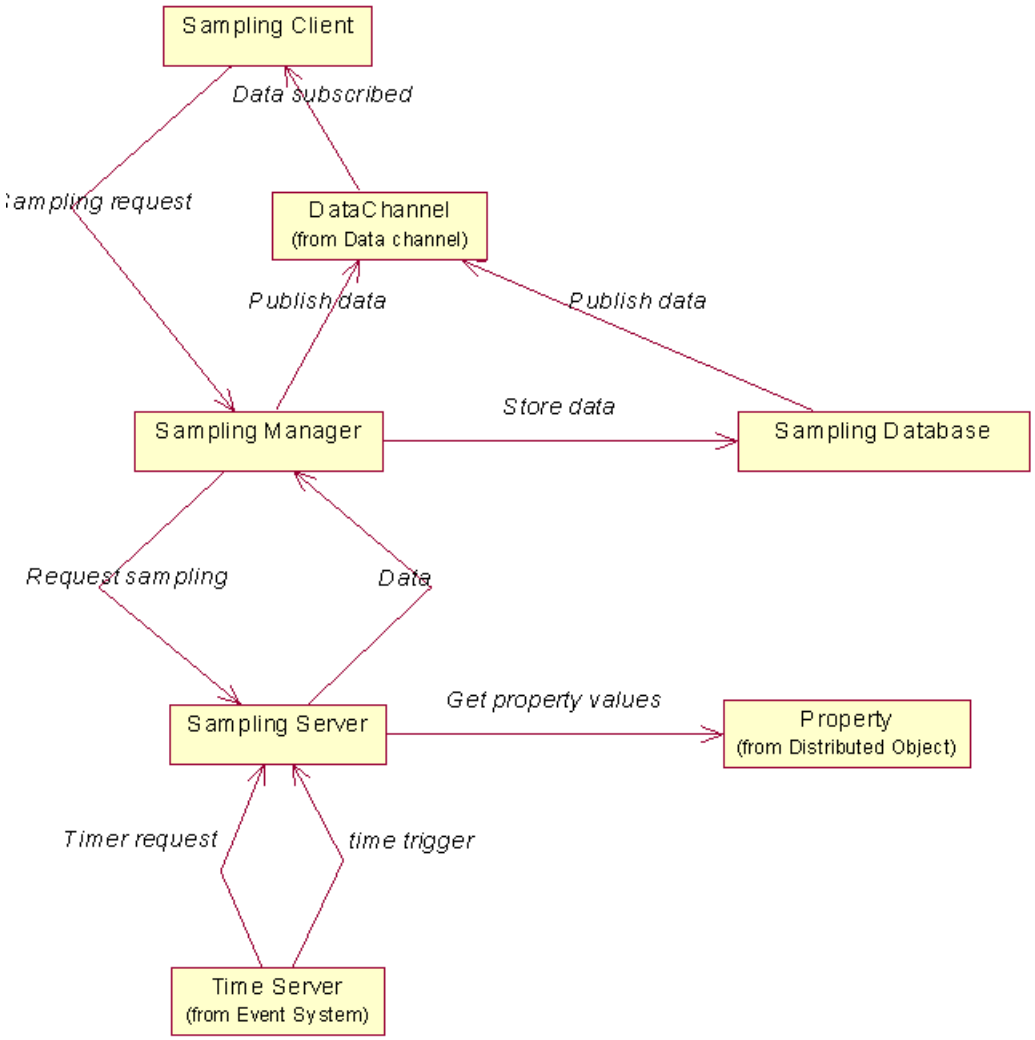

Figure 3.11: Sampling architecture

3.15.9 XML seems an interesting data format for sampling files. This format allows also easy data exchange with relational databases. Style sheets based on the eXtensible Style Language (XSL) can be used to transform XML files in other formats, like simple ACSII blank delimited tabular files.

3.16 User Interface Libraries and Tools

3.16.1 The Java language[RD06] is the current choice for GUI development, in particular coupled with the SWING graphical library

3.16.1.1 A rich set of widgets is part of the standard package.

3.16.1.2 Java is highly platform independent [RD01 - 12.1.1 Portability]. GUIs are platform independent and can be developed on any system and run on any other system[RD01 - 12.1.2 Extended Portability].

3.16.1.3 Java applications can run inside Web Browsers for the deployment of low complexity GUIs on remote sites [RD01 - 12.1.2 Extended Portability].

3.16.1.4 Both Java and Web Browsers have good CORBA support. Orbacus[RD19] is the ORB currently selected to provide CORBA support in Java.

3.16.2 ACS provides a standard set of tools for the development of user interfaces[RD01 - 12.1.3 Tools]. This includes:

3.16.2.1 Interactive GUI builder. A COTS GUI builder has to be selected. The current choice is IBM Visual Age[RD16]: it is available on Windows and Linux platforms and is used by ANKA[RD04] since it provides a very good visual integration of Java Beans. There is no point in developing a proprietary tool.

3.16.2.2 Standard set of widgets for the implementation of applications with common look and feel. The adoption of the Distributed Object concept allows reuse of many Java Beans developed in the particle accelerator community specifically for the implementation of control systems. In particular the Java Beans developed for ANKA[RD04] constitute the core palette for ACS.

3.16.2.3 Standard Java class libraries for the integration with the control systems and for the usage of all services provided by the ACS

3.16.3 All GUIs for generic ACS applications, (Object Explorer, logMonitor…) will be developed using ACS standard GUI libraries [RD01 - 12.1.5 ACS GUIs].

3.16.4 GUIs do not implement any control logic[RD01 - 12.1.4 Modularity]. They are dummy applications that simply send commands to Distributed Objects and access Distributed Object Properties for display and update. Any action performed through a GUI can be also performed using command line commands, script or simple programs and no GUI is strictly required to drive the system.

3.16.5 Integration with the Web through the usage of an HTML/XML Browser provides access to online documentation.

3.16.6 The Web Browser is also used to deploy low complexity GUIs on remote sites without requiring installation of specific and heavy software packages. (Control GUIs and in any case complex GUIs are not supposed to run inside a Web Browser but as stand-alone applications).

3.17 Scripting support

3.17.1 A scripting language is part of ACS [RD01 - 5.2.1 Scripts]

3.17.2 It is used as a glue language between the high level interfaces and the control and data systems[RD01 - 5.2.1 Scripts]:

3.17.2.1 High level procedures of a coordination nature

3.17.2.2 System administration, replacing shell script languages for new developments

3.17.2.3 Rapid prototyping

3.17.2.4 Test applications and procedures

3.17.3 It provides access to the basic features of all Common Software services through[RD01 - 5.2.2.Functionality]:

3.17.3.1 CORBA invocation of remote objects

3.17.3.2 A package of support classes implemented in the scripting language itself

3.17.4 Python[RD24] and Tcl/Tk[RD25] (with combat as CORBA interface) are the chosen candidates [RD01 - 10.5.6.Scripting language]. They seem to satisfy all the requirements listed below

3.17.5 Needed Features:

3.17.5.1 Object Oriented

3.17.5.2 Platform independent.

3.17.5.3 Flow control, variables, and procedures.

3.17.5.4 Plug and play to install scripting language.

3.17.5.5 Connects to CORBA.

3.17.5.6 Easy to use for non-programmers.

3.17.5.7 Rapid Prototyping

3.17.6 Features that would be nice to have are:

3.17.6.1 Similar to programming languages used in the project.

3.17.6.2 GUI development.

3.17.6.3 Embeddable into code for interactive applications.

3.18 ACS Application Framework

3.18.1 The common software has to provide also an application framework that allows an easy implementation of a standard application (a skeleton application) with all essential

features[RD01 - 5.1.1 Framework]. Inheritance and other mechanisms are used to enhance and extend the basic application according to specific application needs.

3.18.2 The application framework enforces common coding solutions and adherence to predefined standards. For example, each physical sub-system is controlled by a "server" process that implements:

3.18.2.1 A minimum number of standard commands with a standard behavior

3.18.2.2 HW level device control

3.18.2.3 Extensions for:

- Specific commands
- Sub-states

3.18.2.4 A standard state machine for the distributed object will be implemented within TICS. This can later be integrated into the application framework.

3.18.3 The application framework will be implemented as a set of classes that provide:

- Skeleton for standard application, with standard commands and states
- Implementation of typical communication Design Patterns
- Implementation of standard startup/shutdown/configuration procedures

3.18.4 This implementation follows defined protocols for command handling, to guarantee for example command concurrence and response time. A way to make sure that these requirements for a server are satisfied is to provide an extensible framework that implements all these design patterns.

3.18.5 At the higher level the same concepts can be applied to coordination applications and most of these same patterns go well over the boundaries of the control system.

3.19 FITS Libraries

The common software provides and integrates also class libraries for the handling of standard file formats (FITS files) [RD01 - 7.2.1 FITS], mathematical libraries (Fats, Fitting) [RD01 - 5.3.2 Mathematics], units conversion and handling (based on the AIP++ unit management classes) and other similar high-level support libraries. It is important to keep in mind that the common software will have to go over the boundary of the Control System in the direction of the data flow and handling.

The FITS Libraries package is just an example of such components.

A list of libraries has to be compiled and standard off-the-shelf libraries should be used whenever possible, eventually with a small wrap around to make them uniform with the rest of the system.

Libraries and application frameworks developed by other teams (Control System, Correlator, Data Flow, Archiving…) and recognized as of general use will have to be added to the Common Software kernel. Input is required in these areas.

Packaging all these libraries, even if off-the-shelf, in the ACS distribution guarantees that all sites involved in the project have the same version of the packages and that they are built and installed with the same options.

3.19.1 Astronomical data flow

ACS should provide also support for the transport of bulk data [RD01 - 7.1. Bulk data transfer], to be used for science data. This is not specified in this issue of the document, but performance tests on bulk data transfer will be performed with TICS, with a TBD prototype application. CORBA provides[RD27] design patterns aimed at the transfer of large amounts of data that should satisfy the requirements expressed in [RD01 - 7.1.1 Image pipeline].

4 Attributes

4.1 Security

Security requirements, stated in [RD01 - 9.1 Security] are not analyzed in this document, but there is no indication that they cannot be met with the described architecture. They will be taken into account for the next major revision, after ACS 1.0.

4.2 Safety

Requirement [RD01 - 9.2.1. Safety] does not impose any particular behavior on ACS, since it states that software based on ACS has no direct responsibility on human and machine safety.

Software limits will be directly supported in ACS [RD01 - 9.2.2. Software limits]. Applications will handle software limits as alarms on Properties representing the value to be kept within limit. In a typical case, a value with software and hardware limits will be handled using a 5-state alarm (normal, high/low SW limit, high/low HW limit). Reaching a software limit will trigger a callback in the application responsible to handle the limit condition.

4.3 Reliability

Reliability requirements, stated in [RD01 - 9.3 Reliability] are not explicitly analyzed in this document, but there is no indication that they cannot be met with the described architecture. In particular, CORBA itself has been designed having high reliability and availability as a key requirement. The specific requirements will be taken explicitly into account for the next major revision, after ACS 1.0.

4.4 Performance

The current choices for hardware and software platforms and the architecture for ACS described in the previous sections should ensure that the required performances could be achieved.

Basic performance requirements have been verified with ACS 0.0 on the Kitt Peak 12m antenna.

All performance requirements stated in [RD01 - 9.4. Performance] will be verified with ACS 1.0 on a prototype aimed at implementing a realistic system in term of number and size of objects [RD01 - 3.3.1 Size] and of number of interactions and with the TICS [RD26] interferometer.

A specific feasibility check of CORBA in relation to astronomical data transfer will have to be performed with ACS 1.0. but is not discussed in this issue of the document [RD01 - 10.5.9 Data transfer]. Here we want to test the data channel concept.

Another area where feasibility tests are necessary is the use of XML. One example is retrieval of logs with big logging archives. Also the use of XML for sampling needs to be verified.

5 Life cycle aspects

ACS is being designed and implemented according to what is specified in [RD01 - 11. Life cycle aspects].

ACS uses procedures and standards already defined by the Software Engineering working group. In particular the software development process is defined by [RD17].

ACS 0.0 is a first essential prototype and has been used to validate the initial choice of CORBA and the basic architecture described in this document [RD01 - 11.1.1 Prototyping].

ACS 1.0 will be used for the implementation of the Test Interferometer Control Software (TICS) and the features to be developed for ACS 1.0 are driven by the requirements for the implementation of TICS [RD26].

5.1 Design requirements

Different versions of the system can exist in different places at the same time, with different Configuration Databases, but it must be possible to synchronize them (for example, development is done on a control model, with certain test motors, and installation is done on the telescope, that has different motors) [RD01 - 13.1.5 Configuration].

Differently from what we thought when we have started the investigation on CORBA, we have verified that code does not always need to be recompiled if new IDL interfaces are made available or if an IDL interface is changed, for example adding new methods[RD01 - 13.1.6. Modularity]. With TAO and ORBacus even if one changes interfaces, both client and server still work if they do not use the new features. It is possible to have the server exporting one version of the interface and the client use another version. If an interface changes, then re-compilation is necessary (as expected) only for those programs that use the new feature of the methods that eventually changed signature. This should

work also with other ORBs, since it is due to the existing CORBA IIOP protocol. On the other hand, this behavior is not explicitly supported and, should the IIOP protocol ever change, different interfaces might lead to run-time errors.

ACS will be integrated and tested with any software specified in [RD01 - 12.3.1 The ACS shall be integrated and tested].

5.1.1 Portability and de-coupling from HW/SW platform

ACS shall be designed to be portable across RTOS, Unix and Linux platforms for what concerns its upper level components [RD01 - 13.1.7 Portability].

The higher level of software in particular (co-ordination and user interface) should be as much as possible portable and de-coupled from any specific HW and SW platform. This pushes in the direction of the Java language and of the usage of a portable scripting language like Python[RD24] and/or Tcl/Tk[RD25]. Java virtual machines and Java ORBs are available on any platform and for any web browser. At present time there are still compatibility issues between Java Virtual Machines ("compile once, debug everywhere" syndrome), particularly on a UI with live content. Most interfaces are not "alive" and do not expose the incompatibilities, but many ALMA interfaces will have live content. Web browsers are particularly sensitive to compatibility problems since they usually incorporate outdated Java Virtual Machines. Experience has shown that it is best to limit the number of target platforms and that real-world Java programs should run as independent applications.

Java performance is still a major issue and the language should be used mainly for GUIs and for high-level coordination applications with no strict performance requirements. For non-UI work and in particular for server applications, C++ (while not perfect) is the language of choice: it is fast, efficient, mature, and has ANSI/ISO standards. It is then suggested to use C++ in performance demanding applications and Java everywhere else [RD01 - 10.3.3 Compiled languages], with the exception of low level drivers that can be implemented in plain C language.

Java and C++ interface at the network level through CORBA via IDL interfaces.

Portability among different versions of UNIX and UNIX-like RTOSs is improved by adopting the ACE libraries[RD23] and the Posix standard [RD01 - 10.3.2 High level operating system], although at the RTOS level specific real-time features provided by the operating system (VxWorks for phase 1 [RD01 - 10.5.3 RTOS]) can be used[RD01 - 10.3.1. RTOS].

5.1.2 Small, isolated modules for basic services

Basic services of use outside the Control Software (for example logging and error system) must be independent from the other packages, so that they can be used also for data flow and management without requiring installation of control system components. Also access to control system data must be de-coupled.

CORBA is very well suited for this, since ORBs are available on any platform and also inside web browsers to provide the basic communication layer.

6 Requirements Traceability Matrix

The following table provides a trace of all requirements expressed in [RD01] into this document.

A **Release** column specifies when the described requirement will be implemented. Four releases have been taking into consideration:

- 0.0 is the first prototype release
- 1.0 is the first production release, that should implement all basic functionality
- 1.1 is a bug fixing release (after a 6 month cycle), providing no new functionality
- 1.2 is the following real release (after a 1 year cycle), providing extensions to the core functionality

When only partial functionality if implemented in a release, the release number is put in parenthesis. Some requirements (like 2.3, 3.1.1 and so on) are at the core of the whole ACS and therefore it does not make any sense to specify a release for implementation. In this case the release is marked as Not Applicable (N/A).

Item **in ACS Technical Requirements** [RD01]	**ACS Architecture**	**Release**
2.3. Reference Architecture	1.3	N/A
3.1.1. Scope.	1.2	N/A
3.1.2. Design.	1.2, 2.1	N/A
3.1.3. Use.	1.2	N/A
3.2.1. Users.	1.2	N/A
3.2.2. Value retrieval.	3.6.3	1.0
3.2.3. Value setting.	2.5.11.2	0.0
3.2.4. Local and central operation.	2.2	(0.0) 1.0
3.2.5. Remote access.	2.2	1.0
3.3.1. Size.	4.4	0.0
3.3.2. Serialization.	3.5.3.1, 3.5.5	1.2
3.3.3. Migration.	3.5.6	1.2
3.3.4. Simulation.	3.5.9	1.2
4.1.1. Direct value retrieval.	3.5.11.1	0.0
4.1.2. Indirect value retrieval.	3.6.3	1.0
4.1.3. Rate	3.5.12.2, 3.5.12.3	1.0
4.1.4. Transparency.	3.6.3	1.0
4.1.5. Values at given time.	3.5.12.3	(0.0) 1.0
4.1.6. Data channel.	3.6	1.0
4.1.7. Quality.	Change request to remove requirement submitted (ALMASW2001080)	
4.2.1. Configuration Database.	3.5.3, 3.13.14	(0.0) 1.2
4.2.2. Database design.	3.5.3.2	1.0
4.3.1. Sampling.	3.15.4, 3.15.7	(1.0) 1.2
5.1.1. Framework.	3.18.1	1.2

5.1.2. Procedures.	3.11.1	1.2
5.1.3. Browser.	3.11.10	1.0
5.1.4. Analysis tool.	3.15.5	1.0
5.2.1. Scripts.	3.17.1, 3.17.2	(0.0) 1.0
5.2.2. Functionality.	3.17.3	0.0
5.3.1. Time conversion	3.9.3.1	1.0
5.3.2. Mathematics	3.19	1.2
5.3.3. Astrometry	3.10	1.2
6.1.1. Messages	3.13.3	(0.0) 1.0
6.1.2. Commands.	3.13.1	0.0
6.1.3. Syntax Check.	3.13.2	1.2
6.1.4. Validation.	3.13.2	1.0
6.1.5. Programmatic use.	3.13.7, 3.13.14	(1.0) 1.2
6.1.6. Command delivery.	-	
6.1.7. Mode.	3.13.3	(0.0) 1.0
6.1.8. Timeout.	3.13.3	(0.0) 1.0
6.1.9. Intermediate replies.	-	
6.2.1. Logging.	3.8.2, 3.8.4	(0.0) 1.0
6.2.2. Persistency.	3.8.6	1.2
6.2.3. Filtering.	3.8.11	(0.0) 1.0
6.3.1. Definition.	3.7.6	N/A
6.3.2. Tracing.	3.7.3	1.0
6.3.3. Severity.	3.7.10	1.0
6.3.4. Presentation.	3.7.8	(0.0) 1.0
6.3.5. Configuration.	3.7.9	1.2
6.3.6. Scope.	3.7.2	0.0
6.4.1. Definition.	3.14	N/A
6.4.2. Behavior.	3.14.4	1.2
6.4.3. Severity.	3.14.4	1.2
6.4.4. State.	3.14.4	1.2
6.4.5. Hierarchy.	3.14.5	1.2
6.4.6. Alarm logging.	3.14.3	1.0
6.4.7. Configuration.	3.14.4	1.2
6.4.8. Binding.	3.14.6	1.2
7.1 Bulk data transfer	3.19.1	1.2
7.1.1. Image pipeline	3.19.1	1.2
7.2.1. FITS format shall be supported.	3.19	1.2
8.1.1. Standard.	3.9.2	1.0
8.1.2. API.	3.9.3.1	1.0
8.1.3. Distributed timing.	3.9.1	1.0
8.1.4. Services.	3.9.3.3	1.0
8.1.5. Resolution.	3.9.5	1.0
9.1. Security.	4.1	1.0
9.2.1. All human and machine safety	4.2	0.0
9.2.2. The use of software limits shall be supported	4.2	1.0

9.3 Reliability	4.3	1.0
9.4 Performance	4.4	0.0
10.3.1. RTOS.	5.1.1	0.0
10.3.2. High level operating system.	5.1.1	0.0
10.3.3. Compiled Languages.	2.2, 5.1.1	0.0
10.3.4. IDL.	3.13.1.2	0.0
10.4.1. CORBA.	3.13.1	0.0
10.4.2. ORB independence.	3.5.8	0.0
10.4.3. LAN.	1.3	0.0
10.4.4. Backbone.	1.3	0.0
10.4.5. Field-bus.	1.3	0.0
10.5.3. RTOS	3.4.2, 5.1.1	0.0
10.5.4. OS	3.4.2	0.0
10.5.5. Configuration DB	3.5.3.2	(0.0) 1.0
10.5.6. Scripting language	2.2, 3.17.4	(0.0) 1.0
10.5.7. Analysis and plotting	3.15.5	1.0
10.5.8. CORBA ORB	3.5.8	0.0
10.5.9. Data transfer	4.4	(0.0) 1.0
10.5.10. The XML format	3.5.5	(0.0) 1.0
10.5.11. LAN	1.3	0.0
10.5.12. Field-bus	1.3, 3.4.1	0.0
11. Life cycle aspects	5	0.0
11.1.1. Prototyping	5	0.0
12.1.1. Portability	3.16.1.2	0.0
12.1.2. Extended portability	3.16.1.2, 3.16.1.3	(0.0) 1.0
12.1.3. Tools	3.16.2	0.0
12.1.4. Modularity	3.16.4	0.0
12.1.5. ACS GUIs	3.16.3	(0.0) 1.0
12.1.6. Location	2.2	1.0
12.2.1. Hardware interfaces	3.4.1	(0.0) 1.0
12.3.1. The ACS shall be integrated and tested	5.1	(0.0) 1.0
13.1.1. Distributed Objects and Commands.	2.1, 3.5, 3.5.1.5, 3.13.1	0.0
13.1.2. Standard methods.	3.5.4	1.2
13.1.3. Groups.	Change request to remove requirement submitted (ALMASW2001079).	
13.1.4. Events.	3.5.12.1	(0.0) 1.0
13.1.5. Configuration.	5.1	1.0
13.1.6. Modularity.	5.1	1.0
13.1.7. Portability.	5.1.1	0.0
14.1.1. Logging of commands	3.8.3	1.0
14.1.9. Dynamic configuration	3.5.3.5	1.0
14.1.13. States	3.5.4	1.2

Notes:

- Requirements in sections [RD01 - 10.1 Standards and Procedures], [RD01 - 10.2. Computer hardware standards] and [RD01 - 10.3 Software] are mostly not directly traced in the document. They are in any case taken into account and satisfied by the choices done for the reference products listed in [RD01 - 10.5 Reference products].

- Requirement's section [RD01 - 14. Requirements for applications] is in principle N/A to this document, since states requirements for applications and not for ACS. Nevertheless, ACS will try to support as much as possible applications in satisfying these requirements. For this reason a few sub-sections are directly traced in the document.

- Requirements [RD01 - 6.1.6 Command Delivery] and [RD01 - 6.1.9 Intermediate reply] are not satisfied by ACS. As discussed during the review phase, they are left for the time being to the responsibility of applications.

- The concepts of quality[RD01 - 4.1.7 Quality] and groups [RD01 - 13.1.3 Groups] has been removed and a change request has been submitted for the requirements document.

- The following table provides a trace of the requirements on ACS expressed in TICS Design Concept [RD26] into this document.

Item **in TICS Design Concept**[RD26]	**ACS Architecture**	**Release**
6. Data Channel	3.6	1.0
7. Configuration Database	3.5.3	(0.0) 1.2
8.1. Scripting	3.17	(1.0) 1.2
8.3 GUIs	3.16	(0.0) 1.0
9.3. Time System	3.9	1.0
10.5.1. Device	3.5	0.0
11.1. Monitor point collecting	3.12	1.0
14.1. Logging	3.8	1.0
14.2. Errors	3.7	1.0
14.3. Alarms	3.14	(1.0) 1.2