

COMPUTING

B. Glendenning

R. Lucas

G. Raffi

G. Harris

J. Schwarz

M. Brooks

J. Pisano

G. Chiozzi

M. Zamparelli

Last changed 2002-JAN-04

12 COMPUTING	3
12.1 INTRODUCTION.....	3
12.2 SCIENCE SOFTWARE REQUIREMENTS	4
12.3 ARCHITECTURAL OVERVIEW	6
12.3.1 Physical.....	6
12.3.2 Logical.....	9
12.4 DATA FLOW SOFTWARE	15
12.4.1 Observing Proposal & Programme Preparation	16
12.4.2 Observation Scheduling.....	17
12.4.3 Observation Execution & Interface to the Array Control Software	18
12.4.4 Near-real-time Pipeline Processing and Feedback to Operations	18
12.4.5 Archiving and Final Data Reduction	18
12.4.6 Acknowledgments	19
12.4.7 References	19
12.5 CONTROL SOFTWARE.....	19
12.5.1 Overview	19
12.5.2 Device (M&C) Interface.....	21
12.5.2.1 CAN Interface.....	21
12.5.2.1.1 Bus Master Nodes.....	21
12.5.2.1.1.1 Bus Slave Nodes	21
12.5.2.1.1.2 AMBSII	22

12.5.2.1.1.3	AMBSI2.....	22
12.5.2.1.1.4	References.....	23
12.5.3	Correlator.....	23
12.5.3.1	Summary.....	23
12.5.3.2	Correlator Computer Hardware Overview.....	24
12.5.3.3	Correlator Computer Software.....	25
12.5.3.4	Data Processor Computer Software.....	26
12.6	TELESCOPE CALIBRATION.....	27
12.7	POST-PROCESSING SOFTWARE.....	28
12.8	COMMON SOFTWARE.....	28
12.8.1	Overview.....	28
12.8.2	Technologies.....	28
12.8.3	Services.....	29
12.8.4	References.....	32
12.9	SOFTWARE PRACTICES.....	32

Revision History:

2000-Sept-29: Complete rewrite.

2000-Oct-06: Correct some typos and phrasing.

2000-Nov-22: Update data rate requirements, ZASI description.

2001-Jan-22: Reflect new AMBSI names, replace ATM with Gb Ethernet.

2001-Dec-12: Update correlator data rate description and references.

2001-Dec-13: Correct AMBSI2 description, update AMB references.

2002-Jan-03: Small revisions to update common software section, correcting for changes to the reviewed architecture document. Major expansion of software engineering description.

2002-Jan-04: Minor update to dataflow description. Update the SSR reference and data rates (the latter in the introduction as well).

12 COMPUTING

12.1 Introduction

B. Glendenning (NRAO)

Last Revised: 2002-01-04.

This chapter describes the operational software required for ALMA. This includes real-time and near-real-time software to monitor and control hardware devices, software to schedule the array, software to format the data suitably for post-processing, software to archive and restore the data, software to perform fundamental calibrations (*e.g.*, pointing) required to operate the array, commissioning software (*e.g.*, holography), and software to implement a near-real-time image pipeline. It does not generally include post-processing software, firmware that is "inside" the device (excepting the correlator), or engineering test software that is not needed during operations.

ALMA Computing, principal requirements

Sustained data rate, science data	6 MB/s (Average). Goal: 12MB/2 60 MB/s (Peak sustained). Goal: 72 MB/s.
Image pipeline	First-look images produced automatically for standard observing.
Dynamic scheduling	Nearly automatic scheduling of the array, accounting for current weather and other conditions, to optimize the scientific throughput of the array.
Archiving	Networked archive of all ALMA raw science data and associated calibration data and derived data products.

12.2 Science Software Requirements

R. Lucas (IRAM)

Last Revised: 2002-01-04

The scientific requirements for ALMA software are described in software memo ALMA-SW-0011¹, from which we reproduce here the introductory remarks. The reader is encouraged to read this report, which includes formal requirements and execution scenarios (Use Cases). Specific requirements for off-line data processing software are being prepared in a separate memo.

The operation of ALMA will have to deal with a larger variety of projects than previous instruments: on one hand at long wavelengths (1-3 mm) due to the high sensitivity and quality of the site, and a long experience with millimeter-wave interferometry, we can predict with reasonable certainty the observing modes that will be used, the relevant observing strategies to schedule the instrument, and the data reduction techniques. On the other hand at the highest frequencies (~300 μ m) no array has been operational yet; we plan to rely on techniques such as radiometric phase correction, fast phase switching and phase transfer between frequency bands, that have been demonstrated, but not applied with the operational scale that we foresee for ALMA. We thus will have to combine in the software a high level of automation, needed to deal with the large information rate that will be available, with a high level of flexibility at all levels to be able to develop and implement new observing methods and reduction procedures. For simple projects, the astronomer with little or no experience of radio techniques should be able to use the instrument and obtain good quality results; however, experts should easily be able to perform experiments we do not even foresee today.

The expert user/developer will need to be able to send direct commands to the instrument through a simple, easily editable command language. Atomic commands in a script language will directly send orders to the basic software elements controlling the hardware: antenna motion, instrument setup, or transmitting parameters to the data processing (pipeline). The script language will support loops, structured conditional tests, parameterized procedures, global variables and arrays ... These scripts, once fully developed and tested, will evolve into the basic observing procedures of the instrument.

The general user will need more user-friendly graphic interfaces to many components of the system. They will propose several templates, corresponding to the available observing modes, and provide a simple way to pass astronomy parameters to the basic observing process and to the corresponding data reduction procedures of the pipeline. Input parameters will preferably be expressed in terms of astronomical quantities, which will be translated into technical parameters by sophisticated configuration tools.

Proposal submission will be in two phases, the first before proposal evaluation, the second to provide information needed for the actual scheduling and observation.

¹ <http://www.alma.nrao.edu/development/computing/docs/joint/0011/ssranduc.pdf>

We believe that dynamic scheduling is an essential feature of the instrument and should be installed from the very beginning of its operational life. Though the site is undoubtedly one of the best for submillimeter observations, it will only be usable at the highest frequencies for a fraction of the time; to improve the total efficiency we must be able to make the best use of all weather conditions, by selecting in quasi-real time the project most suited to the current weather and to the state of the array. This means we should always be able to observe a given project in appropriate weather conditions. This philosophy can be extended to the point where a given project can change its own observing parameters according to variations in observing conditions (such as atmospheric phase rms).

The whole real-time system will be under control of a telescope operator, through a specially designed interface. This must provide an overview of what observation is occurring, the state of the instrument, and observing conditions on the site, and should enable the operator to react to any unexpected event. A general monitoring interface must be also accessible through the

The instrument should produce images, aiming to be final for most projects, even when projects are spread over several sessions and configurations, and/or include short/zero spacings. For this purpose an on-line pipeline is required. It will include calibration of the array itself, to reduce measurements of baseline, delay offsets, and determine pointing models during specific sessions. During standard observing sessions reference pointing and focusing measurements will have to be reduced, with fast loopback of results to the observing process; the phase fluctuations on the phase calibrators must be evaluated, with a feedback to both the real time process and the scheduler. Calibration will be applied on-line and maps/datacubes will be produced according to data processing parameters input by the observer. Single-dish observing sessions will also be reduced on-line. The pipeline must be able to reduce on line the quasi totality of the data, which is expected to be produced at an average rate of 12MB/s^2 , with a peak rate of 72MB/s for some observing sessions.

For most projects the data pipeline will produce results in a form suitable for quality evaluation, and astronomical processing, hopefully leading to fast publication. Uncalibrated UV data will be archived together with the calibrations curves and the resulting images. The archive should enable fast access to the observing parameters and full reprocessing of the data set with improved processing algorithms.

A general requirement is that the various parts of the system should be developed in a highly consistent way, from the very beginning of design; they may however be installed progressively, provided the critical elements are implemented first.

² These values were updated in late 2001 and are goals of the project. The specifications however are unchanged: 6MB/s average and 60MB/s peak sustained.

12.3 Architectural Overview

12.3.1 Physical

B. Glendenning (NRAO), G. Raffi (ESO)

Last Changed: 2001-01-22

As shown in Fig. 12.1, the ALMA computers and networks are organized around a few geographical areas:

- Antenna computers, which are repeated and essentially the same at the 64 antennas
- Central computers at the ALMA site, with central control and coordination functions linked to the whole array.
- Operation Support Facility (OSF) computers to offer user and maintenance interfaces to the whole array from San Pedro.
- Data centers in Europe and USA with archiving and remote access functions
- Other computers accessing ALMA via Internet. Although these should be seen as external to the ALMA project, their presence and functions might have to be supported via certain categories of software and so they have to be considered.

It is important to note that operation at OSF is considered as local operation, i.e. all normal and maintenance operations are available at the OSF with full functionality and performance. The only reason to perform maintenance at the ALMA site has then to be the need for physical access, replacement or movement of parts, but not any limitation in the capabilities offered at the OSF. These should include video and audio links to the central computers at the ALMA sites and possibly to the antennas.

Figure 12.1. Overview of ALMA Computer Systems and Networks

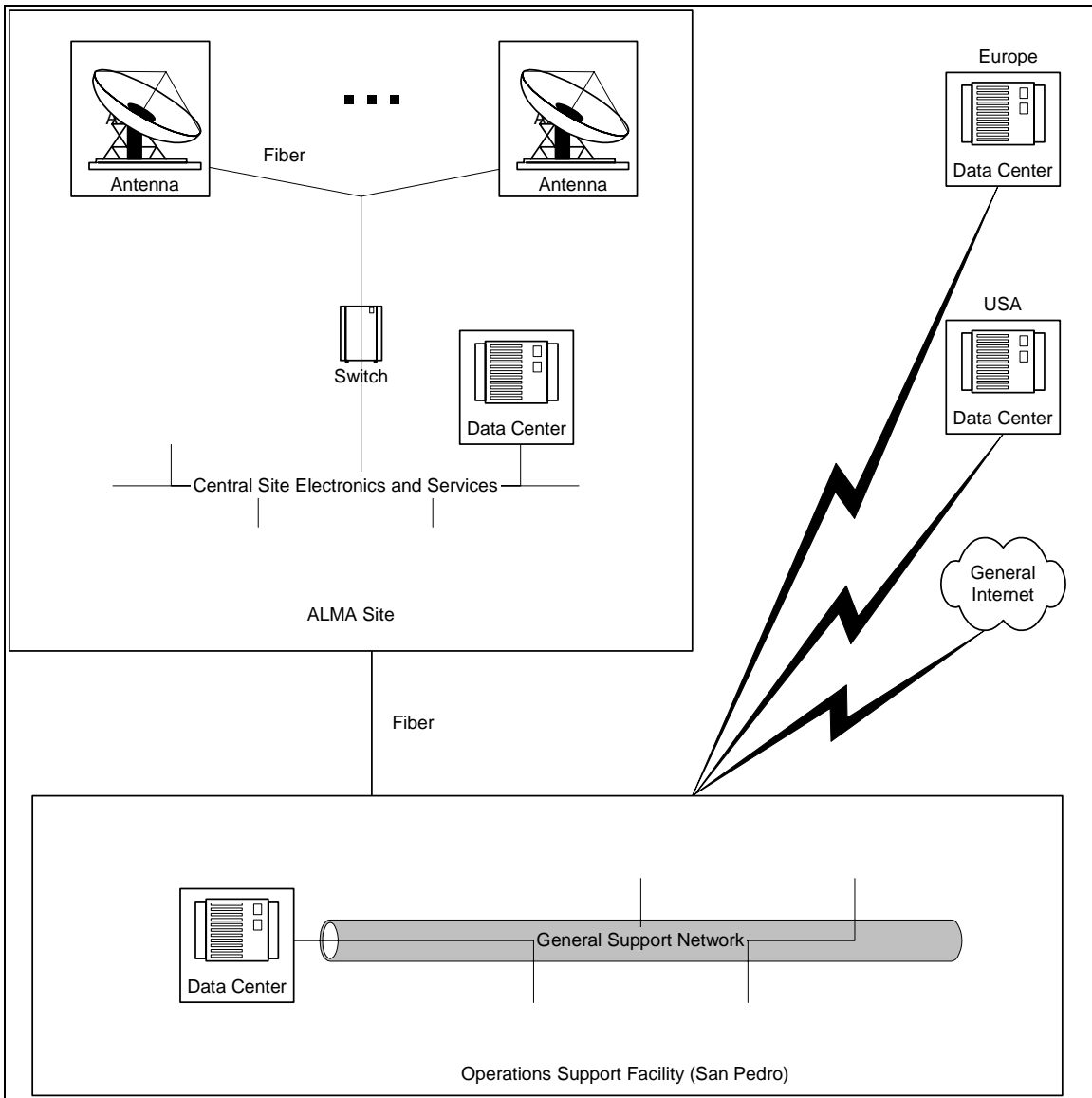
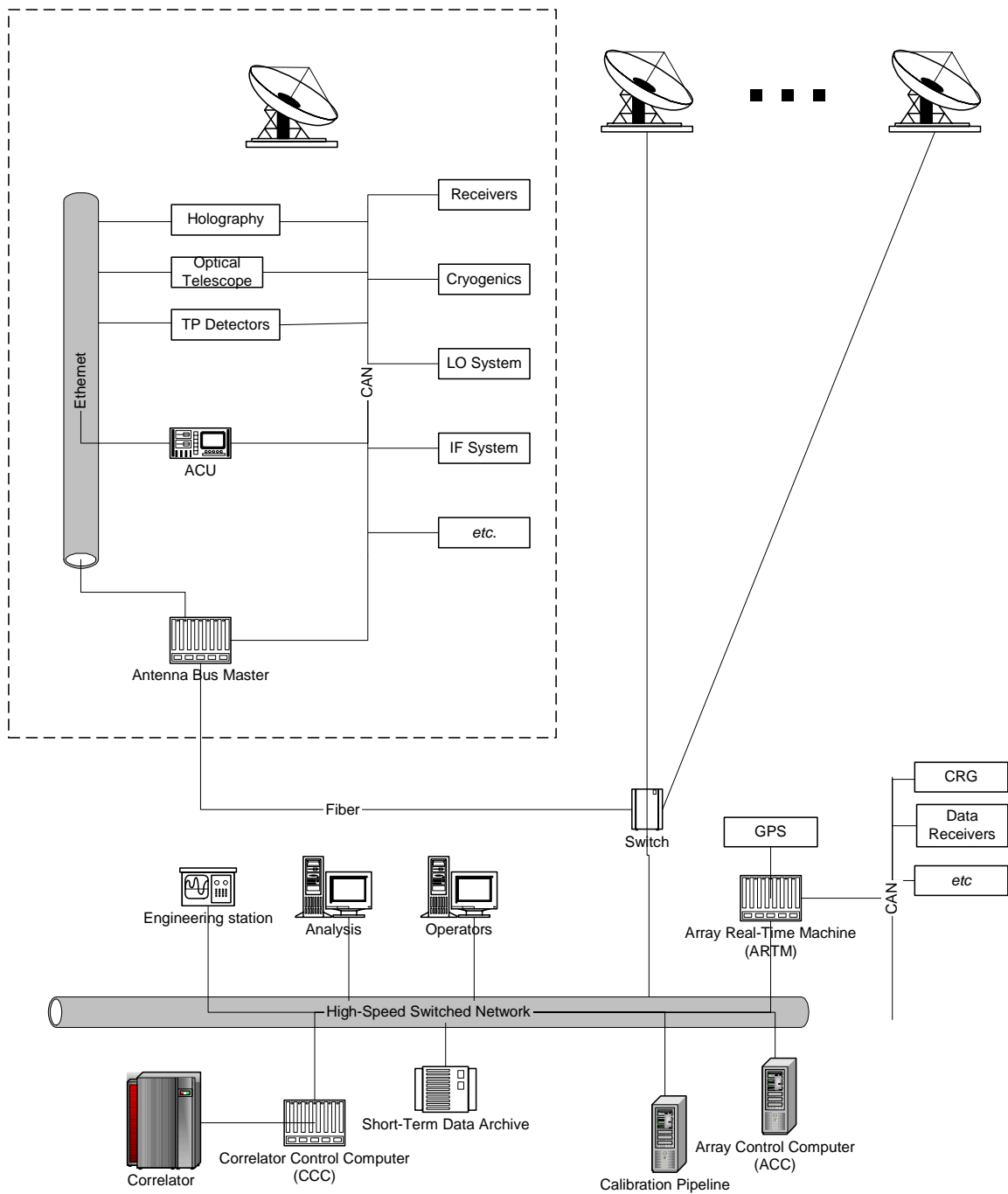


Figure 12.2. Layout of computers at the site



At each antenna there is an Antenna Bus Master (ABM) – a VME Power PC based VxWorks computer. Its principle role is to provide real-time control of the devices at the antenna based upon infrequent time-tagged commands from the center. It serves also as a router for the antenna Ethernet segment.

All devices with computer interfaces are attached to a Controller Area Network (CAN) bus, through which they are controlled and monitored. More details about the properties of these interfaces are described under Monitoring and Control. A particularly important device is the Antenna Control Unit (ACU) that is provided by the antenna vendor. It is implemented as a VME/PPC/VxWorks computer. While like any other device it is controlled over the CAN, it also has an Ethernet interface for software maintenance and the setting of static parameters that do not need to be changed in normal operations.

It is assumed that data transmission from the total power detectors and holography backend is done over Ethernet.

The optical telescope (for the antennas where this will be mounted) is also commanded via the CAN bus. Its video output will be digitized and transmitted over Ethernet and via the ABM to the central computers.

Each ABM is connected to the central systems via a point-to-point Gigabit Ethernet network that terminates at a switch. The switch is in turn connected to a high-speed switched network on which all central ALMA computer systems required to operate the array are attached.

At the center there are two real-time computers. The Array Real-Time Machine (ARTM) plays the role of the ABM, providing local real-time control of its attached CAN devices. The Correlator Control Computer (CCC) is the other central real-time computer. It provides the interface for the correlator, and provides detailed control of the correlator hardware. Both the ARTM and CCC are VME/PPC/VxWorks based systems.

The coordination function is implemented via the Array Control Computer (ACC), which is a high-end workstation running the Linux operating system. It is responsible for controlling all hardware in the array (indirectly through the ABM, ARTM, and CCC computers) under the command of a high-level observing script. It also runs various ancillary software such as model (*e.g.*, delay) servers, data formatting, *etc.* If necessary for performance reasons the ACC functions could readily be split into multiple computers.

There will be a few general purpose Linux systems on the switched network for operator access, astronomical data analysis, software development, and the like. Computers with Labview will be used for engineering and maintenance purposes.

Data from a meteorological data station will also be available at the OSF.

12.3.2 Logical

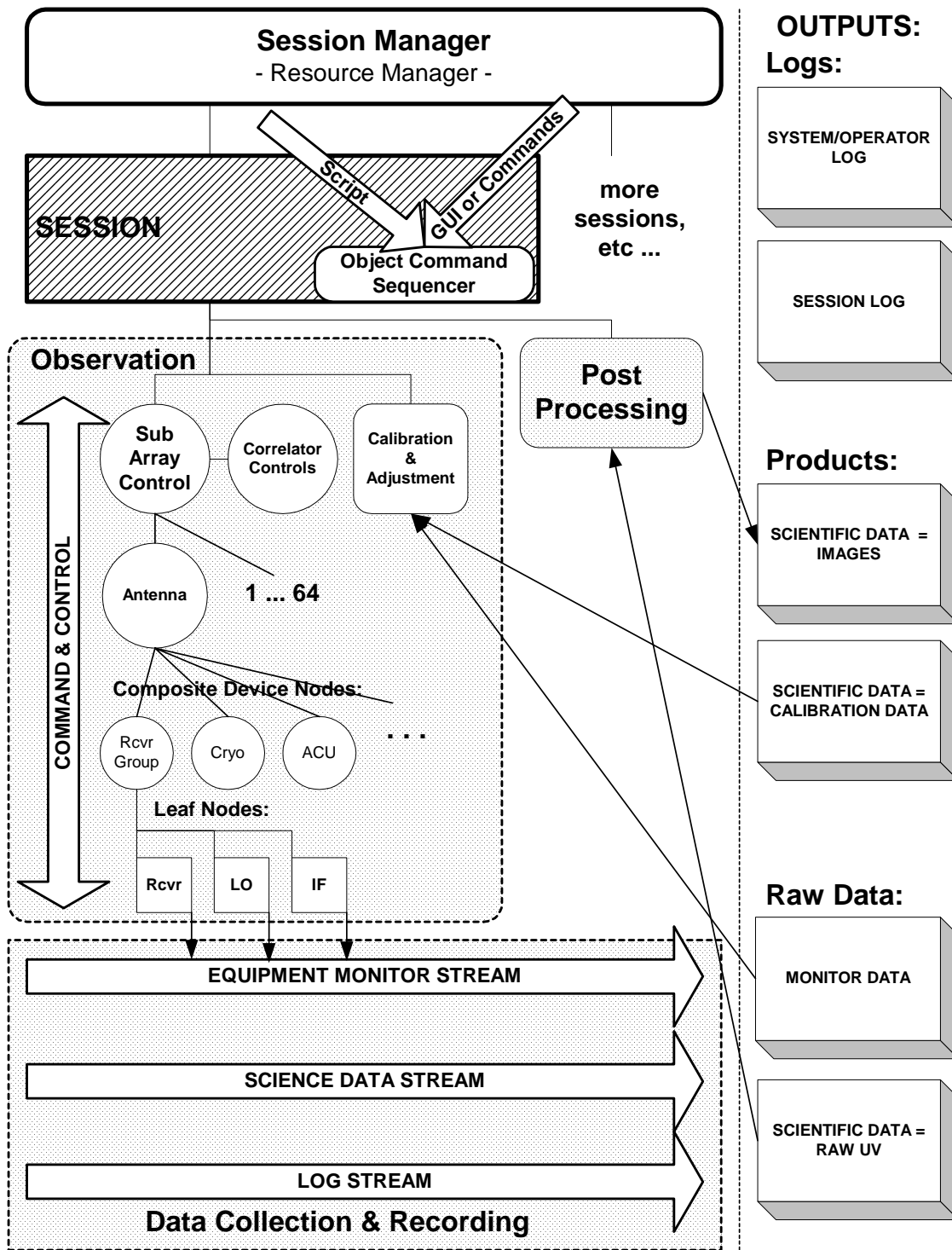
G. Harris (NRAO)

Last Changed: 2000-09-28

The ALMA Control Software System has a number of concepts, among them are session, sequence and data products. The Conceptual Diagram portrays some of these important notions.

Figure 12.3. ALMA Control Software System - Conceptual Diagram

ALMA Control Software System - Conceptual Diagram



First of all, there is a session manager or executive, described below, which creates observer work sessions and allocates resources such as antennas, correlators and post processors to them.

This observer session will perform observations and perform other processing. It is directed by a programmatic script created in advance by the observer, or entered directly in interactive mode through a GUI or commands. These commands will expand through the action of the software system into a sequence of device commands.

While operating an observation, for example, control commands will be issued to a software structure as shown in the shaded box labeled *Observation*. This structure operates the sub-array of antennas and correlator as a unit, issuing device operations in a timed sequence. Then later, the same session might drive a *Post Processing* operation to process the raw data placed in the archive by the *Observation* activity and make it into finished images.

The *Observation* box shows some of its major activities by component. First, the commands flow down the structure of sub-array and antennas in a hierarchical manner. The further hierarchical structure of device groups and devices, shown in the composite and leaf device nodes, leads us to name this structure a control tree, as shown in later drawings. Each tree has groups of devices, such the receiver group with receivers, filters, LOs and IFs. Ultimately, the software tree reaches down to leaf nodes connecting to hardware.

As the observer operates the instrument, the observer's session will create *Outputs* as shown in the diagram. There will be logs, both of individual sessions, and of the system as a whole. There will be raw output from an observation plus processed images and calibrations. Some of these activities may be in parallel, allowing feedback into observation parameters.

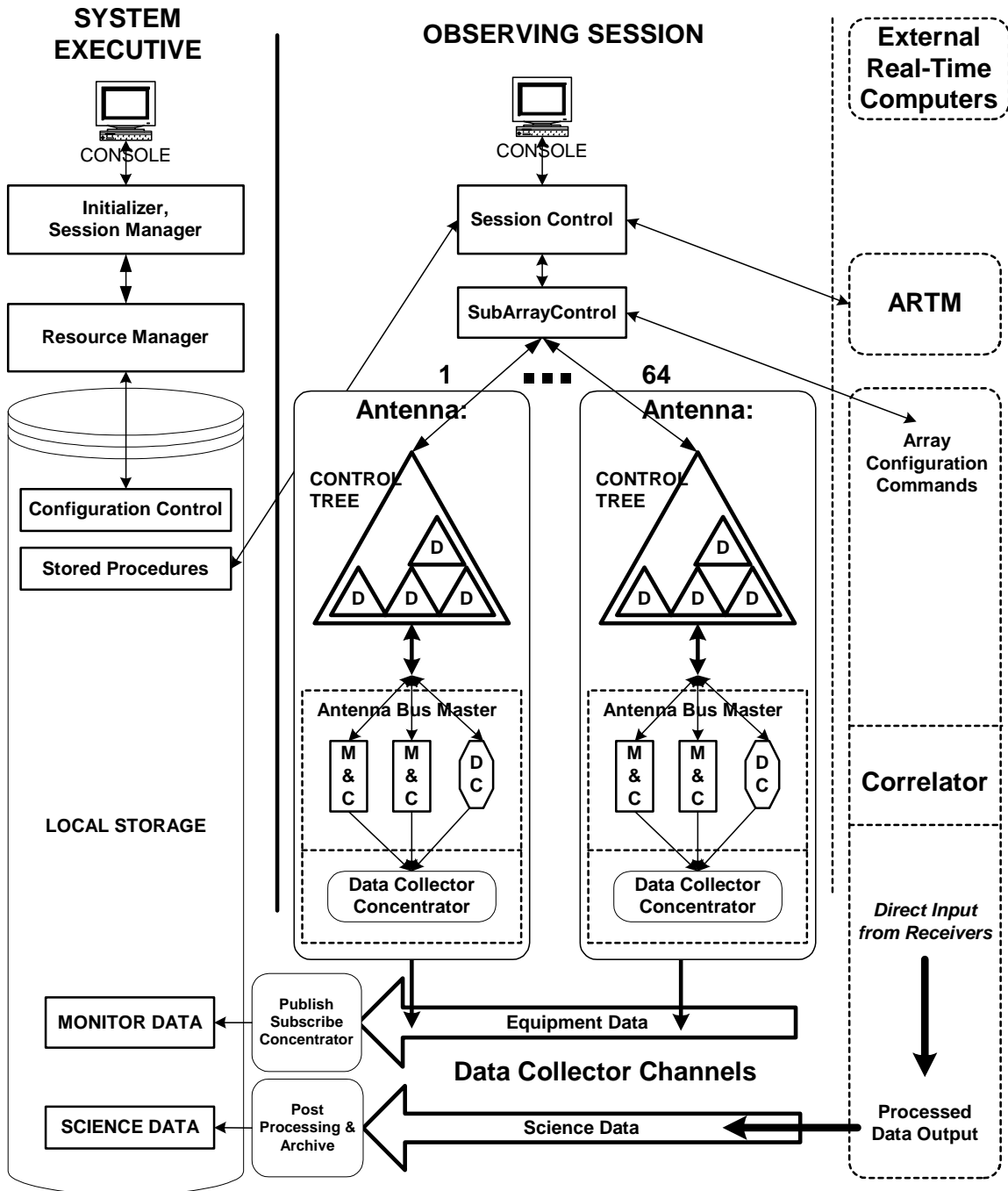
The system also needs to operate independently as observer sessions come and go, keeping logs and equipment status, performing maintenance and calibration and other activities. Supporting this is *Data Collection and Recording*, which operates at the system level. This is shown by another shaded box in the diagram. Data collection from the equipment is not dependent on a control session, nor vice versa.

The collection process streams information from the antennas to central storage and makes it available for the system as well as the observer sessions when they are operating. There is equipment information: temperatures, frequencies, etc.. There is science data processed by the correlator. There is also log information, constantly from the system and operator, and periodically from observer work sessions. There is a method, described below, to subscribe to the information published by the data collection process.

Now moving from the general concepts of session, sequence and data products, look at the details of session operation as shown in the figure for Logical Architecture.

Figure 12.4. ALMA Control Software System Logical Architecture

ALMA Control Software System Logical Architecture



First of all, ALMA has a supervisory program called the system executive. The executive initializes the instrument, creates processes for observers called work or observing sessions, allocates resources, and launches services which collect and process data. It also supports operator activities.

The action is similar to initializing an operating system:

- During startup, a resource manager locates and initializes the devices, sometimes using information from a configuration control database.
- The system processes are started and necessary communications connections established, making devices available to processes in the system. This is especially important for services such as object component naming, which supports various parts communicating with each other.
- The initial allocation of these resources is performed, to a resource pool or to an observer.
- Then observer processes are started to use the resources and perform work. These are typically called sessions.

There may be many observer sessions, operating simultaneously. There may also be additional operator work sessions, as the executive will also provide for some maintenance and configuration activity.

Using the logical architecture diagram, we can see how this will operate. The operator will start the executive after operating system startup.

The executive will first start the resource manager. The resource manager will be responsible for using the configuration database to initialize the antennas. Each antenna will be completely initialized, including the control tree and Antenna Bus Master [ABM], and allocated as a unit to a work session. This process will also register all the objects [devices] in the antenna in the object naming system.

Once these devices can be identified, the executive can also start the data collection process in each antenna's ABM computer. The ABM collector concentrates the equipment information from about a thousand points per antenna. There is also a central collector, which gathers the data from each antenna. [There will be 64 antennas.] This is called a publish/subscribe mechanism. Once the data is collected into the central Array Control Computer [ACC], other processes may subscribe from this central publisher.

With all antennas initialized and the data collector operating, the executive can start work sessions for the observers and give them appropriate resources depending on the desired observing mode. When a work session terminates, the executive will clean up the resources and possibly re-initialize them or place them in a known state for another session. Operator sessions or direct executive commands may be used for various standard activities, parking or stowing, shutdown, special calibrations etc.

[Now look in the drawing at the section labeled SESSION.]

Using the work or observing session, the observer enters commands or retrieves stored commands on the system if in interactive mode. If in automatic mode, prepared observing scripts are executed in the order presented by the system scheduler. The observing commands are propagated into the SubArrayControl and then to each antenna through a control tree, a software construct representing the devices on the antenna.

The method by which this is done is called CORBA [Common Object Request Broker Architecture], an industry standard. Use of distributed object oriented technology is quite extensive in this instrument. Every possible piece of the software is object based, from the configuration data, to the scripts and their functions, to the devices in the control tree and even to the hardware device control points on the antennas. This facilitates the creation, management and execution of the control software.

Some devices are simple and others are treated as a group. The letter D inside the control tree triangle represents a Device, perhaps a composite hierarchy of receiver parts such as LO, filters, etc. The hierarchy of devices, represented by the triangle, expands commands as they propagate through the control tree. Sometimes a simple command by the observer results in many commands to the bottom level devices.

Some software devices are in the central computers, in what may be called the logical time part of the system. Other devices are in external computers such as the correlator, the Array Real Time Machine [ARTM - the time server] or even out on the antennas, next to their associated physical devices in what we call the ABM [Antenna Bus Master]. **These external computers are represented by dotted lines and operate in real time.** The dotted line between the control tree of an antenna and its ABM out on the antenna represents the transition from logical time to real time.

Once commands reach the ABM, they go to appropriate Monitor and Control points [M&C]. Monitor points read the value of a single hardware access point and Control points can set the value of a single hardware access point.

Sometimes a behavior is needed in real time which physical devices do not have. Rather than trying to perform this more complex behavior remotely from the control tree, a software construct called a device controller [DC] is used. For example, a nutator sequence may be given as a single command from the command tree resulting in many actions by a device controller in the ABM, all synchronized in time.

Another feature of this architecture is the decoupling of the control system from the data recording. The control system is not dependent on the data monitoring and recording system. Or vice versa. Equipment may be monitored even when not being used for observations. Just as the antennas are configured and initialized before giving them to a work session, so also is the data collecting system turned on and initialized by the system executive.

The correlator is accessed on two levels. First it is configured on an array basis and commanded as shown in the drawing by the SubArrayControl at the subarray level. As data coming from the receivers is sent to the correlator, it is processed and the correlator output goes into the science data.

Another independent computer with support electronics is the Array Real Time Machine [ARTM]. It coordinates with external clock sources, internal frequency generators and other equipment to provide a timing standard to the array.

When operating in interferometric mode, a single session drives many antennas constituting a sub-array. The SubArrayControl distributes the current command from session control to the antennas, correlator, etc.. Generally all antennas receive the same command, but each can also be individually addressed. Any per antenna differences in processing a command are handled by each antenna's control tree and ABM.

Other activities not shown in this diagram for simplicity are:

- Display sessions using the information from the central data publisher,
- Observing proposal preparation,
- Observation Scheduling and Dispatch,
- Image Processing Pipeline, and the
- Feedback from monitor data, and Image Processing to observations
- Archiving and distribution of data.

12.4 Data Flow Software

J. Schwarz (ESO)

Last Changed: 2002-01-04

The Data Flow Software will encompass the end-to-end processing of an ALMA Observation, from the preparation of a proposal by a prospective observer, through to production and archiving of cleaned images/data cubes from the array. Support will thus be provided to:

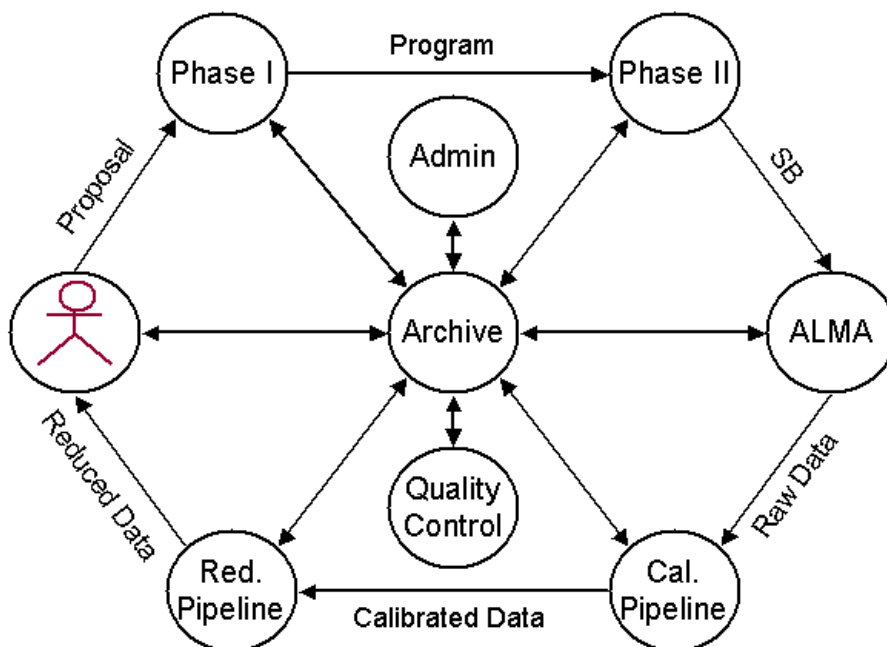
- The proposer/observer;
- The proposal reviewers and those responsible for granting ALMA observing time;
- The array operations staff;
- The archival researcher.

The Data Flow Software will provide:

- Tools for the preparation of observing proposals; these tools will allow the proposer to specify his/her scientific requirements insofar as possible. The tools themselves will then be responsible for translating these requirements into array-level configurations and observing scripts.
- A dynamic scheduler to maximize the observing efficiency of ALMA in service mode. Observations will be scheduled based on assigned scientific priority, as well as appropriate match to the existing observing conditions. Some (user-specified) observing parameters may be adjusted to take account of changes in these conditions.
- A near-real-time pipeline, capable of feeding back results to the observing process, and of producing quick look (“dirty”) images for inspection.
- A post-acquisition pipeline that will produce fully calibrated and deconvolved images that should be of final quality in the majority of cases.
- An archive to store *uv*-data, images/data cubes, calibration data, observing programs (including proposals, observing instructions (as produced by the observer and as actually executed, and observation parameters).

A simplified view of data flow through the ALMA system is given in Figure 12.5.

Figure 12.5. Schematic Data Flow in ALMA



12.4.1 Observing Proposal & Programme Preparation

In order to ensure that ALMA is accessible to the largest possible segment of the astronomical research community, the SSR has decided that prospective observers should be able to input their scientific desiderata (*e.g.*, angular resolution, spectral resolution, field of view, and signal to noise ratio) and that the software tools provided by the Observatory should evaluate these for consistency and feasibility, and derive the necessary Array parameters--from configuration through correlator setup and selection of calibration sources to total observing time.

At the same time, there will be a mode that allows experts to bypass this feature of the tool entirely, specifying in detail the low-level observational parameters to be used. The SSR envisages that a scripting language will enable direct control of the Array. The conditions under which modification of accepted programmes is to be allowed is under discussion.

How Observatory Operations will coordinate these two types of programmes is TBD; in the case of VLT Service Observing, only a limited set of templates are available to the Observer, and low-level control of the telescope by, for instance, a user's *ad hoc* script, is not allowed. Observation Blocks, the basic units of VLT observing programmes, are vetted individually by the operations staff. Modification of accepted Phase II Programmes by the observer is only allowed under special circumstances. "Expert" observing privileges on ALMA may need to be restricted; in any case, the most common use of Expert mode is expected to be for

the development of new observing procedures by the ALMA scientific and operations staff; once successfully tested, these procedures will be included in the palette of standard modes offered by the GUI-based Observing Tool.

12.4.2 Observation Scheduling

Although interactive observing will be supported by ALMA, it is expected that the Array will usually be operated in Service Mode. As in the case of the VLT, the argument here is that ALMA is so expensive an instrument that every effort should be made to maximize its observing efficiency.

In spite of the existence in the observing community of sophisticated software scheduling tools, scheduling remains as much an art as a science. At the VLT, where the Spike inference engine developed for the Hubble Space Telescope was adopted for long- and medium-term scheduling, the final decision about which Observation Block is next to be executed is determined manually by the Operations Staff. Spike is used primarily as a visualization tool at the VLT. It is quite complex, and as it is written in Lisp, requires that comparative rarity, a Lisp programmer, to maintain it. A commercial version, written in C/C++, is also available, and its purchase might alleviate the maintenance problem. Until this is done at the VLT, automatic scheduling is likely to remain a low priority. The VLT Operations Team has not found the manual scheduling procedure to be an excessive load on their manpower resources so far.

On the other hand, the radio astronomy environment may be more hospitable to dynamic scheduling. Wright (1999) described his dynamic scheduler for BIMA, which selected observations on the basis of weather conditions, as a technical success. The algorithm for ALMA, which will include the possibility of modifying some observational parameters (*e.g.*, dwell time on phase calibrators) in response to changes in atmospheric conditions, will be more sophisticated, however. Such decisions will depend upon near-real-time feedback from one of the data reduction pipelines (see below).

Although scheduling “what comes next” may be relatively simple, VLT experience demonstrates that what seems like tonight’s optimal solution may cause problems (*e.g.*, a schedule “hole”) weeks or months later. A scheduling tool capable of generating possible timelines over longer periods than just one night is likely to be needed. Moreover, no schedule can be optimized over a several-month period unless the awarding of telescope time takes into account the distribution of hour angle ranges and observing conditions that will be available.

Another requirement is that the system should be able to schedule short programs (“snapshots”) with common calibration requirements in such a way that observing efficiency can be maximized. This will require knowledge by the system of the calibration requirements of each program, as well as of the resources, especially observing time, needed to satisfy them.

12.4.3 Observation Execution & Interface to the Array Control Software

The interface to the Array Control Software is not yet defined and will be discussed in a future revision of this document.

12.4.4 Near-real-time Pipeline Processing and Feedback to Operations

There is a requirement to feed back the results of *e.g.*, pointing and phase calibrations to Observatory Operations in near real-time. Some form of quick-look imaging will also be needed, though definitive image production (including deconvolution) will need to wait for the results of calibrations that may not be available simultaneously with the science data, as well as the complete science data set itself, which may span more than one array configuration, and whose acquisition may span several months.

Design and performance of the pipeline will be discussed in a future revision of this document.

12.4.5 Archiving and Final Data Reduction

The ALMA Archive must serve two purposes:

1. To support observatory operations and to provide the observer with the data he/she wants and needs in a timely fashion.
2. To support archival research by users *other* than the original observer.

Whether this will require two archives, a “short-term” and a “long-term” one remains to be seen. The SSR has proposed allowing each observer to decide which data he/she wants archived, subject to a limit determined by the correlator data rate (over the long term this is estimated to be 3 Mbytes/sec). If both corrected and uncorrected *uv*-data are to be retained, the effective rate goes to 6 Mbytes/sec, implying that of order 180 Terabytes must be archived each year. (Recently revised estimates of the data rate give 12 Mbytes/sec average and 72 Mbytes/sec peak, but these have yet to be accepted by the Project as formal requirements for budgetary reasons. Future advances in computer performance and storage capacity coupled with price-benefit improvements may make this point moot.) In any case, an archive based on individual observers’ decisions is likely to be very inhomogeneous, and thus may not be suitable as the official ALMA long-term archive.

In addition to *uv*-data, the archive will include images produced by the automatic pipeline reduction system, as well as complete information concerning the conditions under which the observations were performed and the data reduced. On the assumption that ALMA will have adequate computing resources for the task, it will be possible for the archive researcher to request reprocessing of the archived *uv*-data with improved calibration data and tailored algorithms.

12.4.6 Acknowledgments

Dave Silva, Bruno Leibundgut, Michele Peron and Peter Quinn of ESO's Data Management and Operations Division have been generous of their time in discussing the VLT Data Flow experience with me.

12.4.7 References

1. R. Lucas, B. Clark, J. Mangum, P.Schilke, S. Scott, F. Viallefond, M. Wright, "ALMA Software Science Requirements: Version 1.0 Report", ALMA Memo #293 (Revised), 2000.
2. Lucas, R. et al, *ALMA Software Science Requirements and Use Cases*, Revision 3, ALMA-SW-0011, 2001.
3. M.C.H. Wright, "Dynamic Scheduling", ALMA Memo #282, 1999.
4. Schwarz, J. et al, *Initial Software Analysis*, undergoing revision, ALMA-SW-xxxx, 2001.
5. Chiozzi, G., Gustaffson, B. and Jeram, B., ALMA Common Software Architecture, ALMA-SW-0016, 2001.

12.5 Control Software

12.5.1 Overview

B. Glendenning (NRAO)

Last Updated: 2001-01-22

This section summarizes the design for the ALMA Array control software. At present, it is very closely modeled on the test interferometer design. As we gain experience with the test interferometer, the designs will separate (for example, the sub-array facilities for the ALMA array will be very much more elaborate).

In general, we would describe this design as conventional in the sense that designs like it have been implemented for many recent telescope control systems. There are no drivers that we know of that would require us to come up with a radically new design.

Fundamentally, the role of the control software is to take high-level observing commands from the scheduling system and to turn these high-level commands into detailed control of the array, finally producing science data products and monitor data archives. In brief, the scope of the control software is "scheduler to data."

Almost all devices will be attached to a CAN bus operating in a master/slave (polled) fashion. The bus will operate at 1Mbps and is capable of at least 2000 polled operations per second (up to 8 bytes of data per transaction). Devices on the CAN bus will be responsible for implementing a simple in-house protocol to map CAN message IDs to internal device addresses. A few devices will have other connections, in particular Ethernet.

The CAN buses are attached to (mastered by) a Power PC (PPC) VME based computer running the VxWorks real-time operating system. There is one ALMA PPC system at each antenna, and two at the center. One central system is used to master the central CAN devices. The other system is embedded within the correlator. Centrally there is a general-purpose computer running Linux that is the overall master for the system, which is known as the Array Control Computer (ACC). There are also some ancillary systems in the center for, *e.g.*, operators to sit at.

The antenna-based systems are connected to the central systems via point-to-point Gigabit Ethernet network connections. Commercial off the shelf (COTS) solutions for Gigabit Ethernet are available that support 40km fiber runs and quality of service (QoS) guarantees. The central computers are connected with Gigabit or 100Mb (TBD) switched Ethernet on a network segregated from the rest of the networks site networks.

Logically, the software is partitioned so that control flows in a master-slave fashion from a central executive who controls high-level (“composite”) software devices that in turn control their constituent parts. The lowest level software devices are referred to as device controllers, and represent a proxy for the actual hardware – that is, they communicate with the hardware. Data – monitor and backend – is collected from the devices by a collecting process in the real-time computer attached to the hardware, and buffered up for distribution via a publish/subscribe mechanism to consumers of the data, which include the processes that format and archive the data. The software is distributed amongst computers so that only the device controllers and software directly concerned with low-level device activities are on the local real-time computer. All higher-level software entities are concentrated on the ACC.

Engineering access to devices that are installed on the test interferometer will be implemented via access to the device controller interface or to the I/O routines directly from engineering workstations.

The ALMA Time System will establish synchronized switching cycles and mode changes, and provide time-stamping for the resulting measurements. This must be done across the entire instrument including the central building and the geographically dispersed antennas. Additionally, the Time System must be accurately related to external measures of time to correctly determine the position of astronomical objects of interest. The fundamental time system of the interferometer is TAI time maintained in a central master clock. Although TAI is the internal system used by the software, other time systems such as UTC and local sidereal will be accepted from user input routines. Most devices do not have precise timing requirements. For those that do, the control software must arrange to have monitor and control commands sent to the device in precisely defined windows within the pervasive 48ms timing period. This is accomplished by sending time-tagged commands from the center sufficiently in advance of when they are required to account for the non-determinism in the network and general-purpose ACC. The commands are then staged in the ABM until they are required at the hardware. The slave clocks are given the array time of a particular timing event, and they maintain time thereafter by counting timing events.

There are several characteristic (“looping”) timescales of interest to the control software:

- 2ms: This is the shortest timescale at which any device will require interaction (the total power detectors). Timescales faster than this are always handled by hardware.

- 48ms: This is the period of the pervasive timing event sent to all hardware with precise timing requirements.
- 16ms: Fastest correlator dump time. (Autocorrelations achieve 1ms dump times but are “bundled” in 16ms increments).
- 1s: This is the fastest timescale for observational changes, e.g., source change or change in correlator setup.
- >1s: Most devices will be monitored or controlled at rates slower than 1Hz, often much slower (300s).

Some model values are required for some devices – for example, the correlator will need a delay model. These calculations will be encapsulated in “model servers” running centrally. The central control software will call these servers and send a parameterized result (e.g., polynomial as a function of time) to the device that needs them.

12.5.2 Device (M&C) Interface

M. Brooks (NRAO)

Last Changed: 2001-12-13

12.5.2.1 CAN Interface

The CAN bus has been selected as the primary interface to distributed hardware devices within the ALMA project. CAN is an ISO standard multi-drop communications medium used extensively in automotive and industrial applications. A higher layer master/slave protocol has been defined (ALMA Computing Memo #7, ALMA document ALMA08001Nx0001) which governs the behavior of slave nodes at the devices. This bus is hereafter referred to as the ALMA Monitor and Control Bus (AMB).

12.5.2.1.1 Bus Master Nodes

It is required that on any single AMB there be only a single bus master node. Bus master nodes have been coded and tested for the following hardware and software combinations:

- O/S: VxWorks. SBC: MVME2700, MVME1603, MVME2604. CAN Interface: Tews Datentechnik TP816 Dual and Single Port PMC CAN modules (available in the US from SBS Greenspring)
- O/S: Windows NT. Any PC platform with PCI. CAN Interface: National Instruments Dual Port PCI CAN board.
- O/S: Linux. Any PC platform with PCI. CAN interface: Janz CAN-PCI/K20

It is the responsibility of the ALMA Computing Group to design, implement and test all code developed for these bus master platforms.

12.5.2.1.1.1 Bus Slave Nodes

There will be two standard interfaces (SI) to the AMB, the AMBSI1 and the AMBSI2. All designers of hardware interfacing to the AMB should use one of these two; new slave node implementations may be considered where sufficient justification can be provided. All slave

node implementations must comply with the bus specification detailed in ALMA Computing Memo #7, which requires that no slave node initiate transactions on the bus unless polled by a bus master.

12.5.2.1.1.2 AMBSI1

The purpose of the AMBSI1 is to provide a highly flexible interface board with on-board CAN and device-side I/O consisting of parallel, serial and bit-wise ports. The board would be delivered to hardware designers with a standard firmware package supporting basic CAN access to the I/O ports and external bus. The micro-controller on the AMBSI1 will have sufficient spare processing capacity to run additional user-specific code, such as providing local displays. The ALMA Computing Group would be responsible for developing such code in conjunction with the hardware developer and for integration testing the product to ensure that the user code does not interfere with the CAN slave code. All AMBSI1 boards should be similar in hardware, and identical in size and connector arrangement. Code may be loaded into on-board flash memory by means of the CAN bus or the local RS232 port.

The CAN bus servicing code is written to be entirely interrupt-driven. It runs only in response to interrupts from the CAN controller, and those interrupts are set to the highest possible priority (in the Infineon C167, this is interrupt level 15, group level 3). When not servicing an interrupt, the processor executes an idle loop. User-written code may include a function to execute from the main idle loop, or it may simply consist of the callbacks for CAN message reception interrupts.

12.5.2.1.1.3 AMBSI2

This device performs the function of a CAN bus to Serial Peripheral Interface (SPI) converter. The AMBSI2 is designed for use in systems which either have their own microprocessors or need a very minimum amount of I/O to the CAN bus. The AMBSI2 consists of a small daughter PCB of approximately 1.4 by 1.8 in. This PCB would be mounted to the main hardware device board by several SIP headers that also serve as the I/O connectors for the subsystem.

The AMBSI2 provides a powerful 16-bit micro-controller with firmware that links the AMB to the device-specific micro-controller or microprocessor. The link between the AMBSI2 and the device is a three to five-wire serial based communication connection using the Serial Peripheral Interface (SPI). The main area of deployment for the AMBSI2 will be when the Device contains all the basic Monitor and Control functions and simply needs to pass information through a known interface to the AMB. Thus, the purpose of the AMBSI2 is to incorporate the AMB protocol and to link to the Device serially, thus acting as a known interface to the AMB and the Device.

The board would have a unique serial number (using the same Dallas part as the AMBSI1). In addition to providing the CAN interface, the unit would also provide the interface for both the RESET and TIMING pulses which come in through the AMB DB-9 connector or module wiring as RS-485 and be available as TTL signals to the application circuitry.

12.5.2.1.1.4 References

1. ISO 11898:1993 Road vehicles - Interchange of digital information - Controller area network (CAN) for high-speed communication
2. ALMA Monitor and Control Bus Interface Specification , ALMA Computing Memo #7, ALMA Document ALMA08001Nx0001, M. Brooks, L. D'Addario, Revision C, 2001-09-07
3. ALMA Monitor and Control Bus AMBSI2 Standard Interface Design Description, ALMA Computing Memo #12, ALMA Document ALMA08001Nx0007, W. Koski, Revision A, 2001-05-03
4. ALMA Monitor and Control Bus AMBSI1 Standard Interface Design Description, ALMA Computing Memo #13, ALMA Document ALMA08001Nx0002, M. Brooks, Revision A, 2001-05-11

12.5.3 Correlator

J. Pisano (NRAO)

Last Revised: 2001-12-12

12.5.3.1 Summary

This section describes the ALMA correlator control computer hardware and software. This computer system is meant to control the baseline ALMA correlator and process its lags. The real-time computer system consists of two parts. The control computer, a VME PowerPC computer running VxWorks, configures and monitors the ALMA correlator via a CAN interface. The data processing computer, a Beowulf cluster of COTS PCs running Linux with customized device drivers, interfaces to the ALMA correlator LTAs via a FPDP interface and converts raw lag results to spectra.

12.5.3.2 Correlator Computer Hardware Overview

Figure 12. 6

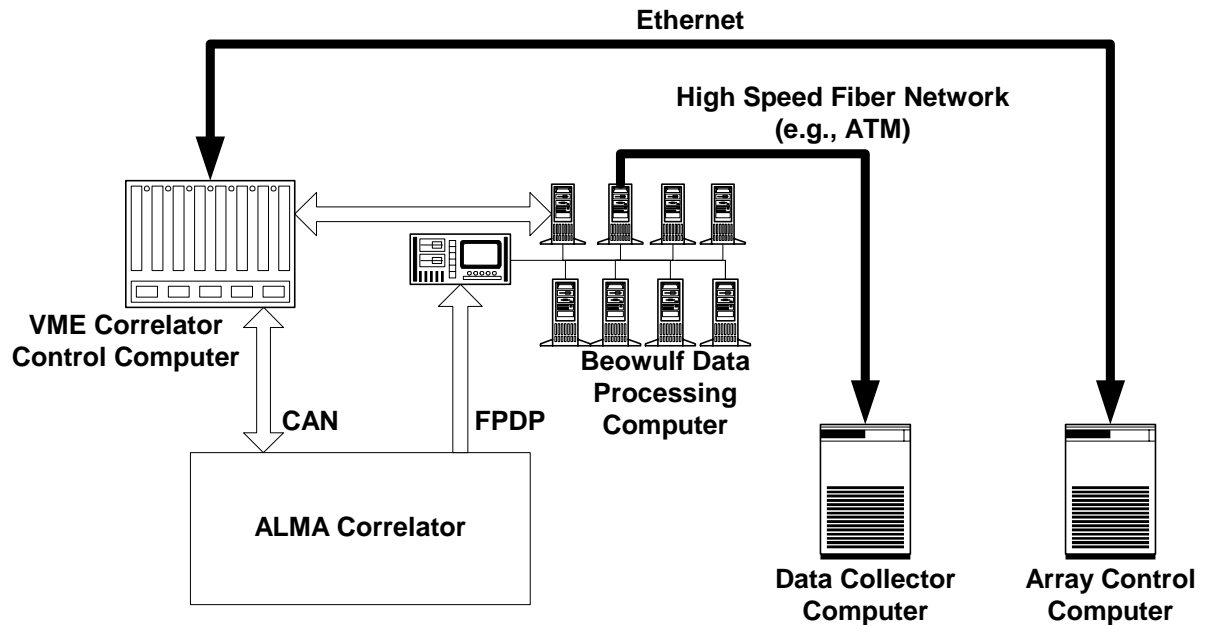


Figure 12.6 shows a block diagram of the correlator computer relative to other systems to which it interfaces. The Array Control Computer (ACC) acts as a master to the correlator control computer issuing commands to the latter. The Data Collector Computer accepts well-defined spectral data sets, applies further processing (flagging bad data, etc.), writes these processed results to a distribution format and forwards them to an archive.

The Correlator Control Computer (CCC) is a Motorola MV-2700 PowerPC computer which runs the VxWorks real-time operating system. Its primary function is to translate commands from the ACC via an Ethernet connection into a format understood by the ALMA correlator using the CAN protocol. The CCC serves as a CAN master to the slave nodes in the correlator hardware. The kinds of commands that the ACC issues include correlator configuration commands, monitor requests and correlator integration control, i.e., starting and stopping of integrations.

The Correlator Data Processor computer (CDP) accepts raw lags from the ALMA correlator hardware via a 32-bit parallel bus (Front Panel Data Port - FPDP) from the LTAs, converts them to spectral results and transports them to the Data Collector computer via a high-speed link. The CDP must accept lags and perform FFTs at extremely high rates. It has been estimated that the data transfer rates can range from 0.5 MB/second to approximately 1GB/s and that processing rates can range from 25 MFLOPS (million floating point instructions per second) to around 20 GFLOPS.

Currently a straw man design of a Beowulf (see <http://www.beowulf.org/> for more information) cluster includes the following items:

- Rack-mounted computer(s) to support 16 FPDPs interface directly to each compute node.
- 16 Dual processor Pentium 4 (or equivalent) nodes with 66 MHz PCI bus
- Gigabit Fiber and router(s), e.g., MyraNet, Giganet to interconnect each node of the Beowulf network in a multi-drop configuration.
- High-speed network interface to the Data Collector computer in order to support the burst data rates from the correlator to the Data Collector computer of 60 MB/sec.
- One or more dedicated Beowulf nodes that act as bridges between the Beowulf network and the ACC-CCC network.

There exists a link between the CCC and the CDP computer systems that will most likely be Ethernet. This provides a communication link between the ACC and the CDP allowing the CCC to send configuration or timing information to the CDP.

12.5.3.3 Correlator Computer Software

The CCC software will be written in C++ as a multi-tasking application utilizing the VxWorks RTOS as there are many time-critical functions in controlling the ALMA correlator.

The CCC software will have the following properties:

- It will utilize the ALMA Common Software (ACS) libraries for basic software services that include peer-to-peer communications, logging of errors and messages, time services, events and alarm systems all of which incorporate the CORBA architecture.
- It will be aware of the 48 ms array time Timing Events in order to synchronize its operation with other hardware and software components in the array.
- It will be capable of controlling the ALMA correlator on 16-ms time boundaries that are the basic time ticks of the correlator hardware.
- It will be able to support the ALMA correlator's specification to handle up to 16 sub-arrays each having a specific set of antennas, bandwidths, correlation modes and dump times – see ALMA Memo 294, *The ALMA Correlator Long Term Accumulator* for details.
- It will be multi-threaded allowing high priority, time critical operations to proceed without interruption from lower priority, non-critical operations to proceed without interruption.
- It will utilize the CAN protocol to communicate commands to and to retrieve monitor data from the ALMA correlator.
- It will follow the general characteristics of a “Device Controller” which is a standard ALMA control computing architecture defining how computer-controlled devices operate.

12.5.3.4 Data Processor Computer Software

The CDP software will be written in C++ for Linux. The software architecture will take advantage of the parallelization of the hardware. Each “data spigot” of the LTA will be connected to a Beowulf compute node to handle set of lags for a given baseline. FFTs will then be run on these lags and sent to the Data Collector computer.

The CDP software will have the following properties:

- It will be able to support all of the correlator modes available to one quadrant of the ALMA baseline correlator.
- It will be capable of supporting the minimum correlator dump times of 16 ms for cross correlations, 16 1-ms auto correlations every 16 ms.
- It will be capable of delivering an output data rate of 6MB/s with a burst rate of 60 MB/sec.
- It will use the FFTW library from MIT (see <http://www.fftw.org> for details) to compute the FFTs. This is an extremely fast FFT engine especially if there are many FFT computations done on similar data sets. Also there is a version of FFTW which can be run on a parallel computer architecture using MPI, although at this time we do not think that this feature will be necessary.
- It will allow the optional processing capabilities of:
 - Van Vleck correction
 - Hanning Windowing
 - Spectral Decimation
 - Spectral Averaging
 - Apply coarse and fine geometric delays for cross-correlations
 - Sum all channels together to provide a single value.
- It will utilize the standard Message Passing Interface (MPI) – see <http://www.mpi-forum.org/> for more information – utilized by many Beowulf systems. It is envisioned that there will be little peer-to-peer communication as each compute node will have all of the lags for a given baseline and will not need to transmit lags between compute nodes. It is likely that the master node will need to communicate information to the compute nodes including configuration and status information and possibly timing information.
- Each compute node will have a small real-time kernel which will handle the high-speed, time-critical task of extracting lags from the ALMA correlator via the FPDPs and distribute the lags to the data processing portion of compute nodes. This separation of the time-critical component of data collection from the processing-intensive portion simplifies the overall design and improves its reliability.

12.6 Telescope Calibration

R. Lucas (IRAM)

Last Changed: 2000-09-28

This section outlines the software needs of the operations to be performed in order to determine the calibration parameters that will be needed to successfully execute and reduce standard astronomical observations. These operations will be under the responsibility of the operators and staff astronomers. They may require all the antennas, but in many cases, only a sub array of antennas will be used. Most will actually be performed by observing standard astronomical sources, so many/most of the general science requirements apply; in particular the data will be processed through the pipeline, and the results fed back to the observing processes; the raw data will be archived as science data. These calibration operations will be implemented in the same way as the standard observing modes. A complete list of those operations cannot be made at this time, since the general calibration plan will be subject to evolution during the next years and even the full life of the array.

Among those operations we may cite:

- **Pointing calibration sessions:**
A set of pointing calibrations are performed on several pointing calibrators all across the sky. This needs to be done after some antennas have been moved, but also for periodical, systematic checks. Occasionally one should measure the relative pointing of the different frequency receivers. Pipeline data reduction will be the same as for the standard pointing measurements during project observations; then a least square determination of the pointing model parameters is performed for each antenna involved.
- **Baseline calibration sessions:**
A set of cross-correlation scans on several calibrators all across the sky are performed, for instance some antennas have been moved (but some antennas which have not been moved have to be included in the sub-array). The data reduction is performed by a least square fit of baseline offsets to the observed phases.
- **Delays calibrations:**
They are needed for most interferometric observations. One observes a strong point source calibrator, in the observable sky, for a very short time to measure the relative delays to the antennas by fitting a straight line to the frequency dependence of the observed phases.
- **Beam shape calibration**
This involves holography measurements on cosmic sources, to monitor the beam shape and the focusing of the antennas, for instance as a function of elevation.

A later revision to this section may also include the processing of array-wide atmospheric transparency monitoring (FTS?).

12.7 Post-Processing Software

B. Glendenning (NRAO)

Last Changed: 2000-09-28

The AIPS++ package (<http://www.aips2.nrao.edu>) will be developed as required to cope with general ALMA data processing needs. The ALMA data products will be written in a FITS-based format so that other packages may be used for special-processing or user preference. This section will be expanded in a later revision.

12.8 Common Software

G. Chiozzi (ESO)

Last Changed: 2001-01-03

12.8.1 Overview

The ALMA Common Software (ACS) is located in between the ALMA application software and other basic commercial or open source software on top of the operating systems. It provides basic software services common to the various applications (like antenna control, correlator software, data pipelining).

ACS is designed to offer a clear path for the implementation of applications, with the goal of obtaining implicit conformity to design standards. In a distributed environment like the one of ALMA, the application software will then become more uniform and therefore more maintainable.

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.

It is intended to develop ACS incrementally via periodic releases. Automatic regression tests are planned in order to achieve that ACS becomes a reliable and robust platform.

More details on ACS Architecture[6] and current status of implementation are available from the ACS Home Page[1].

12.8.2 Technologies

The choice of the technologies used in the ALMA Common Software, and as a consequence on the whole ALMA Software is based on the following initial and explicit decisions:

- Select state of the art but consolidated and widely accepted technologies.
- Adopt an Object Oriented architecture
- Share software rather than re-invent it

The ALMA Common Software is designed based on the experience of the ALMA partners in their previous projects, but is implemented using the technologies and architectural concepts that have found wider industrial acceptance in the last few years.

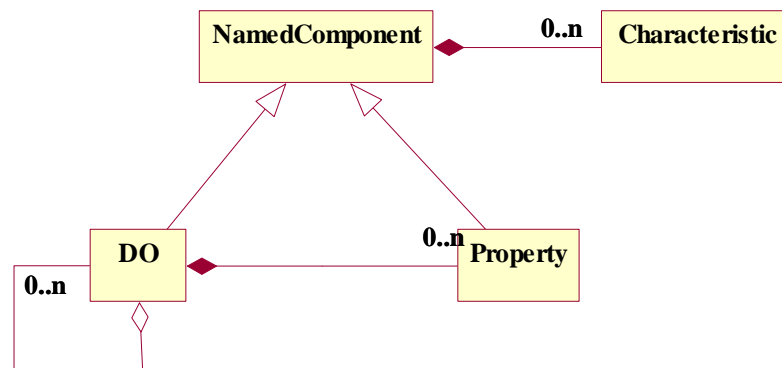
At the very core is the decision to use CORBA[2][3]. The reasons for using CORBA are in short: Object Orientation, support for distributed systems, platform independence, it is a communication standard, it provides a variety of services.

The ALMA software will have to be as much as possible independent from the operating system and will actually run on multiple platforms (Linux and other flavors of UNIX and VxWorks). We have therefore decided to select the Adaptive Communication Environment (ACE)[4] as the basic multi-platform software. This package provides portable operating system interface services and implements a wide set of classes specifically designed for the implementation of distributed real-time systems.

ACE is also the core of The ACE ORB (TAO)[5], a high performance real-time CORBA implementation.

The Object Model for ACS is based on the concept of Distributed Object[6], that identifies three entities:

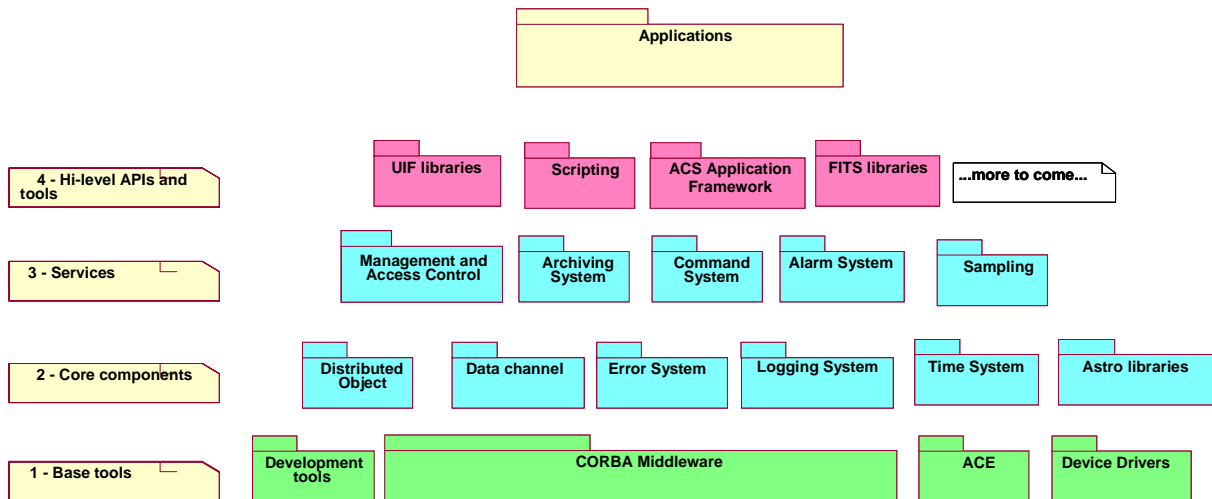
1. **Distributed Object** - Instances of classes identified at design level in the ALMA system, are implemented as Distributed Objects. In particular, at control system level, Distributed Object is the base class used for the representation of any physical (a temperature sensor, a motor) or logical device in the control system.
2. **Property** - Each Distributed Object is characterized by dynamic values, called Properties, that are monitored and controlled (status, position, velocity, electric current).
3. **Characteristic** - Static data associated with a Distributed Object or with a Property, including meta-data such as description, type and dimensions, and other data such as *units*, *range* or *resolution* are called Characteristics.



The choice of CORBA for the implementation of Distributed Objects and of all services that are part of ACS 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.

12.8.3 Services

The following UML Package Diagram[6] shows the main services provided by ACS, represented by the packages that implement them.



Packages have been grouped in layers in order to limit as much as possible the relations between the layers (not shown in the diagram): package are allowed to use services provided by other packages on the lower layers and on the same layer, but not on higher layers.

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.
- **Device Drivers**
Device divers for the control standardized and commonly used hardware and electronic components.

2 - Core components

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

- **Distributed Object**
This package provides the IDL interfaces for the Distributed Object class, for the Property classes to access values of all basic data types (like integer and floating point numbers) and for Characteristics. It provides also the basic server-side CORBA implementation for these same classes. Server-side applications can use directly these classes or implement sub-classes for specific I/O devices. Client-side applications use the IDL interfaces to access all Distributed Objects, not being concerned by the specific implementation details of the server.
- **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 goal for these 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.

- **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.

12.8.4 References

- [1] **ACS home page** (<http://www.eso.org/~gchiozzi/AlmaAcs/index.html>)
- [2] **CORBA - Object Management Group home page** (<http://www.omg.org/>)
- [3] **Advanced CORBA Programming with C++**, M.Henning S.Vinoski, Addison-Wesley, 1999
- [4] **ACE, Adaptive Communication Environment** (<http://www.cs.wustl.edu/~schmidt/ACE.html>)
- [5] **TAO, the ACE ORB** (<http://www.cs.wustl.edu/~schmidt/TAO.html>)
- [6] **ALMA Common Software Architecture**, G.Chiozzi, B.Gustafsson, B.Jeram, 2001
(<http://www.alma.nrao.edu/development/computing/docs/joint/0016/ACSArchitecture-2.0.pdf>)

12.9 Software Practices

M. Zamparelli (ESO)

Last Changed: 2002-01-04

The following set of Software Engineering activities for ALMA has been accepted and documented:

- a. specification of the software process, iterative and incremental, with well defined milestones and deliverables, in accordance with standard industry

practice. The process is requirements based, and accounts for repeated refinements of implemented functionality. The use of CASE tools is instrumental in determining an architecture of the system capable of withstanding the inevitable requirement changes.

- b. specification of documentation planning, formats, review procedures and configuration management. A layout for Microsoft Word document was made available. Documents are considered applicable when passing a formal review meeting, whose minutes are collected and archived.
- c. specification of the software development environment to be used, including tools, operating systems (real time and not), coding templates and makefiles. Such an environment guarantees the separation between stable released code and integration areas for development. Frequently Asked Questions (FAQ) are being collected to facilitate the introduction of new staff.
- d. specification of integration layers for software quality assurance: sufficiency and conformity of documentation, adoption of mandatory coding standards, stability of source code with respect to daily build (compilation and linking), sufficiency and depth of testing coverage, dynamic memory behaviour and finally test results. Integration groups will be charged with testing interfaces between subsystems by providing tests which exercise the system from the front to the back end.
- e. specification of configuration management for source code. It will allow to baseline a configuration for release. It features an easy to use interface and is based upon a minimal set of key principles: separation of responsibility areas into "modules", identification of the items, strict locking mechanism, traceability of changes, accessibility of previous configuration for any item. A minimal set of rules shall be enforced to make sure developers are well behaved when using this system.
- f. specification of a fault reporting/change management system, to be used internally by developers during development during construction, and by users during operation phase. It allows to keep track of faults or change requests by sorting them into subpackages, assigning them to people to be implemented for a given date. The system has an easy to use interface and will be used for minor documentation changes as well.

Work on all these areas has started and is fairly complete in some of them. Progress will depend upon the choice of specific third party software tools which will enable the automation of several tasks involved in software engineering.