



Atacama Large Millimeter Array

Control Subsystem – Design Document

ALMA-70.35.00.00-004-D-PLA

Version: D

Status: Draft

2004-06-17

Prepared By:		
Name(s) and Signature(s)	Organization	Date
Ken Ramey Ralph Marson Alan Farris	NRAO	2004-06-17
Approved By:		
Name and Signature	Organization	Date
Released By:		
Name and Signature	Organization	Date



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 2 of 87

Change Record

Version	Date	Affected Section(s)	Change Request #	Reason/Initiation/Remarks
A	2002-11-20	All		First version for IDR
B	2003-02-21			Updated with comments from IDR
C	2004-05-21			Update in preparation for CDR2
D	2004-06-17	All		Added the internal reviewer's comments. This is the version presented for CDR2.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 3 of 87

Table of Contents

1	DESCRIPTION	6
1.1	Purpose	6
1.2	Scope	7
2	RELATED DOCUMENTS AND DRAWINGS.....	7
2.1	References	7
2.2	Abbreviations and Acronyms	8
2.3	Glossary	8
2.4	Related Interface Control Drawings	8
3	REQUIREMENTS	8
3.1	SSR Requirements	8
3.2	SSR Requirements	8
3.2.1	From section 3.1 – General Requirements.....	9
3.2.2	From Section 3.2 – Real time Software	11
3.2.3	From Section 3.4 – Dynamic scheduling.....	16
3.2.4	From Section 3.5 – Operator Interface	17
3.2.5	From Section 3.6 – Pipeline data processing.....	18
3.3	Engineering requirements.....	18
3.4	Operational requirements	20
3.4.1	Safety	20
3.4.2	Generic.....	20
3.4.3	Design	20
3.4.4	Commissioning	20
3.4.5	Specific comments on individual items	21
3.5	ALMA System Requirements and Constraints	22
3.6	Discussion of Requirements	22
4	ARCHITECTURE	22
4.1	Overview	24
4.2	Package Descriptions.....	27
4.2.1	The ALMA Device Package.....	27
4.2.2	The Monitor Package.....	30
4.2.3	The Telescope Operator Package	33
4.2.4	The Master Controller Package	33
4.2.5	The Array Controller Package	35
4.2.6	The Manual Controller Package	35
4.2.7	The State Model Package	36
4.2.8	Data Capture Package.....	36
4.3	Package Interactions.....	38
4.4	Package Integration	40
4.5	Physical Architecture.....	41
4.5.1	Introduction.....	41



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 4 of 87

4.5.2	Single-Point Failures.....	45
4.5.3	Electrical Interfaces	47
4.5.4	AMB Interface	48
5	EXTERNAL INTERFACES.....	50
6	DEVELOPMENT	51
6.1	Control Command Language.....	51
6.1.1	Language Details	52
6.1.2	Language Structure	52
6.1.3	Parallelism and Synchronization	53
6.1.4	Language Objects	54
6.1.5	Method List by Object	55
6.2	Device Hierarchy	64
6.2.1	Control System	64
6.2.2	Array Hierarchy	65
6.2.3	Antenna Device Hierarchy.....	66
6.2.4	Receiver Device Hierarchy	67
6.2.5	Front End Devices	68
6.2.6	Second Local Oscillator.....	69
6.2.7	DTS Transmitter Hierarchy	69
6.2.8	IF Monitor and Control.....	70
6.3	Physical Device Management	71
6.4	Delay and Fringe Tracking.....	71
6.5	Real-Time Linux.....	72
6.6	Monitor Point Organization.....	75
6.7	Time Synchronization.....	77
6.8	Simulation.....	78
7	LESSONS FROM TICS.....	79
7.1	Software Infrastructure	79
7.2	Software Design	80
7.3	Experiences.....	81
7.3.1	Interface Control Documents.....	81
7.3.2	Python for Rapid Prototyping	81
7.3.3	Check the Cabling First	81
7.3.4	Provide Software Bypass Mechanisms.....	82
7.3.5	Provide a Remote Reset.....	82
7.3.6	Enforce Hardware Interface Standards	82
7.3.7	Plan for a Significant Support Load	83
7.3.8	Real Time Operating System.....	83
8	APPENDIX	85
8.1	CCL Program Examples.....	85
8.1.1	Example 1 – Busy/Wait Synchronization.....	85
8.1.2	Example 2 – Threads and Synchronization	86



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 5 of 87



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 6 of 87

1 Description

1.1 Purpose

The purpose of the ALMA control system is to monitor and control the ALMA telescope and to execute scheduling blocks from approved scientific observing projects. To monitor the ALMA telescope means to keep track of its state and report anomalies. To control the ALMA telescope means to start and stop it and to command it to do useful work including observations, calibrations and diagnostics. Monitoring and controlling the telescope are continuous activities, regardless of whether the telescope is engaged in scientific observing.

The ALMA telescope is a complex collection of devices: 64 antennas each containing receivers, samplers and numerous associated devices; a correlator with 4 quadrants; time and local oscillator generation and distribution devices; meteorology and water vapor radiometry devices; and many other devices, as well as a data collection and communication infrastructure.

One of the key concepts of the ALMA telescope is that of a sub-array. A sub-array is software partitioning of antennas and ancillary devices so that different parts of the array can function independently. Strictly speaking antennas are considered as belonging to the same sub-array if it makes sense to correlate data from them. In general the ALMA telescope can be operated as a single array of all antennas or as multiple sub-arrays operating independently and current designs allow for up to 64 sub-arrays, each one containing one antenna¹. The control subsystem should not constrain operations involving multiple sub-arrays.

Approved scientific observing projects are defined in terms of scheduling blocks that are executed by the control subsystem. These scheduling blocks are themselves complex and the control system must adequately deal with this complexity. But not all activity associated with the ALMA telescope is in terms of approved scientific observing projects. Maintenance, engineering testing and debugging, and special testing of other types must be supported as well. These sorts of activities are supported by the concept of a “manual” mode, one in which commands are entered to a sub-array and executed directly. The control system can operate independently of all other software subsystems when in manual mode although operations that involve other subsystems, like data archiving, will be difficult. In addition, a given antenna or antennas may be completely offline and unavailable to the control system. The control system must manage all of these differing modes of operation.

¹ Hardware constraints will limit what can be done with 64 sub-arrays.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 7 of 87

1.2 Scope

This document details the design of the ALMA Control subsystem software system. Requirements are presented, along with a brief discussion of each. The following sections of the document provide greater detail of each component of the Control software.

The bulk of the material of interest is in section 6 - Development. That section, along with section 4 - Architecture provides a comprehensive picture of the overall structure of the Control subsystem and its interaction with other subsystems.

2 Related Documents and Drawings

2.1 References

- [1] ALMA-75.35.00.00-002-A-STD (ex. ALMA-US Computing Memo #7) ALMA Monitor and Control Bus Interface Specifications, Version C, 2001-September-07.
- [2] ALMA Software Glossary, Draft Version, 2003-May-21
- [3] ALMA-75.35.00.00-001-A-STD (ex. ALMA08002.0006) Generic Monitor and Control Points, Draft, 2000-November-01
- [4] ALMA-70.35.10.02-001-A-MAN. ALMA Computing memo #12, “ALMA Monitor & Control Bus, AMBSI2 Standard Interface, Design Description”, 2001-May-03.
- [5] [Computing Plan] ALMA-SW-Draft, ALMA Computing Plan for Phase 2, G.Raffi, B.Glendenning
- [6] [Architecture] Software Architecture & High-Level Design, J.Schwarz et al. (draft)
- [7] COMP-70.65.00.00-003-C-DSN. Offline Subsystem Design, Draft, 2004-05-28, J. McMullin, K. Golap.
- [8] ALMA-70.00.00.00-004-A-SPE. ALMA Export Data Format, 2004-04-14, F. Viallefond, R. Lucas.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 8 of 87

2.2 Abbreviations and Acronyms

See [2].

2.3 Glossary

See [2].

2.4 Related Interface Control Drawings

Not Applicable

3 Requirements

3.1 SSR Requirements

Until CDR2 the only source of requirements for the control software was the ALMA software science requirements (). However it is apparent that additional requirements are necessary and our initial designs implicitly took some of these into account (such as the need for a more extensive monitoring of equipment). These additional requirements need to become more formalized and some attempt to do this has been made. The additional requirements fall into the following categories

1. Engineering Requirements

These are requirements imposed on the software by other integrated product teams to support the development, testing, commissioning and maintenance of their equipment.


2. Operations Requirements

These are requirements imposed on the control software by the operations staff, to support the efficient operation of the telescope.

There is some overlap in the scope of all these requirements and in general the requirements will be stated in the section which is most relevant.

3.2 SSR Requirements

The scientific requirements account for the bulk of the requirements for the control subsystem. This design is based on the current version ALMA-70.10.00.00-002-I-SPE “ALMA Software Science Requirements and Use Cases” dated 2003-10-03.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 9 of 87</p>
--	---	---

The requirements listed below are the ones relevant to control software. Below each requirement there is a brief commentary stating how the requirement will be addressed in the control subsystem design.

3.2.1 From section 3.1 – General Requirements

1.0-R1 The ALMA software shall offer an easy to use interface to any user and should not assume detailed knowledge of millimeter astronomy and of the ALMA hardware.

This is such an open ended requirement that complying with it will be subject to considerable interpretation. One specific consequence is that the control software will not assume that the operator is an astronomer or an engineer and hence avoid the use of jargon and acronyms.

1.0-R2 The ALMA software shall provide simple ways for the staff or expert astronomers to refine observing modes and develop new ones.

This will be done by having the observing modes developed in manual mode and defined in terms of scripts in the Python language. In manual mode the user issues commands at a command line prompt and awaits the response of the telescope. Once a suitable set of commands, that define a mode, is developed it will be the responsibility of the control subsystem to store the script in a library and provide mechanisms for the parameters to be extracted from the scheduling block (this operation will not be automated). Manual mode will be a superset of the capabilities that are available in automatic mode (where the control system receives scheduling commands from the scheduler subsystem).

1.0-R3 The expert user/developer shall be able to send direct orders to the hardware and to basic quasi real-time software through simple scripts in a Command Control Language. These scripts, once fully developed and tested, will evolve into standard observing modes.

The primary interface in manual mode will be through a rich set of high and low level commands that are available through the Python interpreter. These scripts will become standard observing modes that can be called directly from within the Python interpreter (when in manual mode) and are accessible from within the observation preparation subsystem.

1.0-R4 The general user shall be offered fully supported, standard observing modes to achieve the project goals, expressed in terms of science parameters rather than technical quantities. Observing modes shall allow automatic fine tuning of observing parameters to adapt to small changes in observing conditions.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 10 of 87

The control system will convert science parameters to instrumental parameters when it executes the relevant commands. In general this conversion will be done as late as possible in order to maintain the scientific intent and provide the most accurate conversion (and some generic conversions may be done in the observing tool). The control system will receive, in near real time, feedback from the calibration system and the standard observing modes will contain the ability to adjust the observing in response to these quantities.

1.0-R5 All user interaction with the ALMA system shall be through Graphical User Interfaces (GUIs) except for the low level Command Control Language.

Astronomer interaction will be through the observing tool except when in manual mode where the control command language i.e., functions accessible through the Python interpreter, will be the primary interface. The control system will also provide graphical user interfaces for routine operation by telescope operators. Low level access to the hardware by engineers will be through both a command line and a LabView interface (which provides graphical widgets for the display of data).

1.0-R8 Raw data, monitor data, calibration data, and images will be archived; archived data shall be easily accessible to the users.


All monitor data will be sent to the archive by the Monitor Package which will forward this to the Archive subsystem. The control subsystem will provide interfaces for real-time examination of monitor data.

1.0-R10 An Alarm System shall allow hardware and software faults to be uniquely identified and suppress error cascades. Identification of the faults shall be available to the operator and used in operating ALMA. The faults shall be logged and used as input for system maintenance and for the dynamic scheduling.

The Monitor package will analyze alarms from multiple hardware components and attempt to diagnose the underlying cause of the errors. In general the heuristics used will be deduced through operational experience with the array.

1.0-R12 There should be provision to be able to introduce interference mitigation software at a later stage, if needed

This implies that the control software should be maintainable so that extensions to it can be made in the future. We do this by using widely used languages (C++, Python & Java), standards (CORBA) and computing platforms (Linux). This functionality is obtained through the use of ACS. It also implies the control system is well documented.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 11 of 87</p>
--	---	--

3.2.2 From Section 3.2 – Real time Software

2.1-R1 A Technical Interface shall be available for engineers for debugging and maintenance purposes

A low level interface is mentioned in this design. It is command line (Python) based and the commands are available only in manual mode. This interface will provide access to the hardware with minimal translation & interpretation. In addition a LabView interface will be provided. The control subsystem will not provide the LabView clients.

2.1-R2 In Manual Mode a subset of the antennas shall be directly controlled through a Control Command Language.

The Array Controller package described later in this document will contain a command line interface that allows full control of a sub-array

2.1-R4 In Dynamically Scheduled Mode the array shall execute the highest priority observations (scheduling blocks) selected by the dynamic scheduler. This shall be the default mode of operation.

The observe function in the ArrayController interface executes the scheduling blocks specified by the scheduler subsystem.

2.1-R5 The same Observing Modes shall be available in Interactive Mode and Dynamically Scheduled Mode. They shall include: ... *a detailed list of 19 modes ...*


All these modes will be available through functions in the Command package and available in Dynamic, Interactive and Manual mode.

2.1-R7 The antennas shall be divided into one or more sub-arrays, operated simultaneously and independently, each sub-array being in any of the above modes.

However there should be only one programme in the dynamically scheduled mode at any time (see 2.1R11 below).

The control system allows up to 64 sub-arrays and places no restrictions what mode each sub-array is in. However the ALMA hardware does place restrictions (see below).

2.1-R8 The allocation of antennas to sub-arrays/sessions shall take into account the hardware constraints imposed by local oscillator control (up to 4 different simultaneous LO setups), and by the correlator (up to 16 different correlator sub-arrays).

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 12 of 87
--	---	---

The control system will not create a sub-array unless the hardware can support it. Note that the hardware can support 64 sub-arrays in some circumstances as there are 16 correlator sub-arrays per quadrant (and there are four quadrants).

2.1-R10 ALMA software shall support a phased array mode for VLBI, using all or a sub-array of antennas.

This is a supported mode defined in requirement 2.1-R5 (above).

2.1-R11 As a baseline plan there shall be only one dynamically scheduled research programme at a time, using all available antennas while some antennas have been taken out by the staff as one or several interactively controlled sub-arrays, for calibration or research. Remaining antennas shall be made available to filler science programs, either through an automatic scheduling functionality, or by manual action of the operator. The filler programs must be able to release these antennas on short notice as soon as they are needed by the main (dynamically scheduled) research programme.


The control system places no restrictions on how many sub-arrays are dynamically scheduled. It provides functions that allow observations in a sub-array to be immediately terminated (see the stop function in the ArrayController interface). However the scheduler cannot stop observations, and de-allocate sub-arrays that are in manual mode. This needs to be done by the operator.

2.2-R1 The minimum amount of observing activity that can be obtained by issuing a single observing command (observation) is described by an observation descriptor (see appendix). These parameters fully describe the data taking activity during that observation, including telescope motion and switching schemes.

In normal operation the control system will assemble the information described in an observation descriptor from a number of sources, included the scheduling block (astronomer intent), the calibration subsystem and the monitor package (current conditions). At the moment there are no plans to have an explicit observation descriptor.

2.2-R2 The Control Command Language shall include commands to actually control the hardware (antennas, LO's, correlators) for data taking, according to the observation descriptor.

The command control language will contain functions that allow the astronomical parameters to the observation descriptor to be specified. Instrumental parameters, like the positions of the antennas, will normally be filled in automatically by the control system and changing these parameters will be difficult (but not impossible). Data taking

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 13 of 87
--	---	---

will not be possible if the control subsystem is not connected to the archive (see section 2.2 below).

2.2-R3 During an observation each antenna shall move following a pattern relative to the source direction. That pattern shall be independently defined for each antenna and shall be either:

- a fixed position
- an arc of circle on the celestial sphere (defined by a starting point, the center point, and the angular velocity)
- a general curve interpolated between a set of points on the celestial sphere, and the corresponding times

The control software developed for testing the ALMA prototype antenna already supports the first two of these modes and the ALMA control software will be based on this software.

2.2-R4 During an observation other switching schemes shall include:

- subreflector nutation (TBD),
- LO1 frequency switching,
- load switching for calibration (TBD)

The control software will support these modes if the appropriate hardware is available.

2.2-R5 The Control Command Language shall include commands to convert astronomer's input observing parameters into observation descriptor parameters when this can only be done at the time of the observation. This includes:

- coordinate conversion to the antenna system,
- LO and IF filter setting according to frequency and Doppler tracking parameters (in the current baseline system design).

In general this will be done automatically by functions in the control system. For example an astronomer can specify an observation frequency in terms of a rest frequency, a heliocentric frame and a velocity or they can specify it in terms of a topocentric frequency.

2.2-R6 The Control Command Language shall include commands to access pipeline results or current environmental parameters in order to tune up observing parameters (integration times, loop cycles) in quasi real time, according to pre-defined formulas, when so requested.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 14 of 87

The control system obtains results from the calibration subsystem and the monitor package contains a “state model” package which makes this information along with weather and data quality information available to the control command language. The control subsystem does not get feedback from the quick-look or science data reduction pipelines.

2.2-R7 The Control Command Language shall include commands to setup the pipeline for data reduction.

This facility is provided through the addition of an imaging script in a scheduling block and is not included in the control command language.

2.2-R8 Features in the language built-in functionalities should include:

- macros for abbreviation of frequently typed sequences
- procedures to which parameters may be passed
- definition of variables and arrays, with numeric or character content
- evaluation of expressions, including built-in functions
- conditional execution facilities
- loops
- error recovery facilities including a time out
- interruption facility in procedure execution

All these features are available in the Python language.

2.3-R1 Data taking shall be blanked (i.e. data from a correlator dump will not be integrated) on an antenna-based basis whenever:

- tracking errors are in excess of an observer specified value
- any LO in the data path is out of lock (base band based)
- the subreflector is out of position tolerance
- the receiver mechanical calibration system (e.g. vane) is out of position tolerance
- interference is detected by specific equipment

The minimum amount of blanked data shall be:

- an integer multiple of the 16ms correlator dump time.
- smaller than (TBD) 5 percent fraction of the integration time (if this fraction is larger than 16ms). This means that for long integration times (tens of seconds) being able to blank a single correlator dump is not necessary.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 15 of 87

The correlator subsystem contains facilities for the control system to send blanking information to it prior to the dumps being averaged. As the fundamental loop cycle time in the control system is 48ms blanking on shorter time scales i.e., 16ms, will normally not be possible. This change was raised discussed before the SSR at IDR and agreed upon. As the blanking information needs to go from the antenna to the correlator via non-deterministic communications links (Ethernet) it is not possible to guarantee that this information can be made available to the correlator in a timely fashion. To cope with this the correlator will blank data if it does not receive blanking commands from the control subsystem, and benchmarking will be done to establish that this happens less than 0.1% of the time.

2.3-R2 It shall be possible to flag integration periods when the data is (or may be) affected in a way that could lead to wrong science. The flagging information shall identify, as Boolean information, the origin of the malfunction. Conditions which shall cause flagging include at least:

- Antenna-based flags, integration based:
- Last WVR calibration failed
- Current WVR hardware defect
- WVR currently degrades data (based on calibrator amplitude or phase)
- Last pointing calibration failed (or no pointing calibration done)
- Last temperature scale calibration failed (all data was blanked)
- Temperature scale calibration system hardware defect
- Last temperature scale calibration failed, Tsys is currently estimated and not measured (by baseband)
- Shadowing: the antenna aperture is shadowed for any reason (the amount of shadowing shall be kept along with data, in addition to the flagging information).
- Total power out of range (by baseband)
- Integration partly blanked (including blanking condition identification): as a warning.
- Integration totally blanked (including blanking condition identification): obviously quite severe.
- Bad data (by baseband, reserved for use by pipeline)
- Interference is detected by specific equipment during integration.
- Baseline-based flags, observation based:
- Correlator malfunction (baseband based) e.g.: Correlator chip failed last self-test;
- Excessive closure error last calibrator observation, ...

Parameters ranges leading to flagging shall be settable. Flagged integration periods can later be optionally used or discarded for further data reduction.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 16 of 87

The control subsystem will provide flagging information as part of the auxiliary data it provides to the data capture component. This will include the first ten categories listed above and the interference based condition. Other subsystems (correlator and telescope calibration) will be report separately to data capture the flags for conditions they can detect.

2.3-R8 The shorter integrations allowed by the hardware shall be supported (16ms for correlation, 1ms for autocorrelation only, 2ms for the continuum detectors).

The correlator integration times are addressed by the correlator subsystem. Continuum detectors are handled by the control system and this will provide data at a 2ms data rate. However the control system will sample these detectors at a 48ms rate (and read 24 values).

2.3-R12 When a band is equipped with a double sideband receiver, it shall be possible to process and store data from both sidebands using software sideband separation or to store data from a single sideband using sideband suppression by local oscillator offset.

This is a requirement on both the control subsystem and the correlator subsystem. Sideband separation is accomplished by switching the phase of the local oscillators (LOs) by 90 degrees and this switching is done using either the first or second LO's. The removal of this switching, and the accumulation of both sidebands is done in the correlator. Sideband suppression is done by offsetting the frequencies by equal amounts in the first and second LOs and this is done solely by the control subsystem.


3.2.3 From Section 3.4 – Dynamic scheduling

4.0-R4 Fully interactive observing shall be available (as a special case), using the whole array or a sub-array (when justified).

The control system makes no distinctions on whether you are using the whole array or a sub-array (the whole array is just a big sub-array). In interactive mode observing is done by creating scheduling blocks using the observing tool, sending them to the scheduler which schedules them immediately and passes them to the control subsystem.

4.0-R5 A manual mode shall be available to the staff (as a special case), for testing new observing procedures, by sending commands directly to the observing system, using the whole array or a sub-array.

In manual mode you are communicating directly with the control subsystem (compared with interactive mode). Again there is no software distinction between observing with the full array or any number of sub-arrays.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 17 of 87</p>
--	---	--

4.0-R10 At all times two receiver bands are active (used in turn for target observations and calibration), and a third one is kept on standby, ready to become an active band when the next project is observed.

Consequently, during the execution of a SB, scheduling shall periodically (e.g. every 15 min) select the SB that would be selected if the currently executed SB would finish at this time, and change the receiver band on standby accordingly, if needed (whether this evaluation may use extrapolation of observed environmental parameters should be considered during design). When the SB actually finishes, only the SBs making use of receiver bands either active or on standby are to be considered.

The control subsystem will provide a mechanism for the dynamic scheduler to query the state of the receiver bands and an interface that allows the dynamic scheduler to change the state of receivers that are not being used by the current scheduling block.

3.2.4 From Section 3.5 – Operator Interface

5.0-R1 The Operator Interface shall include a basic “current array status” display, including:

- technical information such as current pointing position(s) in a variety of coordinate systems,
- receiver status,
- correlator configuration,
- and live information on data acquisition (current observation, scan, scheduling block, project, etc.)


This information will be provided by GUI's in the control system. However it will get the correlator configuration GUI from the correlator subsystem.

5.0-R2 The Operator Interface shall include a weather display indicating the current site conditions, with warnings issued whenever the current conditions forbid outside activities by the local staff, or when hardware (antennas) must be safely docked.

This will be part of the Monitor package.

5.0-R3 The Operator interface shall include a video display from cameras at strategic locations on the site.

The image from these cameras will only be downloaded upon user request, but the GUI will contain an update parameter.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 18 of 87</p>
--	---	--

5.0-R4 The Operator interface shall have a hierarchical structure with:

- a top level monitor to show integrated information, including the current observing plan, the array status, and telescope status.
- a second level to show important monitoring items for the current observations.
- a third level with on-demand monitoring.

This hierarchy is reflected in the design of the control system interfaces. The system GUI will be embedded in the frame created by the Executive system, which provides the top level of the hierarchy.

5.0-R5 The Operator Interface software shall be made available from any location where the Operator may work. It shall be available in a read-only mode from other locations, in order to support remote monitoring of the array system.

The control system interface design allows for this by providing separate interfaces for monitoring and controlling each sub-array. Hence different access control levels will be established for the two interfaces.

3.2.5 From Section 3.6 – Pipeline data processing

6.2-R2 The Pipeline shall be data-driven. All necessary parameters will be specified either by the PI/CoI's at proposal preparation stages (science data) or by the ALMA staff (telescope calibration).

The control subsystem will make available data that is not known at proposal preparation stage, and necessary for the pipelines to operate available in the meta-data blocks. This excludes the output from the correlator which is handled separately.

3.3 Engineering requirements

There are no specific engineering requirements. However some science requirements cover engineering issues, in particular requirements 1.0-R8, 1.0-R10 & 2.1-R1. Because there is no explicit list of engineering requirements we will define our own requirements based on discussion with engineers and our experiences at the ALAM test facility.

ER1 There should be no way for software to damage hardware.

The ICD's between a piece of equipment and ALMA control SW all impose the restriction that control software cannot damage the hardware.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 19 of 87

ER2 All monitor points specified in the appropriate ICD with a suggested sampling rate shall be sampled and archived at a rate near the suggested sampling rate. Monitor points that are labeled as debug will not be automatically sampled. Some monitor points will be sampled and archived only when certain conditions are met (such as at startup or when there is a source change).

This is an extension of the first part of science requirement 1.0-R8. It defines in more detail what monitor points will be archived and at what rate they will be sampled.

ER3 No monitor point will be sampled faster than once every 48 milliseconds.

The 48 millisecond interval is the fundamental timing event that is ubiquitous in the ALMA electronics. There is one exception to this rule and it is the total power detectors. To collect all the data these monitor points will need to be read 24 times per 48 milliseconds and special mechanisms are needed to handle this. No other special mechanisms are planned.


ER4 No individual monitor point will be archived faster than twice every second.

It is possible and in many cases necessary to sample monitor points at rates up to 20.83 times a second. But to limit the amount of data written to the archive this high rate data will be sub-sampled so to meet the above mentioned restrictions. This will be done in the monitor package.

ER5 There will be a low level interface to all hardware on the AMB that will be accessible from LabView. This interface will be accessible from remote locations. Access from outside the AOS and OSF will be read-only.

LabView is commercial software that is widely used on the benches of the electronics groups. It can generate CAN commands and engineers use it to test there equipment. We will provide an interface so that these engineers can use there test fixtures remotely and while there equipment is installed on the telescope. We will look at ways to ensure that that controlling equipment can only be done from the AOS or the OSF. In all cases this debugging cannot be allowed to interfere with normal observing.

This is almost certainly an incomplete list of engineering requirements. Future requirements can only be accommodated if they fit within the existing control software design and if they do not, in total, require more than 0.5 FTE's worth of effort. Larger amounts of work will require additional resources or a de-scoping of the existing requirements.

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 20 of 87
--	---	---

3.4 Operational requirements

The ALMA operations plan is in its late draft stage. From this Computing IPT management has distilled a series of requirements in COMP-70.00.00.00-005-A-SPE “Operations Requirements and Specifications on the Computing IPT” dated 2004-05-17. There are a number of requirements in this document that relate directly to the control subsystem and many of them overlap with the engineering and science requirements listed above. I will group these requirements into a few categories

3.4.1 Safety

Requirements 4.1.1 & 4.1.2 relate to safety and are a duplicate of ER1 (above). A particular point that is mentioned is the need for the antennas to stow if the AOS to OSF link is unavailable. This can be handled by enabling the feature whereby the antennas automatically stow if no command is received after a preset time. At the ATF this feature was turned off as there was a risk that unexpected antenna motion could harm someone on or near an antenna.

3.4.2 Generic

Requirement 4.1.3 is a generic requirement which, like science requirement 1.0-R1, is not easily quantifiable.


3.4.3 Design

Many of the requirements have already been addressed by the control system design. These include requirements 4.1.6 – 4.1.11, 4.1.15 – 4.1.16, 5.1.2, 7.1.9, 8.1.5, 8.1.10 – 8.1.11, 9.2, 9.3.1.

3.4.4 Commissioning

Requirements 8.1.1 & 8.1.2 relate to commissioning and early science activities. Whenever these activities are identical to what would be done in late science there is no impact on control software (except perhaps to adjust the schedule for delivery). Some commissioning activities do impact control software and the most notable of these is optical pointing and holography. Following my reading of the latest draft of the commissioning and science verification plan, and in private discussions with its author Robert Laing, I am assuming that:

1. Holography will not be done with the ATF holography receiver. Instead interferometric holography will be done at the OSF with two ALMA antennas and a ALMA receiver.
2. Optical pointing will be done with the ATF optical telescope. It will be done at the OSF and may be done at the AOS. In both cases the observer will be in a nearby building i.e., optical pointing cannot be done at the AOS with an observer at the OSF.

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 21 of 87
--	---	---

3.4.5 Specific comments on individual items

Here I would like to comment in detail on a few specific requirements

4.1.12. Maintenance time.

The time estimates listed here are reasonable once we get into routine observing. It should be stressed that this requirement does not hold during commissioning or during the early science phases.

7.1.3 Observatory calibration time charging

I think this requirement will impact control as it is the control subsystem who knows how long a calibration observation ends up taking. I'm guessing this mean that an additional parameter will need to be attached to calibrations indicating whether to the time should be charged against the SB or not.

8.1.4 Array operations to be conducted from the OSF as soon as fiber link is established.

This requirement concerns me because it implies that it will be possible to operate the array, from the AOS, with out the fiber link. I do not see how this is possible as there will be no spinning disk at the AOS (requirement 4.1.6). And without a disk the real-time computers cannot boot. To fulfill this requirement we will need to overcome some technical challenges.


8.1.9 There will be, independent of the ABM, & ACC a way to perform an emergency stop on all ALMA antennas.

This mechanism, which is an extension of the conventional emergency stop function, is based around the “utility module” which is effectively a set of switches with a TCP/IP interface. This feature does rely on the Ethernet switches and cabling.

11.6.1 It must be possible to jointly access and display the archived monitor information and correlator visibilities.

This feature is not in the current control software design and adding it to control would take about 0.5 FTE's. I agree that it more sensibly belongs in the data reduction subsystems. This allocation of work is still to be decided.

11.11.1 An inventory control system shall be implemented.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 22 of 87</p>
--	---	--

The control software can help here as it can keep track of the serial numbers on all devices with an AMBSI interface that are connected to the array. This is a small part of the overall inventory system and if and how the control software is to participate is yet to be decided.

11.13.2 Site characteristics data base.

This involves control as well as it's possible that some information that is input to this data base will be determined, dynamically by control. It's certain that this database will be used by control and its priority should be raised.

3.5 ALMA System Requirements and Constraints

The control subsystem needs to be reliable and respond in a predictable fashion to failures by telescope hardware, communications infrastructure and other software subsystems that it normally communicates with. To facilitate this, the control subsystem will be able to operate in a manual mode whereby it does not use services from any other software subsystems. In this mode no science observing can take place. Monitoring and control of all devices in the array will still be possible however no monitor points will be archived.


It is important to make the distinction between manual mode of the entire control subsystem and manual mode of a sub-array. The former is a superset of the latter and when the control subsystem is in manual mode it will still be possible to create sub-arrays, but they will always be in manual mode.

3.6 Discussion of Requirements

Throughout this section of I have discussed individual requirements in the section where they are stated. Here I will just have a few overarching comments. Since the first version of this design (presented at PDR) there has been some expansion of the requirements. To date none of the new requirements has seriously impacted the design. Many of the new requirements are already considered in the existing design and quite a number require only a modest increase in effort. This may not always be the case and requirements which impose a significant change in the design or the overall architecture should be supported by a strong science case.

4 Architecture

An overview of the ALMA software system is given in <http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/Architecture>

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 23 of 87</p>
--	---	--

Shown in Figure 1 is how the control subsystem fits in this context. In order to understand the role of the control system and its relationship to the other ALMA subsystems we will briefly state the main functional paths within this system.

The executive starts the control system by telling it to initialize itself, which means that it is responsible for initializing and configuring all hardware devices within the system. At this point the software discovers what hardware is available for it to manage (using a broadcast protocol and a database of possible hardware). The software subsystem also determines whether the correlator, calibration and archive subsystems are available at this point.

Normal scientific observations are initiated when the executive tells the scheduling system to begin scheduling work. Work gets to the control system in the form of scheduling blocks. These are stored in the archive as parts of observing projects, having been created by an observer using the Observation Preparation tool. The scheduling subsystem determines how to split up the array into sub-arrays and tells each sub-array which scheduling block to observe. The control subsystem executes the scheduling block commanding the antennas, correlator, and other hardware. The result is that raw data - from the correlator - and meta-data - from the control subsystem - are made available to the calibration and quick-look pipelines. This raw-data and meta-data are also stored in the archive. Subsequent parts of the system analyze the data and inform the PI of its availability.

The architecture of the control subsystem reflects the fundamental concepts related to its mission. We will introduce these concepts as motivation for the package structure of the subsystem.

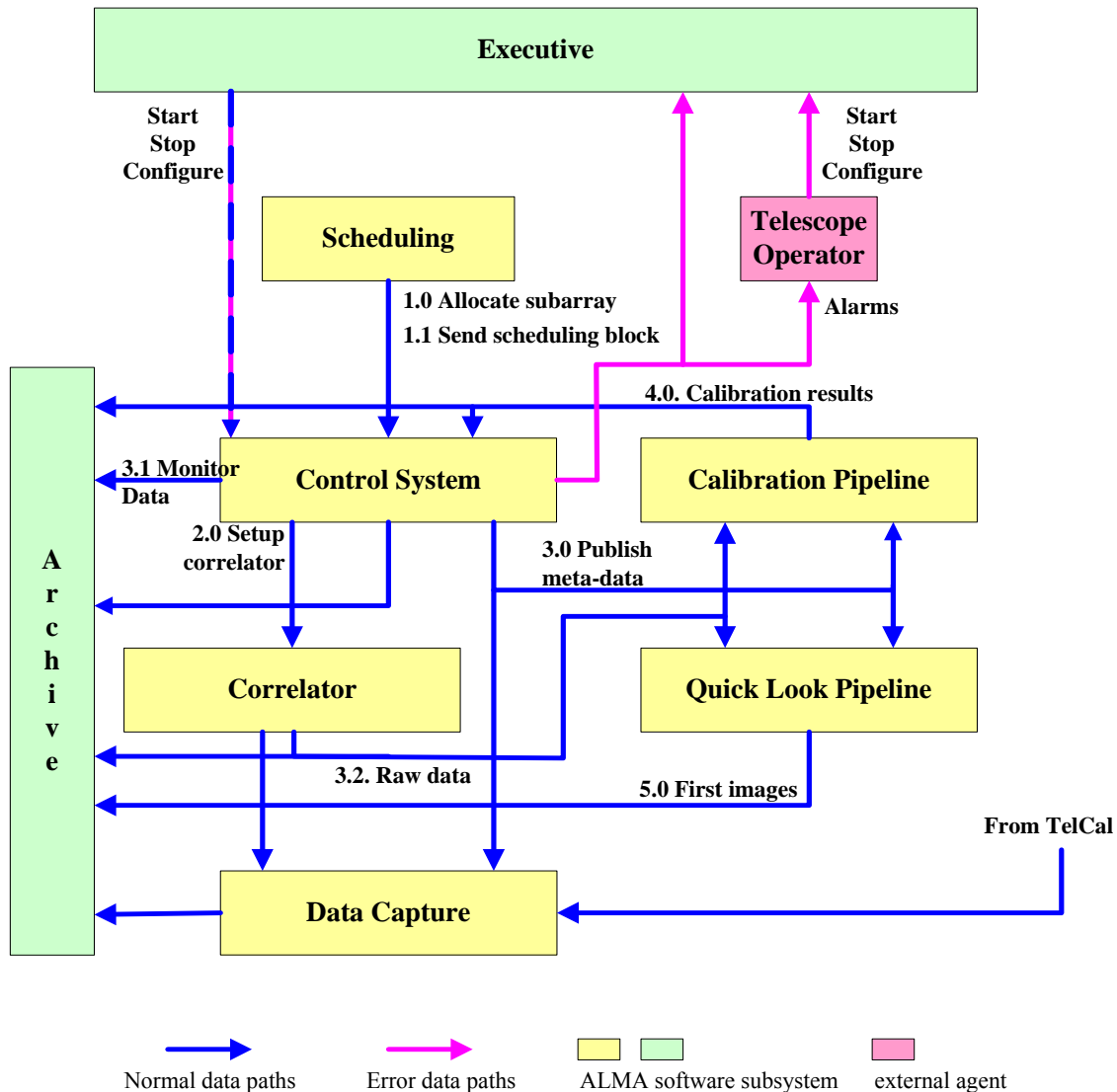


Figure 1: ALMA Software System (from the perspective of the control subsystem)

4.1 Overview

First, it is clear that we must have a concept of a low-level device that is intimately tied to the actual hardware that carries out the functions of the system. In addition the low level devices can be hierarchically structured in ways that reflect their relationships. Let us refer to this collection of devices and the software that directly manages them as the ALMA Device package.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 25 of 87

Next, the concept of monitoring plays a special role and is central to the mission of the control system. In fact, monitoring is a many faceted activity and we will see that it pervades the entire system. The monitoring data is to be structured, analyzed and stored. So, we will need a Monitor package.

Monitoring naturally brings up the issue of who is going to examine the data collected during monitoring and who is going to be the recipient of alarms and resolve anomalies. The function of telescope operator is an important activity that deserves its own package, partly because of its complexity but also because the telescope operator serves as a human supervisor who must know what is going on in order to be able to appropriately handle any unexpected event. So we will need a Telescope Operator package in order to support and assist the actual human telescope operator.

We have already stressed the concept of a sub-array. In fact, controlling and commanding one or more sub-arrays of the antennas that make up the ALMA telescope carries out scientific projects. Such activity is represented in the Array Controller package.

The action of commanding a sub-array in order to carry out some scientific mission requires that commands be executed. These commands may be at high level, using one of the standard observing modes, or at a lower level as a script containing exact descriptions of how the hardware is to be configured (for more specialised observations). Scripts are sequences of commands embedded within the usual looping and decision making structures. Either high or low level commands may be used. Let us call the package that translates these commands to instructions that the control the hardware the Manual Controller package.

In order to adequately carry out scientific intent of a scheduling the block the control subsystem must be able to make decisions based on the current conditions. This includes the current weather conditions as well as the results of recent calibrations. In some cases there must be a close collaboration between calibrations and observations in order to know when the observations should be terminated. So we need a package that collects data about and represents the current state of the system in a form that is useful and accessible to the command executor. Let us call this package the State Model package.

Activity within the control system must be coordinated. It requires careful management and synchronization. Furthermore, sub-arrays are dynamic. They are created, function for a time, and then cease, i.e., they are combined to form larger sub-arrays. There must be something that manages the creating and destroying of sub-arrays. This same package should start up and shutdown all the control sub-system i.e., software components and all ALMA devices. In addition it would manage resources so that only sub-arrays that can work independently would be created. We will call this package the Master Controller.




ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 26 of 87

These seven packages make up the architecture of the control system. All of these key concepts are present in the following re-statement of the description of the control system.

The **control system**, operating under the supervision of a telescope operator, monitors the ALMA telescope, a collection of devices, that function as one or more controlled sub-arrays. To control these sub-arrays a manual controller is employed that translates high-level commands representing scientific intent into executable device commands. This translation must take into account the current state of the telescope and its surrounding environment. All activity of the control system is coordinated and managed by a master controller. In addition, a Data Capture package provides a transition from the domain of the physical telescope and associated devices to the domain of the science being carried out with ALMA. These packages are depicted in Figure 2, with the exception of the Data Capture package, which is described in ore detail in Section 4.2.8.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 27 of 87</p>
--	---	--

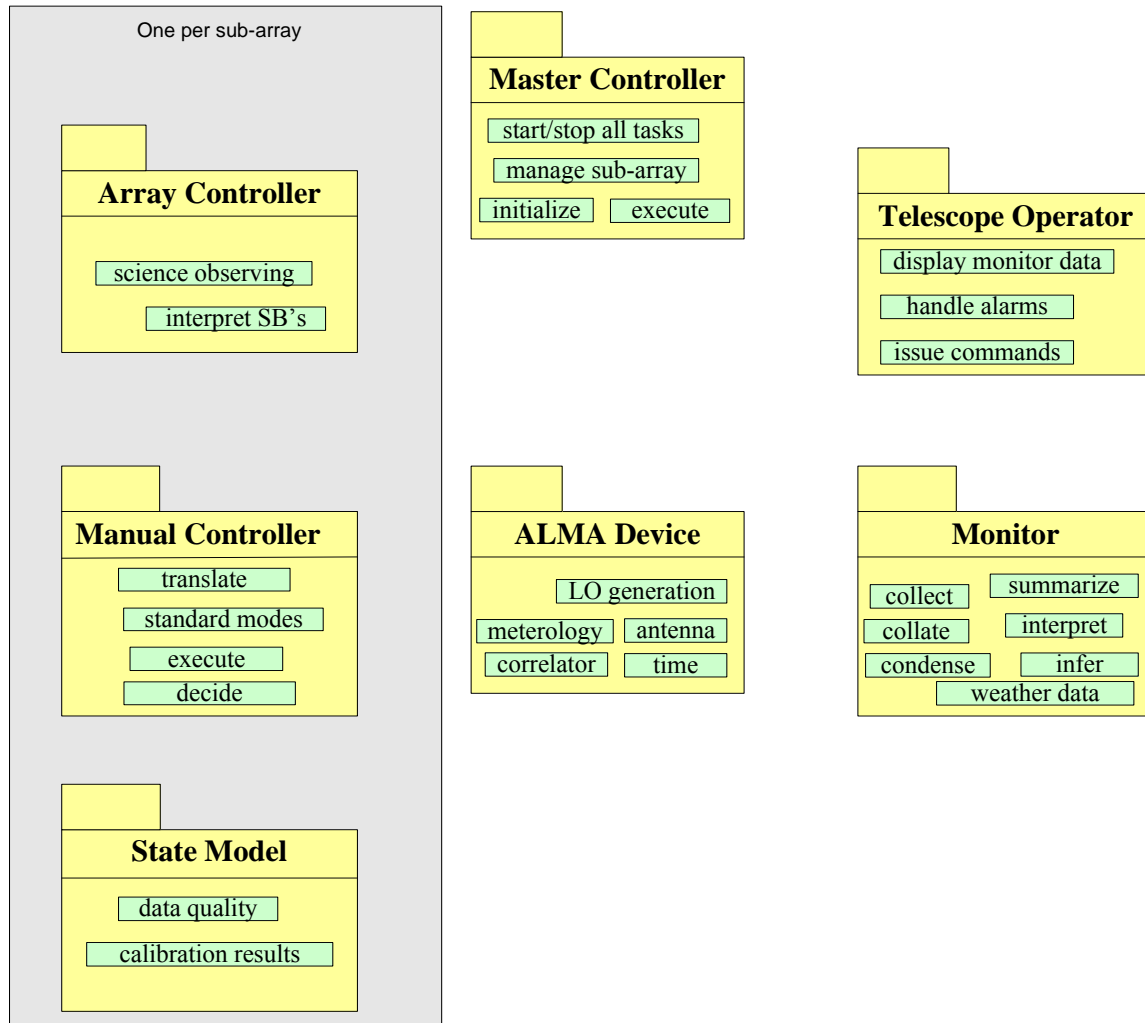



Figure 2: Control system Packages

4.2 Package Descriptions

4.2.1 The ALMA Device Package

The device package of the control system relies heavily on the ALMA Common Software (ACS) system². This system supports devices viewed as distributed objects that have static attributes (characteristics) and dynamic attributes (properties). These distributed objects are software components that are in direct control of hardware components. In addition, these distributed objects can communicate with each other via high-speed networks (CORBA Notification Channel). A generic model of an ALMA device is

² See http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/70_25_Common_Software/workspace

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 28 of 87</p>
--	---	--

shown in Figure 3. (This picture is a functional depiction and is not a class diagram in the object-oriented sense.)

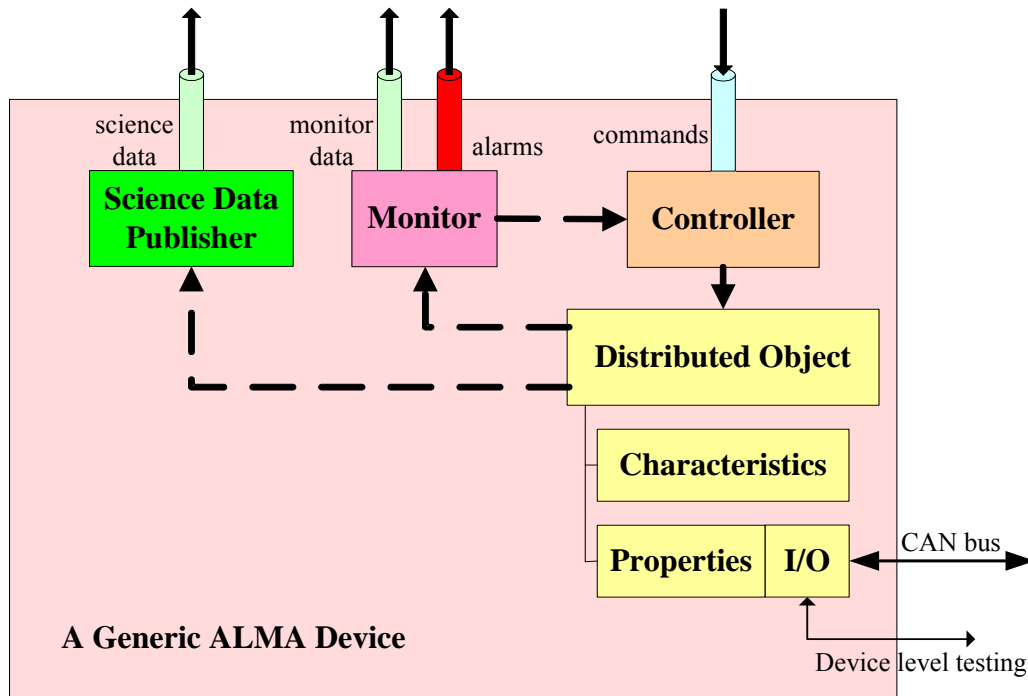


Figure 3: An ALMA Device

In general, the inputs to a device are a set of commands. Device level commands may be interpreted, internally, into a set of low-level commands that directly control the hardware. There is also special, direct access to the low-level device to support engineering testing and debugging. This mechanism bypasses much of the control subsystem and is not employed during normal operations. The hardware device, under the control of the software, generates routine monitor data as well as special alarms, warning of anomalous conditions. The generation of these alarms is based on the properties and characteristics of the device. In addition, a device can generate a stream of science data.

ALMA devices can be related to each other in any number of ways. The most common structure will be loose coupling and is shown in Figure 4. Here a controller device coordinates the actions of two devices that are attached to hardware. The output data streams are independent. This structure may be used to coordinate the maser and GPS devices in a higher-level “timekeeper” device.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 29 of 87

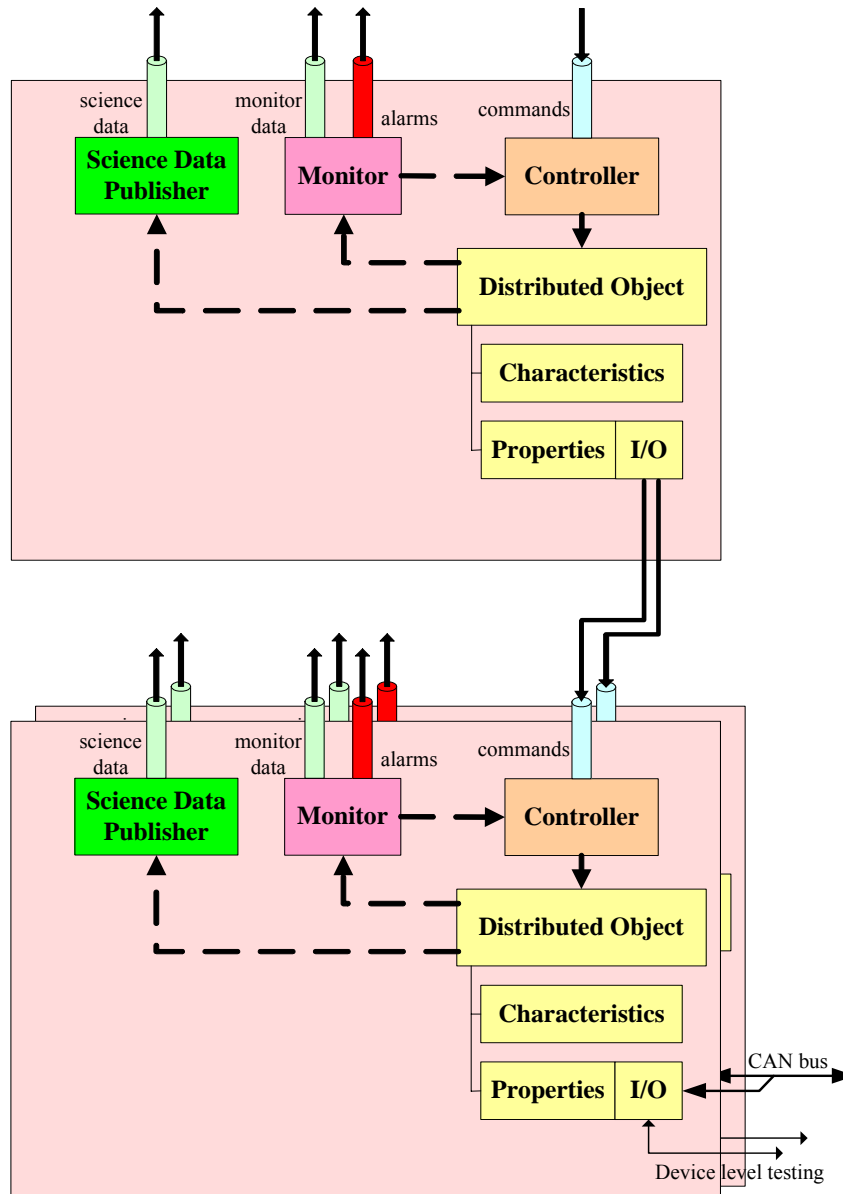



Figure 4: Loosely Coupled devices

Devices can also be structured hierarchically so that the alarms monitor, and science data from low level devices is all daisy chained to higher level devices. This will be the primary organization of the Control subsystem. Various interfaces within the system will provide access at varying levels of the hierarchy, depending on the specific needs of users and support personnel.

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 30 of 87
--	--	---

The device package within the control system is a structured implementation of the previous concepts and is depicted in Figure 6. Each box in the diagram represents a generic device in the previous sense. The black diamonds represent hierarchical relationships. The current Test Interferometer Control System (TICS) is largely concerned with implementing various aspects of the ALMA Device package.

4.2.2 The Monitor Package

The role of the monitor package is depicted in Figure 5. The Monitor is started by the Master Controller and runs continuously and independently of all other packages in the control system. It stops only on command from the Master Controller. The Monitor is an essential part of the telescope system.

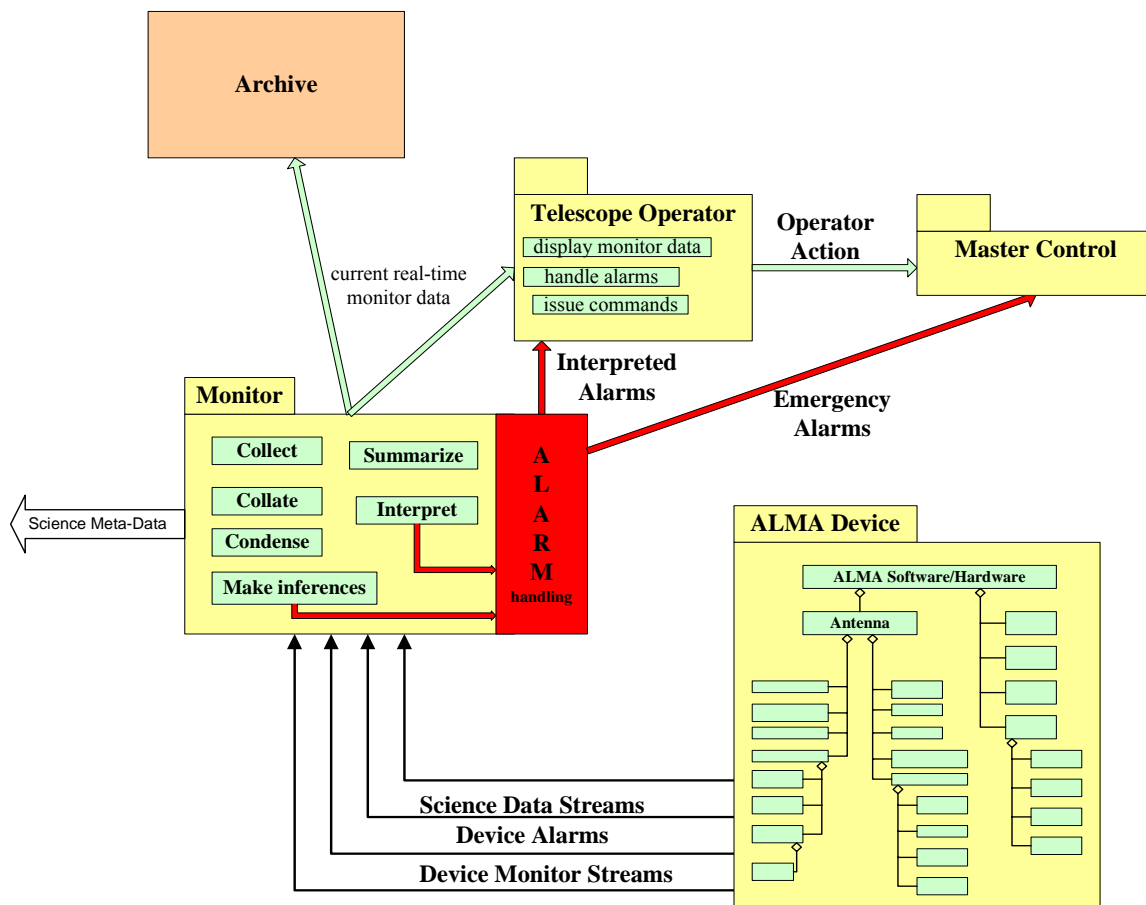


Figure 5 - The Monitor Package



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 31 of 87

The sampling of monitor points can be at rates of up to 20.833Hz however the archiving of any monitor points will be limited to a 2Hz rate. The monitor package will be responsible for reducing the rate and it will use different algorithms. By default the high rate data will be sub-sampled i.e., archive one of the samples and discard the rest, but in some circumstances the average, minimum or maximum data will be archived. The algorithm used will depend on what makes most sense for the monitor point.

The Monitor package has the following responsibilities.

- Collect monitor data from all ALMA devices
- Collect Alarms from all ALMA devices.
- Collect Science Data from all ALMA devices except the Correlator.
- Collate and organize data into intelligible structures.
- Condense data using lossless techniques.
- Store all monitor data in the archive.
- Interpret data in conjunction with past values e.g. trend analysis.
- Draw inferences using data that crosses device boundaries
- Provide meaningful high-level summaries of data.
- Send requested monitor data, interpretations, summaries, or inferences to the Telescope Operator package.
- Detect, report and log all alarms, whether directly detected or inferred, to the telescope operator. If the alarm is severe enough inform the Master Controller package using a high-priority mode.
- Provide science data to the Science Observing portion of the Manual Controller.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 32 of 87

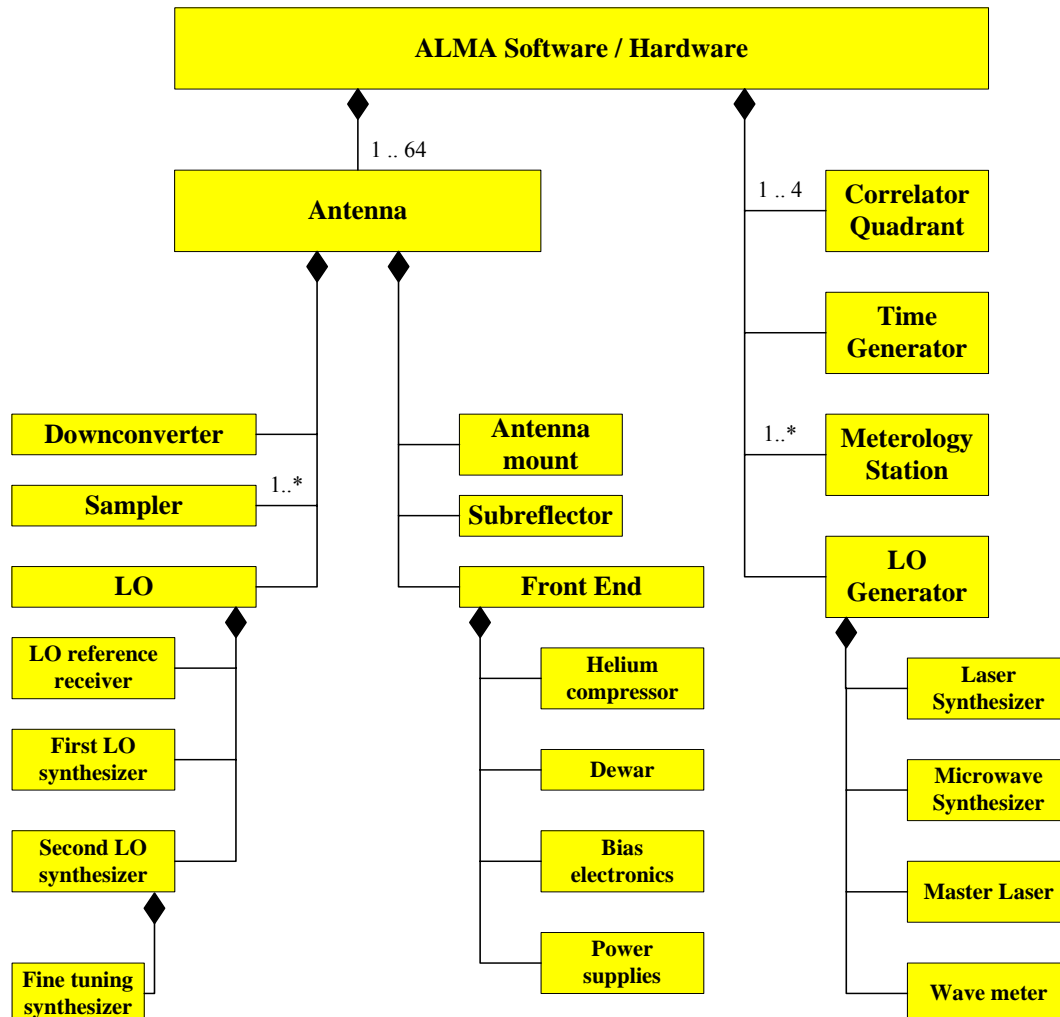



Figure 6: The ALMA Device Package

It will be difficult to fully test the functionality of the control system without all the hardware that it is intended to control. To avoid serial development we will develop “simulators” for every actual piece of hardware. These simulators are software that simulates, in a crude fashion, the behavior of every monitor and control point in the actual hardware. Simulators will be written so that they run on a separate computer connected via the hardware interface that will be used to communicate with the actual device. In addition it will be possible to run the simulators so that they can communicate via a software bus with device drivers. This will allow developers and testers to run the control subsystem and all the drivers and simulators on a single computer. In general, these simulators will not be part of a package, but will be used on an “as needed” basis for testing and debugging.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 33 of 87</p>
--	---	--

4.2.3 The Telescope Operator Package

The Telescope Operator package allows for multiple displays to multiple operators, separating the physical presentation from the logical structure of the information being displayed.

The Telescope Operator package provides mechanisms for display all monitor data, including interpretations, summaries and inferences. The state of any ALMA Device can be displayed, as well as aggregates such as individual antennas or sub-arrays.

The package also displays all alarms on a high priority basis.

The Telescope Operator package can, if the appropriate authentication is provided, allow a telescope operator to send commands to the Master Controller and Array Controllers. Via this command interface a telescope operator can:

- Stop the execution of a scheduling block,
- Destroy sub-arrays (which may interrupt an observation)
- Create sub-arrays in manual mode and manipulate them
- Stop the entire control system.

The Telescope Operator GUI must have a look and feel that is consistent with other subsystems, such as the scheduling subsystem, that interact with the telescope operator.

4.2.4 The Master Controller Package

The master controller is shown in Figure 7.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 34 of 87

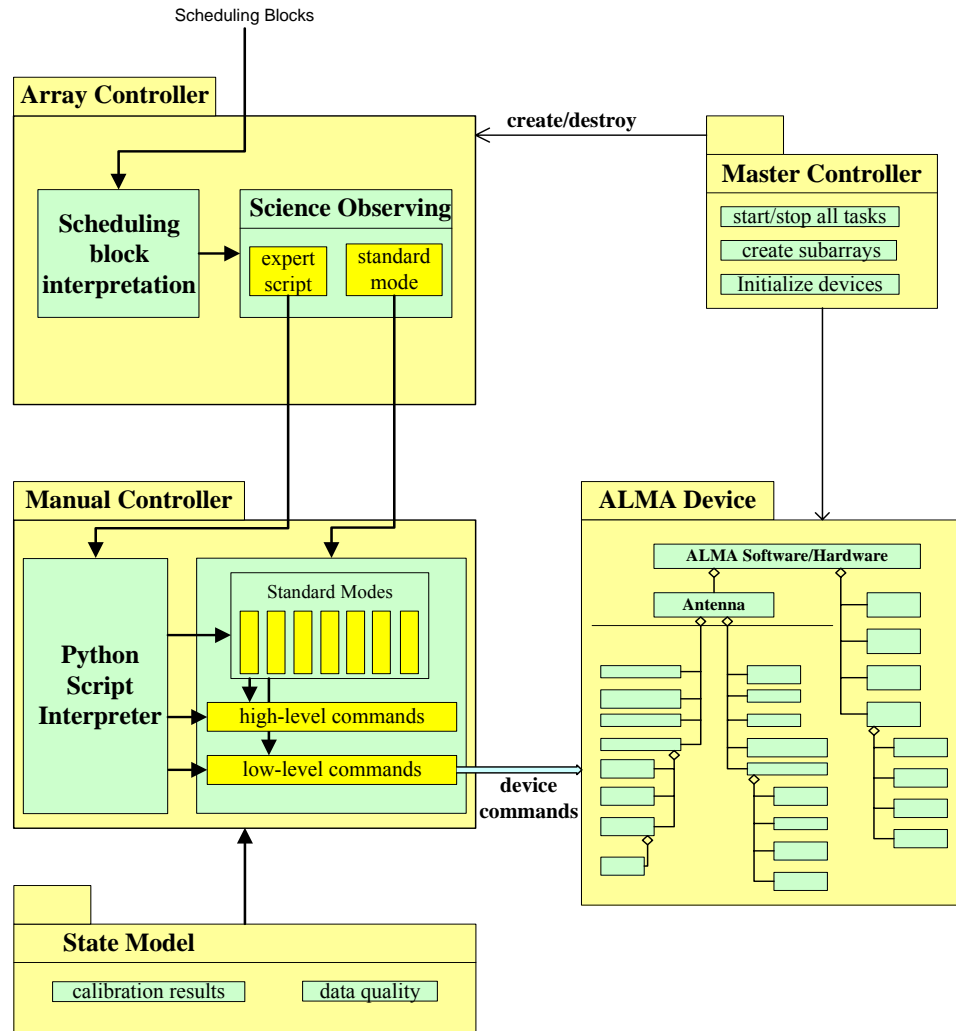



Figure 7: Master, Array and Manual Controllers & State Model

The Master Controller is responsible for the following activities:

- Initializing and configuring all ALMA devices.
- Stopping all ALMA devices gracefully.
- Initiating all tasks within the subsystem, such as monitoring, that must run continuously,
- Making certain that continuously running tasks are in fact running and logging any errors detected,
- Stopping all tasks within the subsystem,
- Creating sub-arrays from existing idle antennas in either manual or science observing mode,

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 35 of 87</p>
--	---	--

- Ensuring that sub-arrays can operate independently,
- Destroying sub-arrays (antennas are marked idle),
- Handling alarms and dealing with them appropriately, including the possibility of shutting down any ALMA device.

4.2.5 The Array Controller Package

The Array Controller package is also represented by Figure 7. Array Controllers are used for observations in automatic (science observing) mode where scheduling blocks are sent from the scheduler. A single instantiation of an Array Controller object controls one sub-array. In general, the Master Controller controls many Array Controller objects.

The Array Controller package has the following responsibilities.

- Interpret a scheduling block.
- Supervise the execution of a scheduling block,
- In the case of a “standard” mode, directly execute the command (using the manual control package).
- In the case of a “expert” script, submit the entire script to the interpreter in the manual control package
- Ensure that the scheduling block does not exceed its maximum allocated time.

4.2.6 The Manual Controller Package

The Manual Controller package is shown in Figure 7. A suggested implementation is also illustrated.

The Manual Controller contains an interpreter that executes scripts written in Python. This interpreter has access to the library of commands that control the ALMA devices. Therefore, when scripts that contain these commands are executed, the ALMA devices are commanded. This interpreter also has access to the body of information that makes up the State Model and the monitor package. So that scripts can also contain statements that access the state of weather, calibration data or any monitor data.

The Manual Controller contains the library of standard observing modes. In science observing mode, an Array Controller executing a standard mode from a scheduling block is merely passing a set of parameters to one of the standard modes in the Manual Controller. If the scheduling block contains a so-called “expert” script, that entire Python script is passed to the Manual Controller and executed by the interpreter.

The Manual Controller has the following responsibilities:



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 36 of 87

- Create and maintain the meta-data for the current observation and scan.
- Store the accumulated meta-data for an entire session (or scheduling block) in the archive.
- Maintain a library of standard observing modes.
- Provide high (science) level functions for controlling the array.
- Provide low level functions for more detailed access to the hardware
- Decompose all these functions into device specific commands

4.2.7 The State Model Package

The State Model package is also shown in figure 7. It is closely associated with the Manual Controller discussed above. The State Model package has the following responsibilities:

- Maintains a library of data representing the current state of the pipeline calibration results, and data quality information,
- Makes this library available to the Manual Controller in a form that is easily accessible by the script interpreter,
- Provides a listener that receives data from the calibration pipeline that represents the results of calibrations being carried out on current observations,
- Operates independently of the manual controller.

4.2.8 Data Capture Package

The Data Capture package translates the data that is being generated by the on-line subsystems i.e., Control, Correlator and Telescope Calibration and puts it into the archive in a format suitable for the offline subsystems. It is a joint responsibility of the Control and Offline subsystems and as such it is described in a separate document[7]. For completeness an outline of the Data Capture package will be presented here, with the focus being its interaction with the control subsystem.

The Data Capture package will be implemented as a Java component, created by an Array Controller object, as the array is built. The Data Capture package has interfaces used by the Control, Correlator and Telescope Calibration subsystems. Each of these on-line subsystems will send information to the data capture component as it is produced. The Data Capture package will buffer and reorder this information, generate indices and derived quantities to ensure that what it writes to the archive is complete and suitable for further processing by the data reduction subsystems. The output data format will be ALMA Export Data Format, as defined in [8].



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 37 of 87

There will be one Data Capture component per sub-array and it will be created, by the Control subsystem as it begins executing each scheduling block. The component will be deleted sometime after the scheduling block has completed (exact details are TBD).

If is useful, at this point, to briefly review the logical structure of an ALMA project.

The fundamental unit of a project is a Schedule Block. These are created by the Observation Preparation Tool, stored in the Archive, and retrieved by the Scheduler to be evaluated and scheduled for execution.


Each Schedule Block has an associated Execution Block. In practice, an Execution Block may need to be processed multiple times, so this can logically be thought of as a collection of Execution Blocks. Execution Blocks, in turn, are made up of a sequential series of Scans, each of which may consist of a series of Sub-Scans. And Sub-Scans will generally be made up of a series of integrations.

The Control subsystem will send information to the data capture component as XML documents. Data capture will use a schema to validate the format and contents of these documents before it begins detailed processing of their contents. There will be six types of documents that the control subsystem will send to the Data capture component. These correspond to start and end of the three highest-level structures within a Schedule Block. There will be one at the start of the execution, another at the start of each scan and a third at the start of each sub-scan. There will be corresponding documents that are sent at the end of each sub-scan, scan and at the end of the execution of each scheduling block.

These documents will contain information that is known by the control subsystem at that time. For example the “start execution” document will contain the number and location of the antennas, while the “end execution” document will contain the completion status for the whole execution. Similarly a “start scan” document will contain the scientific intent of the scan and the “start sub-scan” document, will contain the observing frequency. Exact details of the information sent by the control subsystem to data capture are being refined and more details are available in [7].

Not all monitor points will be sent by the control subsystem to data capture. In particular monitor points that are not used to generate information in the AEDF are not sent to data capture. However all points are separately sent to the monitor point archive which operates continuously even when observing is not occurring (unlike data capture).

Some data produced by the control subsystem will be voluminous, and the item of particular note is the output from the total power detectors on each antenna. There are six of these in each antenna and they are each sampled (producing two bytes) every 2ms leading to a total data rate of about 400kBytes/sec. This data and other high volume data

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 38 of 87</p>
--	---	--

will be sent to data capture in binary form in a manner similar to that used by the correlator output.

The correlations, as produced by the correlator subsystem, are not sent to data capture as the data volume is too high. Instead the correlator output streams directly into archive and suitable header and indexing information is sent to data capture.

Data capture may choose to discard or average some of the data that is sent to it by the control subsystem. For example pointing directions of each antenna will be sent to it sampled 21 times every second, but data capture may average this to one direction per integration.

There are additional features of the data capture design which I will only mention in summary because they do not directly relate to its interaction with the control subsystem. These are:

- Its interfaces with correlator and the telescope calibration subsystems (for input).
- Its interfaces with the archive (for output)
- Its ability to produce an interim AEDF format data that is used by the quick-look pipeline and the telescope calibration subsystem.
- Its ability to produce events that are used by scheduling and the executive to assess the status of the overall observation process.

4.3 Package Interactions

The relationships between the packages in the control system are presented below. Also included are inputs from other subsystems in the ALMA software system. All the packages in the control system are shown in figure 8. The arrows between the packages represent a summary of the points below.

ALMA Device:

- Initialization is from the Master Controller; configuration parameters are from the archive.
- Initialises all hardware
- Sends science, monitor & alarm data to the Monitor package.
- Provides low and device level functions to be executed by Manual Controller
- Lowest level commands provide direct access to the hardware for engineering testing and debugging.
- Receives shutdown commands from Master Controller.

Monitor:



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 39 of 87

- Receives monitor and alarm data from ALMA Device system.
- Collates monitor data and stores it in the archive.
- Sends monitor and alarm data to telescope operator.
- Sends alarm data and special commands in response to alarms to Master Controller.
- Provides science data to the Manual Controller.

Telescope Operator:

- Receives monitor and alarm data from Monitor.
- Submits commands to the Master Controller or the Manual Controller

Master Controller:

- Initializes all ALMA devices.
- Gracefully shuts down all ALMA devices.
- Starts and stops the Monitor.
- Starts and stop the State Model.
- Creates and destroys Array or Manual Controllers.
- Under special circumstances, shuts down any ALMA device.

Array Controller:

- Accepts scheduling blocks from the scheduling subsystem.
- Sends commands or scripts to Manual Controller.

Command Executor:


- Accepts commands for sub-arrays in manual mode.
- Uses data from State Model and Monitor package
- Generates commands for the ALMA Devices.
- Receives science meta-data from the monitor.
- Creates the execution block and stores the meta-data in the archive.

State Model:

- Receives data from calibration pipeline.

Data Capture:

- Collects and translates data being generated by on-line subsystems and puts it into the archive for later processing by the offline subsystems

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 40 of 87</p>
--	---	--

4.4 Package Integration

The entire system is shown in Figure 8. In addition to all the packages of the system, major and secondary data paths are shown, as well as control data paths and alarms.

The ALMA Device package can be developed independently of the other packages. All other packages are either directly or indirectly dependent on the ALMA Device package. As the schedule for delivery of hardware is variable we intend to write device simulators so that other parts of the software can be developed in parallel with the hardware development. The Manual Controller is dependent on the ALMA Device package. The Monitor and Master Controller packages are mainly dependent of the ALMA Device package, while the State Model and Telescope Operator packages are dependent on the Monitor package. The Array Controller package is dependent on the Master Controller package.

Monitoring the state of the ALMA system is carried out by the Monitor system, which operate continuously. Commanding is carried out by the Array Controller and Manual Controller and ultimately by the ALMA Device packages. The Telescope Operator and Master Controller packages supervise and coordinate the entire system. The State Model package operates independently of the Array Controller to receive the output from the calibration pipeline.

As data is generated by the Control subsystem and collected from other subsystems, such as Telescope Calibration and the Correlator, the Data Capture package captures this data. It is verified for content and format, before being placed in the ALMA Archive for later access and processing by offline subsystems.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 41 of 87

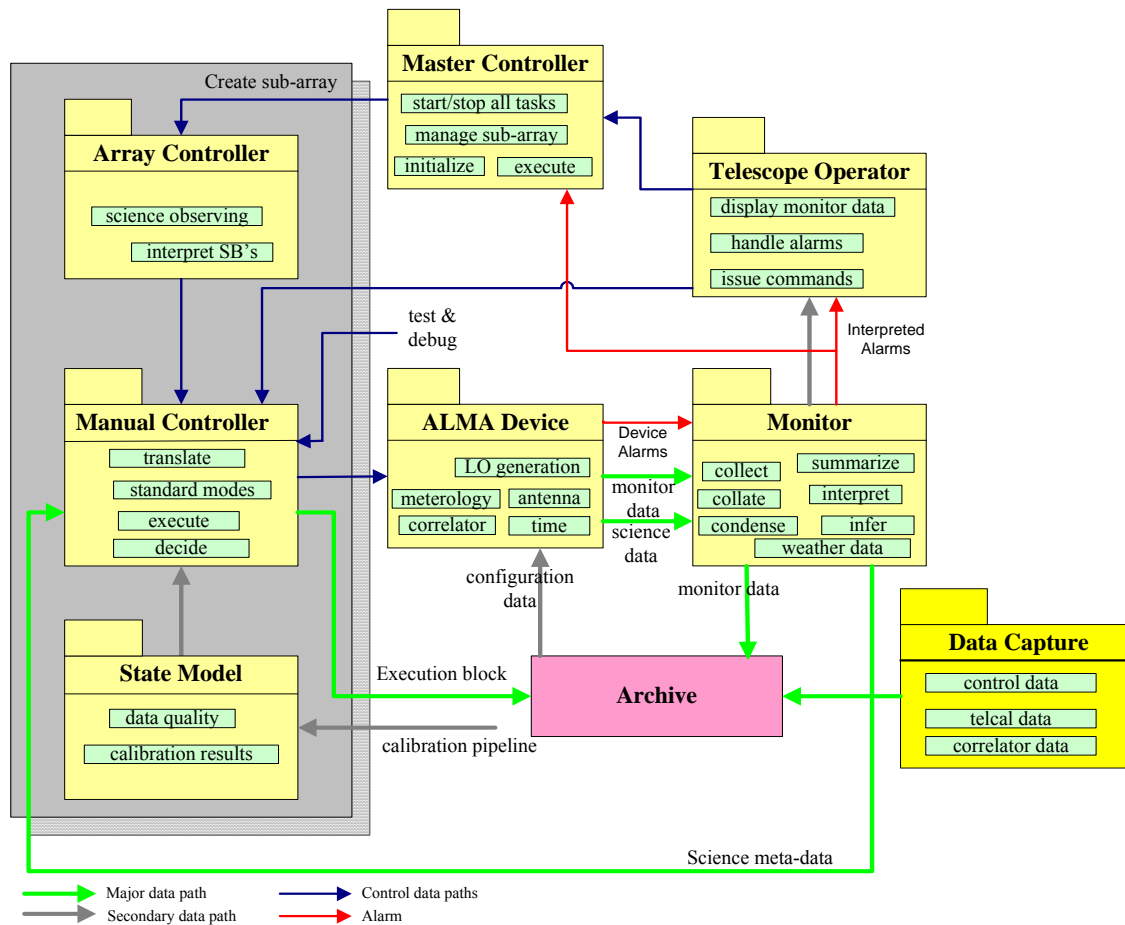


Figure 8: The control subsystem

4.5 Physical Architecture

4.5.1 Introduction

The physical architecture describes the layout of computers and networks, how the control system interacts with the hardware and the flow of data to other subsystems. Figure 9 shows the main aspects of the physical architecture. In this aspect the control subsystem will be very similar to the design used for testing the ALMA prototype antennas at the ATF.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 42 of 87

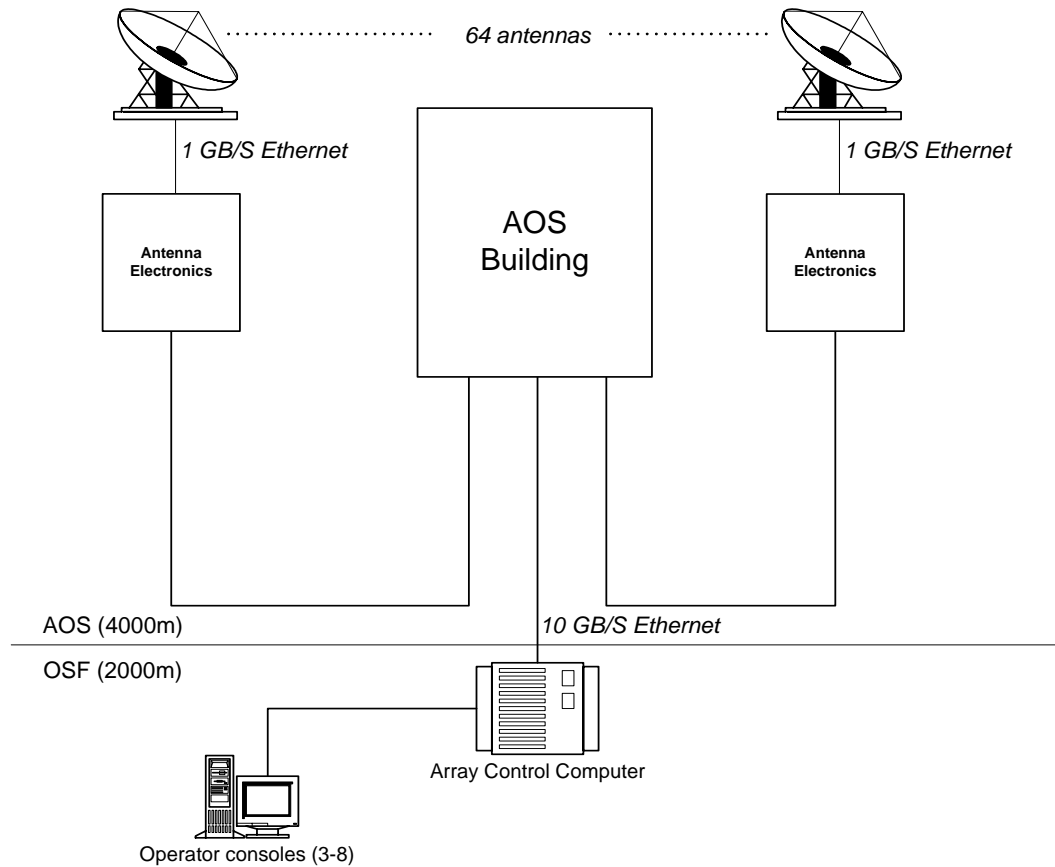



Figure 9 - Physical architecture of the control subsystem.

The control subsystem will deploy one computer, the Antenna Bus Master (ABM), in each antenna. This computer will manage communications i.e., Monitor and Control, to all the hardware devices in that antenna. This includes the antenna servos, receivers, and data samplers. A full set of devices that will be managed by the ABM is given Figure 10.

The ABM will communicate to all devices using an extension of the Controller Area Network (CAN) bus. The CAN bus is an industry standard, deterministic multi-drop serial connection that uses twisted pair cabling. The maximum bit rate on the CAN bus is 1Mbit/sec and the maximum cable length at this data rate is 35m. CAN uses one twisted pair and the ALMA extensions specify additional twisted pairs to carry a timing signal and a hardware reset signal. ALMA extensions also specify a master/slave protocol that the ABM (master) uses to communicate with all the devices (slaves). These hardware additions and the specification of the protocol are known as the ALMA M&C bus (AMB) and is specified in complete detail in [?].

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 43 of 87
--	--	---

An important point to note is that there can be no collisions on the AMB as all monitor and control requests are initiated by the bus master. This makes communications on this bus deterministic.

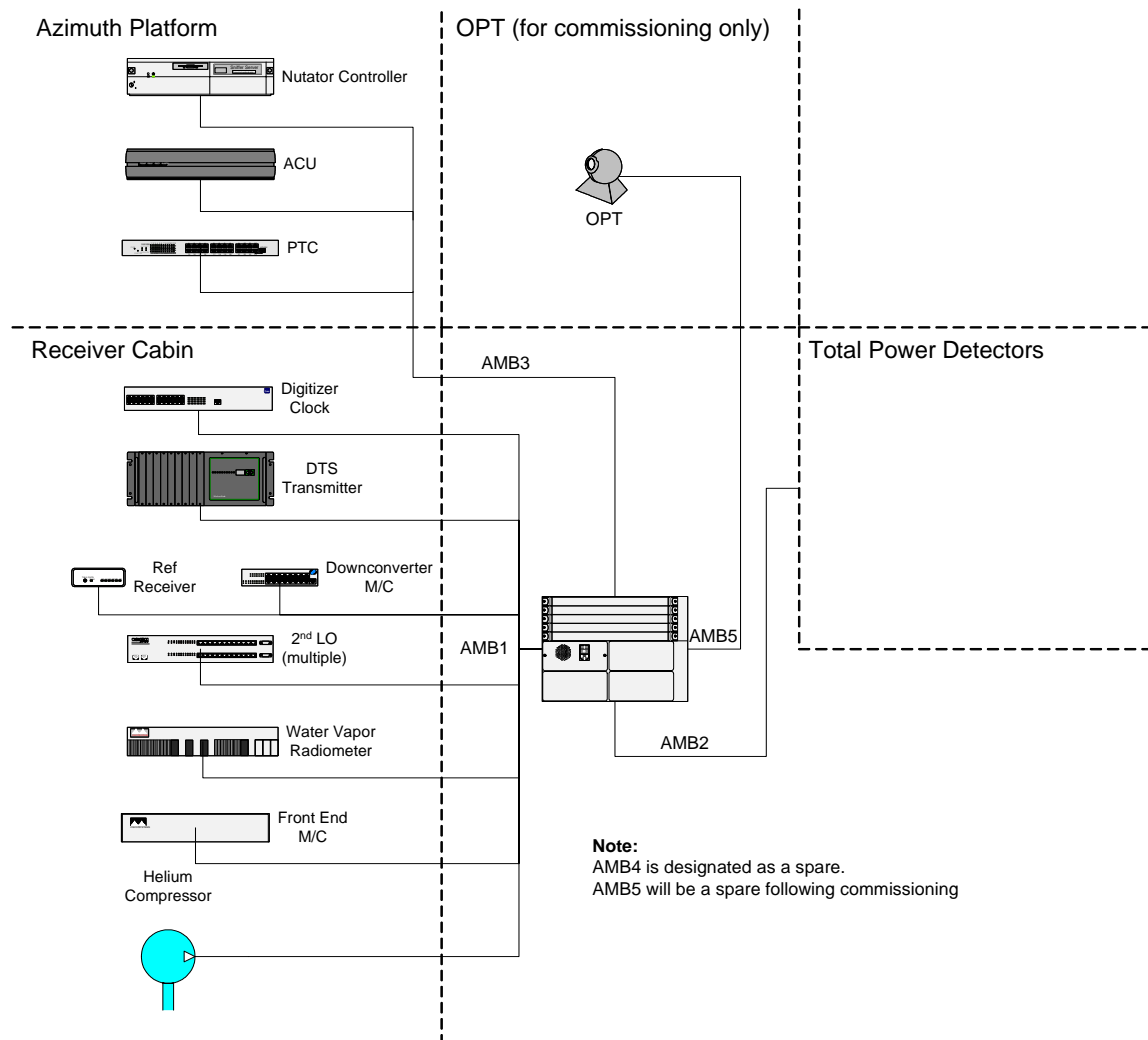


Figure 10 - Antenna Electronics

The cable length restrictions mean that one AMB cannot be used to reach all devices in an Antenna. We will place the ABM near the physical centre of the antenna (in the receiver cabin), so that an AMB can be run to the apex and another to the azimuth platform and pedestal room. The data rate restriction also dictates a separate CAN bus be used to connect the ABM to total-power receivers. We will dedicate one bus for all the equipment in the receiver room and at the ATF a fifth CAN bus is used to connect the optical telescope, which is attached to the back up structure of the reflector.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 44 of 87

The control subsystem will deploy a real-time computer near (within 40m) all central devices. Currently only one computer is planned (and one hot spare as discussed below). This computer is called the ARTM and will monitor and control devices like the local oscillator generator, GPS and maser (see Figure 11).

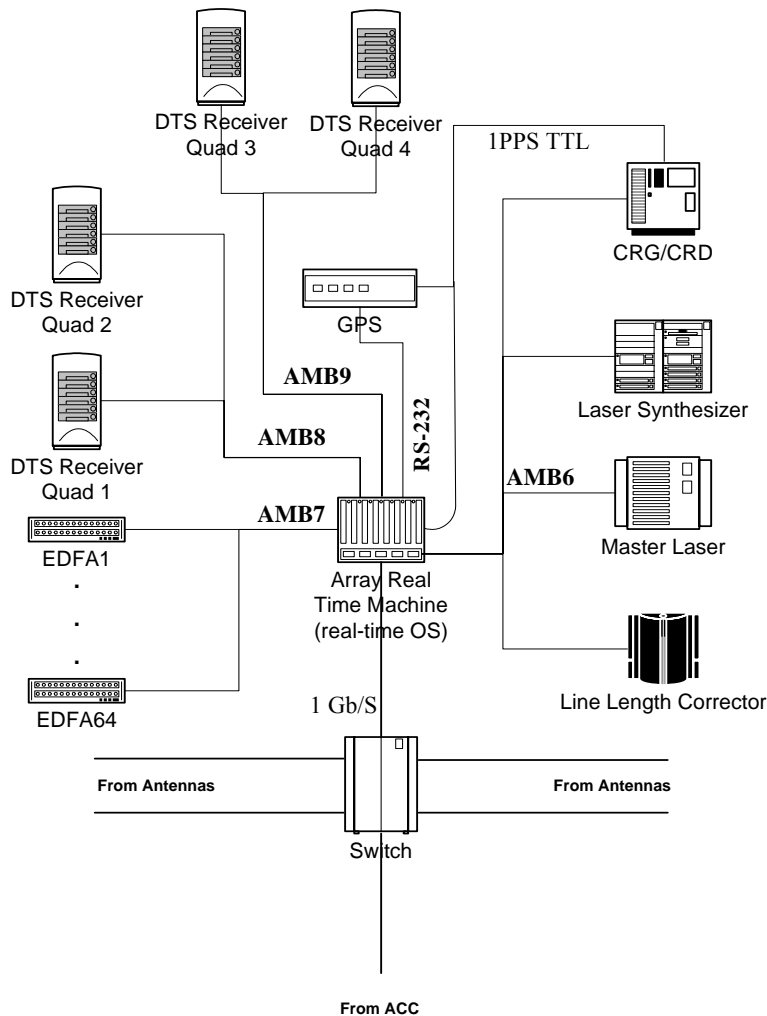



Figure 11 - Detail of AOS Building

As these computers are operating at 5000m we are planning to make them diskless, and hopefully passively cooled i.e., no fans. They will boot and load the software over the network from central computers at the Operations Support Facility (OSF), at 2000m. These computers will communicate using the ACS-provided CORBA protocol with a

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 45 of 87</p>
--	---	--

centralized computer (ACC) at the OSF that coordinates operation across all the antennas and other hardware via Ethernet.

The central computer will be standard computer running Linux and contain most of the packages described earlier (Master Controller, Array Controller etc.). Only the device package will run primarily on the computers at 5000m.

The high data rate astronomical data collected by the receivers and digitized by the samplers is not routed through the ABM and the control system. Instead it is sent directly to the correlator via specialized protocols on fiber optic cables (in the same bundle as the Ethernet signals). This is not true of the total radiometry data. However the data rate from these devices is much smaller (6000bytes/sec/antenna) and will be accommodated with a dedicated AMB bus between the total power radiometer and the ABM.

Communications between the control subsystem and the scheduler, archive, calibration and quick look pipelines will be via Ethernet within the OSF. It is not inconceivable that some of the other software subsystems, in particular the scheduler or calibration pipeline, will also run on the ACC.

4.5.2 Single-Point Failures

There are a number of places in the ALMA control software, where a failure will cause observing on all ALMA antennas to cease. These are termed single point failures and in this section we will enumerate them and list how we plan to address them.

Single point failures can be split into two categories, hardware and software. While the effects of both types of failures are similar, the remedies are different, so each will be discussed separately in the following sub-sections of this document.

Hardware

The hardware category refers to an electronic failure in the computers or other equipment. The equipment which falls into this category includes:

- ARTM
- GPS
- ACC
- Network link between the AOS and the OSF

In the sections that follow, we address each of these systems in turn.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 46 of 87

ARTM

We plan to install two ARTM's. At any instant only one is active, and the other one is turned off. However both are connected, in parallel, to the same AMB busses, GPS receivers and Ethernet switches. It will be possible to remotely switch either of these computers on (but only one should be on at any one time). In the event of a failure of the primary ARTM, the secondary ARTM can, from the OSF, be switched on and the software can be restarted on the secondary ARTM.

The scheduling block that was being observed at the time of this failure will need to be restarted. It is expected that the total amount of lost time for an ARTM failure will be around one hour, and the switchover can be done, by the telescope operator, from the OSF. Maintenance personnel will, at the next convenient opportunity, diagnose the fault in the primary ARTM and this will involve a trip to the AOS.

GPS Receiver

The GPS receiver is commercial equipment that is used to initialize the time on the Master timekeeping electronics. Hence it is only essential when the relevant electronics has been turned off. At other times the GPS is used to monitor the drift in the timekeeping electronics.

Assuming the electronics is correctly initialized ALMA can run quite satisfactorily without the GPS. So we do not plan to install a redundant system.

Array Control Computer

The ACC is located at the OSF and hence can be attended to by ALMA personnel in the event of a failure. As with the ARTM we will have a backup computer. The simplest solution is to have a complete spare ACC which can be started when the main one fails.

An alternative, which we will assess shortly before purchasing the ACC is to use either a high reliability computer which incorporates the ability to continue operation if some of its components, like disks and memory fail.

As with an ARTM failure it is expected that the total amount of lost time for will be around one hour and this includes the time lost in having to re-observe the scheduling block that was interrupted.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 47 of 87

Network Link

Failure of the network link is probably the most catastrophic single point failure mode. We have specified two, 10Gbit/sec network links between the AOS and the OSF. It is expected that both these links will be in the same trench. A failure in one of these network links will not halt all observing but will result in a reduced bandwidth and hence may restrict the observations to those that do not use high data rates.

A failure in both links will stop all observing.

Software

Much of the ALMA control software will be running on the ACC, so that a bug, leading to a failure in this software, will halt observing. We will address this by vigorously testing the software, initially against software simulators, then at the ATF and finally during commissioning. In addition we will always have the ability to revert to a previous version of the software so that lost time due to bugs in the new version of the software can be minimized.


The software in the ABM's will largely be identical, so it is likely that a bug in this software will affect all antennas. As such the ABM software, along with the ARTM software, can also be considered a single point of failure.

We will have, at the OSF and the relevant regional centers, suitable equivalents of the real-time computers and the array control computer so that software faults can be diagnosed and repaired at either location. It is expected that minor faults will be diagnosed and repaired in Chile and major faults repaired at the regional centers.

4.5.3 Electrical Interfaces

The electronics which comprise the Control subsystem and its link to the Correlator subsystem are made of up of three different signaling protocols. Each protocol has its own strengths and weaknesses and serves a different purpose in the ALMA system.

A general computer network is provided by Ethernet. This is used for communication between varying computers at the AOS, as well as communication between the AOS and the OSF. Ethernet signals are routed through high-speed switches, located in the AOS computer room and the patch panel room.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 48 of 87</p>
--	---	--

Data transfers between the antennas and the Correlator subsystem are carried out via a high-speed fiber link. This link implements a custom communications protocol, carried out over DTS transmitter/receiver pairs. No control data, control commands, or monitor streams are carried on this interface.

All other communications are carried out over ALMA's extension of the CAN bus – the AMB (ALMA Monitor and Control Bus), described elsewhere in this document. One AMB link goes directly from the ARTM to the Correlator. Other AMB connections are routed through the patch panel room, where they can be moved between antennas and other equipment as needed.

4.5.4 AMB 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 in Brooks and D.Addario (2001) which governs the behavior of slave nodes at the devices. This bus is hereafter referred to as the ALMA Monitor and Control Bus (AMB). That memo also defines the timing specifications for communicating with devices which have synchronization associated with 48 ms Timing Events. A distributed reset pulse is also defined for the purpose of remotely resetting all nodes on a CAN bus.

In the ALMA system there will be a number of different operational CAN buses:

- o At least one at each antenna for M&C (most likely two)
- o One at each antenna for total power samples
- o One in the central control building for M&C

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 O/S: VxWorks. SBC: MVME2700, MVME1603, MVME2604. CAN Interface: Tews Datentechnik TP816 Dual and Single Port PMC CAN modules (available in the U.S. from SBS Greenspring)
- o O/S: Windows NT. Any PC platform with PCI. CAN Interface: National Instruments Dual Port PCI CAN board. LabView bus master routines.
- o O/S: Linux/RTAI. VME platforms with Motorola processors and TP816 PMC CAN module.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 49 of 87

- o O/S: Linux/RTAI. Motorola processor VME platforms with TP903 PMC CAN module.
- o O/S: VxWorks. Motorola processors in VME platforms, using TP903 PMC CAN module.

Bus Slave Nodes

There are currently two designs for circuits allowing the connection of hardware devices to the ALMA Monitor and Control Bus (AMB) in response to the requirements defined in Brooks and D.Addario (2001). These two circuits are referred to as the AMB Standard Interface, Type 1 (AMBSI1) and the AMB Standard Interface, Type 2 (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 which requires that no slave node initiate transactions on the bus unless polled by a bus master.

The AMBSI1 and AMBSI2 have been developed to serve the needs of device designers requiring varying degrees of complexity in local hardware monitoring and control, and for accommodating different physical size constraints. The AMBSI1 requires space for a Eurocard 3U height board and associated connectors. It provides a powerful 16-bit microcontroller with default firmware providing a small set of I/O capabilities. Device designers may also implement additional device-specific code to access the serial ports or to interface to devices on the external bus. The use of device-specific firmware is expected to be the main area of deployment for the AMBSI1.

Thus, the purpose of the AMBSI1 is to incorporate the AMB protocol and device-specific functionality onto the one board. Conversely, the AMBSI2 is a very small daughter-card for use in devices which either have their own microprocessors or need a very minimum amount of I/O to the CAN bus. There is no provision for device designers to add device-specific firmware to the AMBSI2.

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 closing fast sub-millisecond control loops. The board also provides for the Global Slave Reset pulse and 48 ms Timing pulses to generate appropriate signals to the micro-controller.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 50 of 87

The ALMA Computing Group is responsible for developing any device-specific 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 AMBSI 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 so as 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. Self-test routines should also be included in user code.

AMBSI2

This device performs the function of a CAN bus to Serial Peripheral Interface (SPI) converter. It 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 resulting subsystem requires +5 Volts at less than 2 ma and occupies a small daughter PCB of approximately 2 in by 1 in. The PCB is mounted to the main PB Board by several multi-pin headers which also serve as the I/O connectors for the subsystem.


There is 16 bytes of RAM accessible by either the CAN bus or the SPI port. An *Attention* output indicates when RAM contents have been altered by the CAN bus. The unit has its own oscillator and reset circuitry, completely independent of the host microprocessor. All of the software in the subsystem is developed by and the responsibility of the software group.

In addition to providing the CAN interface, the unit also provides the interface for both the Global Slave Reset and 48 ms Timing pulses which come in through the AMB DB-9 connector or module wiring as RS-485 and is available as TTL signals to the application circuitry.

5 External Interfaces

A detailed treatment of all external interfaces provided and used by the scheduling subsystem is available in the Control Subsystem Interface Control Document³.

³ See Control ICD in <http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/Interfaces>.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 51 of 87</p>
--	---	--

6 Development

The development plan for the control subsystem is presented in the ALMA Software Development Plan for the Control Subsystem⁴. The control subsystem will be object-oriented, using Java as the primary implementation language. The command interpreter is Python and many of the observing modes will be coded in Python. Some packages, such as the device drivers will be implemented in C++.

Viewing the architecture of the control system presented here provides a vantage point from which to view the current Test Interferometer Control System (TICS). The primary purpose of that system is to implementing various aspects of the ALMA Device package in order to support hardware testing and debugging. There are also implementations of certain selected portions of the Command Executor and Monitor packages.

6.1 Control Command Language

This section is a summary of a preliminary version of a proposed "Control Command Language"(CCL) for the ALMA project. These are the commands that "expert users" may place in scripts to be executed during their Scheduling Block (SB). We will also provide an environment, known as "manual mode", in which users can type commands directly at a console. It is expected that only qualified staff astronomers and engineering staff will use this mode, primarily for developing standard observing modes and diagnosing equipment problems. This section does not address the unique problems inherent in manual mode – such as the need for the Control subsystem to create a SB. As such, the commands described here will be a major subset of the full set of commands that will be available in "manual" mode. We are primarily concerned here with the use of the command language in pre-written scripts to be executed.

The exact mechanism for getting into the script environment has been discussed among the Control group and with the Scheduling group. We have come up with a mechanism that we feel will provide the scripting environment in a fairly straightforward manner. The basic procedure will be:

1. At the time a SB is created for an observation that includes a script, it will be marked as an "expert mode" SB and a script will be attached to it. Based on work done earlier by the ObsPrep subsystem group, the script would be part of a SB's "dynamic" component and will be embedded as a string which will be passed to the Control subsystem for execution.

⁴ See Control Software Agreement in http://almaedm.tuc.nrao.edu/forums/alma/dispatch.cgi/DRAFT_Computing_Plans.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 52 of 87

2. When the "expert mode" SB is chosen for execution, the Scheduling system will query the Control system for a list of available antennas. Scheduling will then ask Control to create the appropriate subarray for this SB.
3. Scheduling will then pass the SB to the Control system, which will identify it as including a script to execute, create an execution environment for the script, and submit the script for execution to the newly created array (an ArrayController within the Control subsystem) which will execute the script.

6.1.1 Language Details

The basis for the CCL will be Python. Python was chosen based on the recommendation of Barry Clark, after being encouraged by Robert Lucas and his experiences of using Python at the ATF. Both Robert and Barry pointed out some features of Python that they found useful, in particular the object-oriented syntax in which commands (methods) are associated with appropriate objects. Additionally, Ralph Marson was quick to point out possible simplifications in the implementation of the CCL that became possible through the use of Python.

Since the language will be object-oriented, every command executed will need to be provided by an object in the environment. At this stage, it seems appropriate that all commands are made available through an instance of the Array object, already a defined class in the ALMA Control subsystem. The Array object is created when the SB is submitted for execution and will be bound to a Python object (Array) in the execution environment. In order to accomplish the goals of the high-level commands, it may be necessary to issue several lower level commands. These commands may be implemented by other objects which will not normally be exposed to the script writers. As the language is extended to address more complex use cases, we will refine these boundaries to expose lower level objects when necessary, while hiding complexity where possible.

6.1.2 Language Structure

The CCL is being designed to follow a hierarchical structure. This hierarchy is expressed in the objects that are exposed to script writers.

At the highest levels of the language are the more abstract objects which encapsulate several different lower level components of the ALMA system. For instance; the highest level object exposed to script writers in the CCL is the Array object. This object represents a collection of antennas that accept commands as a single unit. It also encapsulates many aspects of the Correlator subsystem. It is expected that the majority of scripts will be written by taking advantage of the Array and the ancillary objects supporting it.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 53 of 87

The next lower level of the CCL objects is represented by the Antenna object. This single object represents everything needed to produce a working antenna object in the real world. It includes the antenna's mount, the LOs, FTS, Baseband Processor, etc. Through the Antenna object, commands may be passed to any of these components.

Below the level of the Antenna object, users may get access to components that make up an antenna structure. In the current implementation, only the Receiver object is accessible at this level, but others will be exposed as the CCL develops.

One artifact of this hierarchical structure is that there may appear to be redundant commands, when the CCL is considered in its entirety. For instance; there is a "setDirection()" method in both the Antenna object and the Array object. Both accomplish the same end result, although in the former case, it affects only a single antenna, while in the latter, all antennas in an array are directed to a new pointing direction. This redundancy has been intentionally designed in to the language in order to simplify the transition between the higher level objects and the lower level objects.

6.1.3 Parallelism and Synchronization

As mentioned briefly before, the mechanism for taking advantage of parallelism and of synchronizing parallel processes in the CCL will be accomplished by Python Threads.

Most distributions of Python – on any platform – include the "thread" library. This library includes mechanisms for creating independent threads of execution ("Lightweight Threads"), waiting for a thread to complete, and synchronizing access to shared resources, through software "locks".

In order to take advantage of these capabilities, Python scripts need to have the appropriate import statement;

```
import thread
```

New threads of execution can be created by executing the `start_new_thread()` method. This will create a new thread and begin executing a command passed in as an argument. A trivial example would be:

```
thread.start_new_thread(time.sleep, (10,))
```

This would start a new thread of execution and immediately put it to sleep for 10 seconds. In the meantime, commands following this one in the script would continue to execute while the new thread sleeps.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 54 of 87

As a final example, let's assume a user wants to start a thread in a script, execute some other commands, and then terminate the script only after the sleeping thread has completed. With Python threads, this can be accomplished in a very straightforward way;

```
testThread = thread.start_new_thread(time.sleep, (10,))  
.  
.  
.  
testThread.join()
```

This code fragment will spawn another thread which again only sleeps for ten seconds. After the thread is created, other commands are executed. When the other commands are complete, the script pauses at the “join()” command to wait for the thread to complete. After the “join()” is satisfied, execution of the script ends by the simple expedient of running out of lines to execute.


Additional examples of using threads will be provided in the appendix of this document. More details on the Python thread module can be found on-line at:

docs.python.org/lib/module-thread.html

6.1.4 Language Objects

Since the CCL will be object-oriented, each command – or “method” – available in the language must be implemented by an object of some class. The following list of objects will be defined in the CCL and will be made available for script-writers to use. In later sections of this document, the methods implemented by each object will be detailed.

Object	Description
Array	Created by Control subsystem at request of Scheduling to represent all antennas assigned to the current SB. May also be explicitly created by a script, such as in cases where a users wishes to divide a single Array into two or more smaller Arrays.
Antenna	Object through which a script may interact with a single antenna. Antenna objects are created by requesting them from an Array object.
CorrelatorConfiguration	An object containing fields representing each of the parameters that can be set to control operation of the Correlator.

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 55 of 87
--	--	---

Source	Represents an astronomical source to be observed. A Source may be created by retrieving a named object from an observing catalog or may be created by specifying a coordinate pair in the RA/DEC or AZ/EL form.
ScanPattern	This object is used to describe a physical scanning pattern the array (or antenna) is to use for observing. It may be attached to a Source object to drive the antenna during the SB execution. The scan pattern is superimposed on the motion of the source.
Receiver ⁵	A receiver installed in an antenna. This object may be, obtained from the Antenna object and accessed directly so that more precise adjustment may be made by an experienced script writer.
Nutator	Like the receiver object the Nutator object may be obtained from the Antenna object and it provides access to Nutators. Most antennas in the ALMA will not have a nutator.
NutatorProgram	An object which contains a list of positions that the Nutator should switch between when it begins running. It is used by the Nutator object.

Table 1 - CCL Objects

6.1.5 Method List by Object


Array

status() : Returns the status of this array. This would include such information as whether or not all antennas are functioning, if the array is on source, and if the array is being used for an observation. Possible return values include IDLE, BUSY, DOWN.

antennas() : Returns a list of all the antennas which make up the current Array object. This will return an array (tuple) of strings with each string being an antenna name.

setDirection(Source) : For each antenna in this array, set its direction to be towards the object defined by the “source” input parameter. This method returns when all antennas

⁵ **Note:** There is a discrepancy between Engineering and the Scientists over what to call this device. Engineering refers to a “Receiver” as an electronic system, including multiple cartridges, each of which enables the “Receiver” to accept a different range of signals. Astronomers, however, refer to each “cartridge” as a separate “Receiver”. This ambiguity needs to be resolved, but will likely require input from the SSR and Systems Engineering groups.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 56 of 87</p>
--	---	--

have begun moving to the requested direction. Each antenna will also begin tracking if the source refers to a “trackable” object.

getDirection() : Return a list of current pointing directions for each antenna in this array.

getPointingError() : Return a list of the difference between commanded pointing direction of the antennas in this array and their current pointing directions.

setSkyFrequency(freq) : Tunes the receiver to the frequency designated by the “freq” parameter, which is given in Hz⁶. This method will switch the receivers to use an appropriate band (if necessary). This method returns immediately, even though the actual process of tuning may take some time.

setBand(bandName) : Switches the receiver in each antenna to use the specified band. Normally this is not necessary as the setSkyFrequency command will do this. However, frequencies between 84 – 90 GHz can be observed using either the band 2 or band 3 receivers. Using this function prior to calling the setSkyFrequency will determine which band is used in this situation. This method returns immediately, even though the actual process of tuning may take some time.

getSkyFrequency() : Return a list of observing frequencies that the for each antenna in this array. Normally they will all be the same. Note this does not mean that the relevant receiver is “locked”.

isLocked() : Return a list of Boolean values indicating whether the receivers in each antenna are phase locked.

observe(ACSTime) : Begin collecting data, using configuration parameters already set. Observing will begin at the time specified by the ACSTime parameter or immediately (NOW) if the time is not given.

beginScan(Purpose) : Marks the start of a single scan for data processing purposes. This method causes the Control subsystem to generate an event which notifies other subsystems that a scan is beginning. The Purpose parameter is used to differentiate scans based on the intent of the data collection.

beginSubScan(Purpose) : ALMA allows scans to be divided into smaller units – sub scans. This method marks the beginning of a sub scan. The Purpose parameter serves the

⁶ To simplify parameters which represent frequency, time, etc., we have decided that these values will be entered in fundamental units, rather than using multiplied units. This eliminates uncertainty – for instance - over whether a specific parameter is in Hz, MHz, or GHz. Under this philosophy, a sky frequency of 165 GHz, would be entered as 165E9.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 57 of 87

same function as in the `beginScan()` method. This command cannot be issued unless a ***beginScan()*** command has previously been given.

endScan() : Marks the end of the most recently started scan.

endSubScan() : Marks the end of the most recently started sub scan. This command must be given before an ***endScan()*** command may be used.

configureCorrelator(startTime, CorrelatorConfiguration) : Send the `CorrelatorConfiguration` object which is passed in as a parameter to the Correlator in order to prepare for observing. Due to limitations of the Correlator, observing should not begin until at least 2 seconds after this method is executed. “`startTime`” represents the array time when the `CorrelatorConfiguration` should be used.

getTelCalResults() : After the end of a scan (or sub scan) whose purpose is to perform some type of calibration, this method will retrieve the results from the TelCal subsystem.

getAntenna(AntennaId) : Returns an `Antenna` object so that the script may interact directly with a single antenna if needed. The `AntennaId` specified must be a member of the `Array` object to which this command is sent, or an error occurs.

splitArray(AntennaIdList) : Constructor for `Array` object. Creates a new instance of an `Array` comprised of the antennas specified in the `AntennaIdList` parameter. All antennas in the list must be members of the parent `Array` for this to succeed.

removeAntennas(AntennaIdList) : Removes all antennas specified in the `AntennaIdList` parameter from the current `Array` object. They may then be reassigned to an existing `Array` object or used to create a new instance of `Array`.


addAntennas(AntennaIdList) : Adds the antennas specified in the input parameter to the current `Array` object.

Antenna

setDirection(Source) : Set this antenna’s direction to be towards the object defined by the “`Source`” input parameter. This method returns when the antenna has begun moving to the requested direction. The antenna will also begin tracking if the source refers to a “trackable” object.

status() : Returns the current status of this antenna.

getDirection() : Return this antenna’s current pointing direction.

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 58 of 87
--	---	---

getPointingError() : Return the difference between the antenna’s commanded pointing direction and the current pointing direction.

setBand(bandName) : See the setBand function in the Array object.

getSkyFrequency() : See the getSkyFrequency() command in the Array object.

isLocked() : See the isLocked function in the Array object.

setSkyFrequency(freq) : See the setSkyFrequency object in the Array object.

getReceiver() : Returns a reference to the receiver in the current antenna. Using this Receiver object, a script may gain access to the LO chain, FTS, Baseband Processor, and other elements of both the “Front End” and “Back End”.

getNutator() : This method returns to the user a reference through which the Nutator may be accessed. This method will fail if the antenna is not a NutatingAntenna.

Source

isObservable() : This method returns a Boolean value indicating whether or not this Source object is in a position to be observed from the current location.

isTrackable() : Boolean method that tells if the object can be tracked. For a Source object such as “stow” this would return “false”. For an object such as “Mars”, it would return “true”.

altAzCoordinates() : Returns the current position of the Source object, in Alt and Az values.

ScanPattern

A ScanPattern is a passive object, created to hold a definition of a set of movements an Array or Antenna is to make while observing a Source. The details of this are TBD.

CorrelatorConfiguration

setSpectrum(bw, pp) : Sets fields in the CorrelatorConfiguration object related to the spectrum to be observed.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 59 of 87

- ***bw*** is the bandwidth: “2GHz”, “1GHz”, “500MHz”, “250 MHz”, “125MHz”, “62.5MHz”, “31.25MHz”
- ***pp*** is the polarization products: “single”, “dual”, “full”
- **Defaults:** ***bw*** = “2GHz”, ***pp*** = “single”

For the prototype correlator, we have the following relationship between bandwidths, polarization, and total number of channels.

BBC	Bandwidth per Pol'n Prod. Products	Num. of Channels	Polarization Products	Oversampled
1	2 GHz	256	0-0	No
129	2 GHz	256	1-1	No
4	250 MHz	2048	0-0	No
132	250 MHz	2048	1-1	No
100	31.25 MHz	8192	0-0	Yes
228	31.25 MHz	8192	1-1	Yes
7	2 GHz	128	0-0, 1-1	No
10	250 MHz	1024	0-0, 1-1	No
101	31.25 MHz	4096	0-0, 1-1	Yes
13	2 GHz	64	0-0, 1-1, 0-1, 1-0	No
16	250 MHz	512	0-0, 1-1, 0-1, 1-0	No
102	31.25 MHz	2048	0-0, 1-1, 0-1, 1-0	Yes

Table 2 - Prototype correlator modes

setObservingTimes(dump, int, obs) : Sets fields for observing times.

- ***dump*** is the dump duration in seconds
- ***int*** seconds integration time
- ***obs*** seconds total observation time.
- *Note: ***dump*** and ***int*** must be divisible by 16 ms.*
- **Default values:** ***dump*** = 0.016, ***int*** = 10.0, ***obs*** = 1

setCAM(mode): Set the Correlator Accumulation Mode. ***mode*** can be 1 or 16 which represents the correlator chip accumulation mode of 1 millisecond or 16 milliseconds. If mode = 1, then on auto-correlation products are available and 1 millisecond dumps are possible. If mode = 16, then both auto- and cross-correlation products are available with a minimum dump duration of 16 milliseconds.

Default value: ***mode*** = 16

setBinMode(numberOfBins, binDurations) : Configure correlator binning.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 60 of 87

- ***numberOfBins*** can be 1 or 2.
- ***binDurations*** defines a list which specifies the number of fractional seconds in each bin.
- *Note that the sum of the bin durations must equal the dump duration specified in ***setObservingTimes()***.*
- **Default values:** ***numberOfBins*** = 1, ***binDurations*** = dump duration specified in ***setObservingTimes()***

setObservingFrequency(centerFrequenciesQuadrantIDs) : Assigns a center frequencies to quadrant IDs. centerFrequencies are in Hz and quadrant IDs are integers 0 – 3.

centerFrequenciesQuadrantIDs is a list of tuples, e.g., [(100000000000.0,0), (900000000000.0,1), (870000000000.0,2) (850000000000.0,3)]. This function also assigns the CorrelatorConfiguration to basebands (or correlator quadrants).

Default values: [(0.0, 0), (0.0, 1), (0.0, 2), (0.0, 3)]

setQuantizationCorrection(doQC) : Sets quantization correction depending on Boolean value of ***doQC***.

Default value: ***doQC*** = True

setFFT(doFFT) : Sets FFT mode, i.e., outputs raw lags or spectra depending on Boolean value of ***doFFT***.

Default value: ***doFFT*** = True

setAPC(APCDataSets) : Sets APC mode:

- ***APCDataSets*** = 0: supply one data set (APC-uncorrected),
- ***APCDataSets*** = 1: supply one data set (APC-corrected),
- ***APCDataSets*** = 2: supply two data sets (APC-corrected and APC-uncorrected)
- **Default value:** ***APCDataSets*** = 0

setWindowFunction(windowFunction) : Sets the data windowing function.

windowFunction can be: “Uniform”, “Hanning”, “Hamming”, “Bartlett”, “Blackman”, “Blackman-Harris”, “Welch”.

Default value: ***windowFunction*** = “Hanning”

setChannelAverageSubbands(subIntegrationDuration,subBands) : Set the channel average sub-integration duration and the channel average subbands. The sub-integration duration is in fractional seconds of which an integral number must fit into “int” specified in ***setObservingTimes()***. “subBands” is a list of tuples which specify the channel average subbands. The number of subbands must be at least 1 and not greater than 10. Each tuple must specify the starting channel (as a 0-based number) and the channel subband width. The sum of all of the channels in the subband must not exceed the maximum number of channels for the selected bandwidth and polarization show in Table 2. For example, 2



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 61 of 87

subbands which span channels 0 – 99 and 110 – 210 would be as such:
[(0,100),{110,100)].

Note: As more of the correlator configuration parameters are exposed and discussed between Correlator, ObsPrep, and Control, more methods will be added to this object.

Receiver

status() : Return the current status of the Receiver object.

readGains() : Read the current gain settings for the various stages of the Receiver and return them to the caller.

initBand(band, mode) : Initialize the cartridge and First LO for the band specified by the “band” parameter into the named “mode”. Mode is one of; “off”, “standby”, or “on”.

initIFSwitch(mode) : Set the IF switch to the specified “mode” which must be either “off” or “on”.

initLOSwitch(mode) : Sets the photonic switch to the specified mode; “on” or “off”.

setBandFrequency(band, freq) : Set the specified “band” to the frequency given by the “freq” parameter. This message may only be sent to a band that has already been set to “on”.

setAttenuator(number, value) : Set the appropriate attenuator (0 – 3) to the attenuation value specified by the second parameter. Legal values for attenuation value are 0 to -15dB.

setCoarseFrequency(freq) : Sets the DYTO frequency to coarsely tune the Second LO. Frequency is specified in Hz, with the valid range being from 8e9 to 14e9.

setLO2Range(tuning) : Sets the LO2 loop switch to determine if the FTS signal should be added or subtracted from the relevant comb line. The “tuning” parameter should be either “high” or “low”.

resetLO2Status() : Resets all bits in the status monitor point, ensuring that attempts to read status will always return current information.

getLO2Status() : Returns information on the logical status of signal levels and loop conditions in the LO2.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 62 of 87

setFTSFrequency(freq) : Set the Fine Tuning Synthesizer to “freq” Hz. Normal operating range for the FTS is 20e6 to 40e6.

setFTSPhaseVals(val1, val2, val3, val4) : This method sets a set of four values that are added to the output of the phase accumulator of the DDS. The phase values are switched at a rate of 125μs, in a sequence defined by the “setFTSPhaseSequence()” method. The first value specified (val1) will be used when both phase switching sequences are 0. The second value (val2) is used when fast switching is “1” and slow switching is “0”. The next (val3) is active when fast switching is “0” and slow switching is “1”. The final value (val4) is used when both switching values are “1”. All values are specified in degrees.

setFTSPhaseSequence(SequenceList) : This command sets up a list of 128 bits controlling the fast and slow switching sequences. The argument is a list of 16 bytes, with the MSB of the first byte being the first bit to be streamed and the LSB of the last byte being the last bit to be streamed.

setIFDCGains(gs1, gs2) : Set control lines for two digital attenuators in the IFDC. The gain values set the input power levels to the total power detectors. Each value may be in the range of 0 – 15 dB.

setIFBBGains(ga, gb, gc, gd) : Controls four digital attenuators in the IFBB. The values set in this method will affect the input power levels to the total power detectors. Each value may have a value between 0 and 16 dB.

setIFDCSBPaths(sab, scd) : This method controls signal paths between IFDC inputs S1/S2 and IFBB outputs A/B/C/D. If *sab* is “high” then outputs A and B will be connected to S1. If *sab* is “low”, the outputs will be connected to S2. Similarly, *scd* controls the path switching for outputs C and D to S1 and S2.

setIFDCFilterPaths(fa, fb, fc, fd) : The parameters in this method route each output base band channel through either the low frequency filter (4 – 8.5 GHz) or the high frequency filter (7.5 – 12 GHz). For each value, if it is “high” the corresponding output (A, B, C, or D) is routed to the high frequency filter. If “low”, the outputs are routed to the low frequency filter.

setTPSwitches(sw1, sw2, swa, swb, swc, swd) : This method controls the source to the S1, S2, A, B, C, and D total power detectors. If a value is “on”, then the corresponding switch is set to receive a live signal. If it is “off”, the signal is removed to allow measurement of zero-power offsets in total power data.

resetTPDigitizers() : Resets all digitizers and commands internal calibrations of total power detector.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 63 of 87

startTPContinuousConversion() : This method causes the total power detector to begin digitizing data until commanded to stop. This is the normal observing mode for the total power detector.

stopTPContinuousConversion() : This method ends a continuous conversion run, started by the startTPContinuousConversion() method.

Nutator

stow() : Locks the nutator subreflector at it's "0°" relative deflection position. The current nutator does not have hardware stow capability, so this command actually loads a program into the nutator, with two switching steps, both of which are set to 0° deflection. This program is then run to hold the nutator at the 0° position.

setStandbyProgram(NutatorProgram) : The nutator is capable of holding two program in memory at the same time. One is the "active" program; the other is the "standby" program. This method loads the NutatorProgram parameter into the nutator's "standby" position, without executing it or disturbing the current "active" program.

loadStandbyProgram() : This method moves the program which is currently in the "standby" position into the "active" position. It still does not execute any program steps.

startProgram(ACSTime) : Causes the current "active" program to be executed at the time specified by the ACSTime parameter.


stopProgram(ACSTime, step) : Stops the current program the first time it is at the specified "step", following the time specified by the ACSTime parameter. This method causes the nutator to pause at the specified step, with the nutator energized and the servos running, until another startProgram() method is executed.

abortProgram() : Immediately halts the current nutator program and de-energizes the nutator. After this command executes the subreflector is not controlled and may move in response to slewing or wind forces.

isRunning() : Returns a Boolean value indicating whether the nutator is currently running a program or not.

NutatorProgram

A NutatorProgram is a passive object. When it is created, it is given an array of values which represent the positions to be taken by the subreflector while the program runs.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 64 of 87</p>
--	---	--

6.2 Device Hierarchy

In keeping with the object-oriented paradigm of ALMA, the devices inherent in creating a functional Control subsystem have been organized into a hierarchy of devices. At the highest levels of this hierarchy are more abstract devices which may not even exist as physical constructs, but instead represent a collection of devices which, together, implement the functional behavior of the named device.

An example of this principle, taken from the diagrams that follow is the device referred to as an “Array”. In reality, there is no single device that is an “Array”. Instead, this device is a collection of devices which provides an interface to several antennas, allowing them to be treated as a single device.

Through this hierarchy, commands may be issued to high level, more abstract constructs and the user may rely on the system to translate those commands to a sequence of lower level commands to more concrete, physical devices within the system.


It should be noted that the complete ALMA system will have 64 antennas (not counting the compact array) at all times and may have from one to 64 subarrays in operation at any given time. In order to manage these resources in a manner that more closely reflects physical reality, we intend that both ArrayController and AntennaController objects be created as dynamic CORBA objects.

This means that there will not be a standard CDB entry for each antenna or for any subarrays. Instead, we will use the dynamic object creation methods available in ACS since release 3.0 to instantiate the CORBA objects as they are needed. We will also dispose of them when they are no longer in use. To improve the performance of this process slightly, we will not destroy an ArrayController object until the Scheduler has notified us that a particular array is no longer needed.

We believe that this will help us to make ALMA more responsive by not having extraneous objects loaded and running on containers when they are providing no useful services. It also means that we don’t have to worry about periodic bursts of status data coming from idle antennas.

6.2.1 Control System

The highest level of the Control subsystem device hierarchy is – appropriately enough – the “ControlSystem”, shown in Figure 12. This logical device in the subsystem represents the “master component” of our subsystem and is the primary interface to

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 65 of 87</p>
--	---	--

Control from other subsystems, such as the Correlator, Scheduling, and Executive subsystems.

Through this device, the Control subsystem may be started, initialized, stopped, and updated. It is also through this interface that the Scheduling subsystem obtains lists of available antennas and asks for subarrays to be created in order to execute Scheduling Blocks.

This device also monitors and controls the site video cameras, weather stations, and the master clock. As mentioned before, it is responsible for the creation of subarrays and for disposing of them when no longer needed. It also makes status information available to the Executive subsystem and provides the primary interface for controlling the Control subsystem.

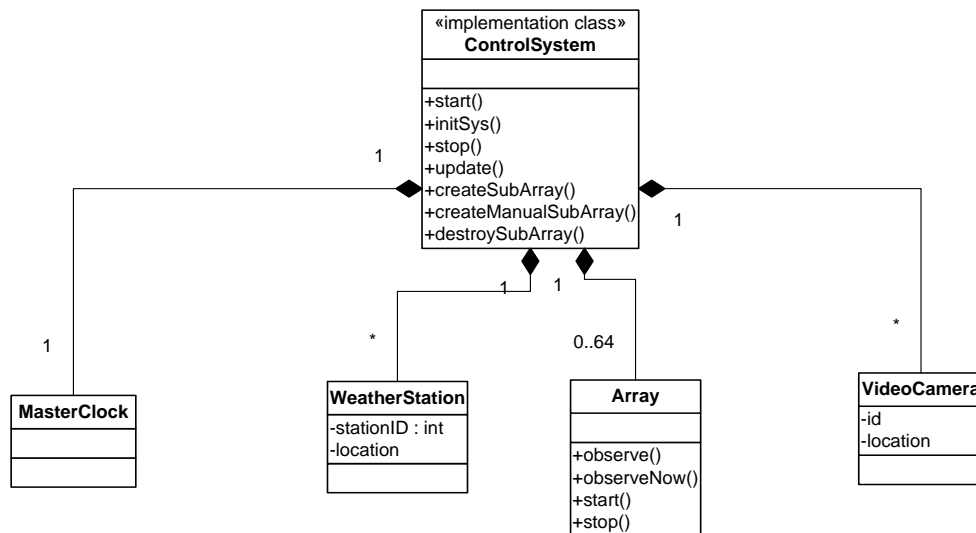



Figure 12 - Top Level of Control System Device Hierarchy

6.2.2 Array Hierarchy

The next level down in our hierarchy is illustrated in Figure 13. This figure shows the relationship between the Array object – represented in the Control subsystem by either an ArrayController object or a ManualArrayController object – and the components which provide its functionality. In normal operations, this will be the Scheduling subsystem’s primary interface to the Control subsystem.

When Scheduling selects a Schedule Block to be executed, the appropriate Array object is created and Scheduling asks the Array object to execute the Schedule Block. The Array object will be responsible for configuring the Correlator, tuning the receivers, and

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 66 of 87</p>
--	---	--

ensuring that the antennas that make up the array are aimed at the proper object and tracking properly.

During execution of the Schedule Block, the ArrayController (or ManualArrayController) issues several events that allow other subsystems to track progress of execution. In addition, monitor data relevant to the Schedule Block are collected and archived and the performance of the array is monitored for problems. Any problems which may affect the validity of the collected data are flagged so that the Telescope Calibration and Pipeline subsystems may take appropriate action.

The Array object also has access to the Control Command Language interpreter. This allows Schedule Blocks to include scripts to control an observation.

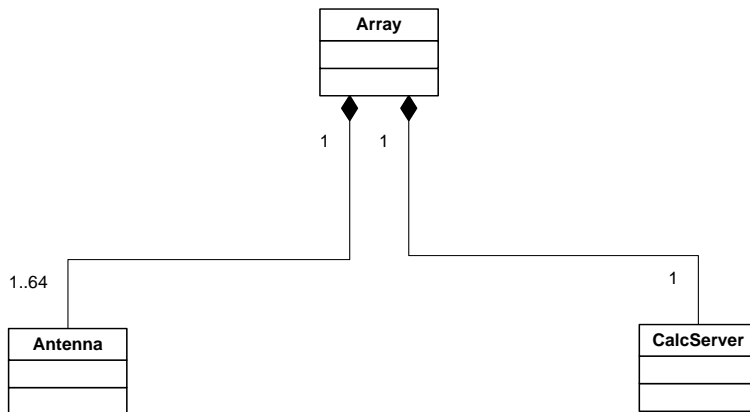


Figure 13 - Array Device Hierarchy

6.2.3 Antenna Device Hierarchy

Each Array object contains one or more Antenna objects. The hierarchy of an Antenna device is shown in Figure 14. Antennas are made up of a Receiver, a DTS Transmitter, and a Helium Compressor. The Antenna object is responsible for monitoring and controlling each of these devices and passing relevant data to its containing Array object.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 67 of 87

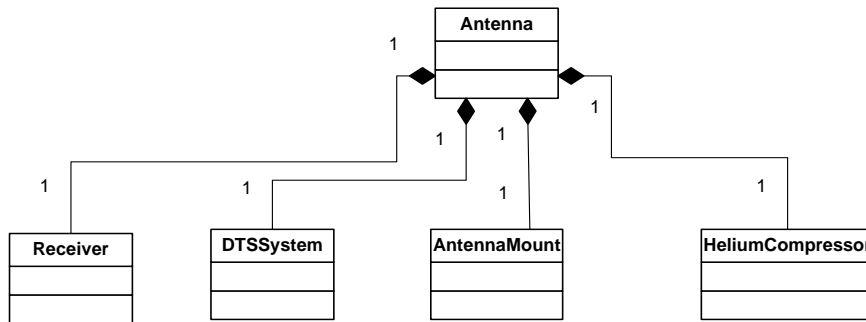


Figure 14 - Antenna Hierarchy

The device within an Antenna which does much of the work associated with collecting data is the Receiver object. Through the Antenna object, other components of the ALMA system have access to the receiver tuning as well as several pieces of monitoring data. For purposes of encapsulation, our ideal of a “receiver” includes several other components, which are detailed in the next section of this document.

Band	from (GHz)	to (GHz)	Input device type
1	31.3	45	HFET amplifier
2	67	90	HFET amplifier
3	84	116	SIS or HFET (TBD)
4	125	163	SIS mixer
5	163	211	SIS mixer
6	211	275	SIS mixer
7	275	370	SIS mixer
8	385	500	SIS mixer
9	602	720	SIS mixer
10	787	950	SIS mixer

Table 3 - ALMA Frequency Bands

6.2.4 Receiver Device Hierarchy

As mentioned previously, a great deal of the work done by the Antenna object is actually performed by the Receiver, contained within the Antenna.

For the ALMA Control subsystem, we consider a receiver to be made up of four separate devices:

1. Front-end electronics
2. Second Local Oscillator



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 68 of 87

3. IF Monitor and Control
4. IF Total Power Digitizer

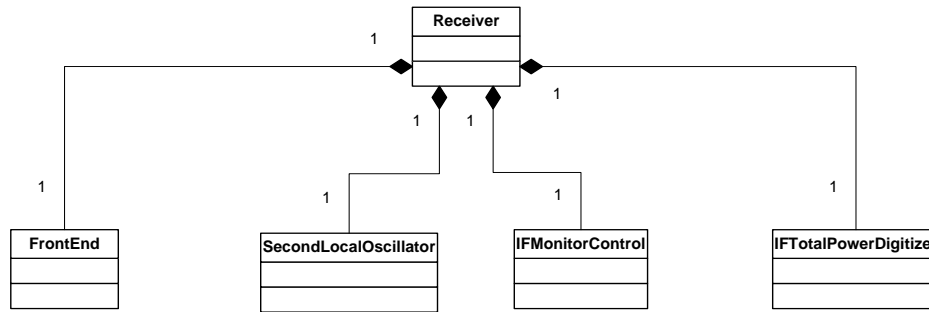


Figure 15 - Receiver Devices

This configuration is illustrated in Figure 15. The receiver is responsible for setting and holding the appropriate band for an observation, as well as allowing tuning within that band. The ultimate purpose of all of this is to provide an output data stream in a frequency and format that the rest of the system may utilize the results of an observation.

6.2.5 Front End Devices

Figure 16 is a diagram of the Front End of an ALMA receiver. The Front End, in the view of the Control Subsystem, consists of the First Local Oscillator, the Photonic Switch, and the IF Switch. Together, they are responsible for accepting the focused beam from the antenna's secondary reflector over a selected band of frequencies. The Front End then amplifies and converts this band to an intermediate frequency band in several channels, and delivers the IF signals as outputs.

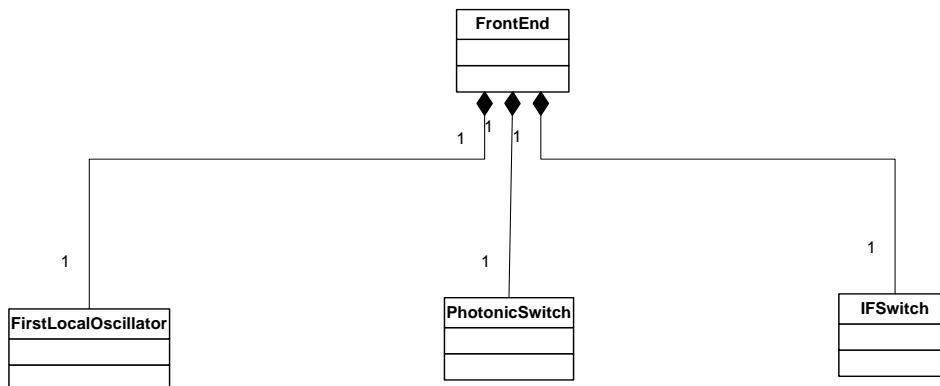



Figure 16 - Front End Detail

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 69 of 87
--	---	---

The ALMA front end subsystem will cover frequencies between 30 GHz and 950 GHz as given in Table 3 - ALMA Frequency Bands.

6.2.6 Second Local Oscillator

As shown in Figure 17, the Second LO is made up of two software-controllable components; a digitally tuned YIG Oscillator and a fine-tuning synthesizer (FTS). Each antenna actually contains four of these assemblies, each on a separate control channel of the AMB.

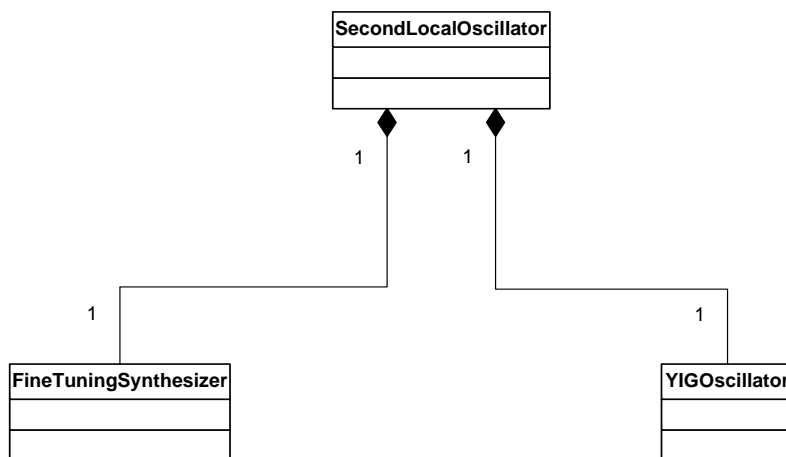



Figure 17 - Logical Structure of Second LO

In order to tune the Second LO, both the YIG Oscillator and the FTS must be adjusted. Coarse tuning is achieved by setting the YIG Oscillator to a frequency close to the desired final frequency of the LO. Fine tuning is then achieved by adjusting the FTS until the LO output frequency matches the desired frequency.

Normally, this tuning process is handled by the Control subsystem, transparently to the observer. However, the CCL will provide access to the Second LO, so that an experienced user may experiment with different tunings.

6.2.7 DTS Transmitter Hierarchy

Moving back up slightly in our hierarchy, we come to the DTS Transmitter, shown in Figure 18. The DTS Transmitter takes data coming out of the Receiver and transmits it,

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 70 of 87</p>
--	---	--

over a high-speed fiber link, to the Correlator for additional processing and signal conditioning.

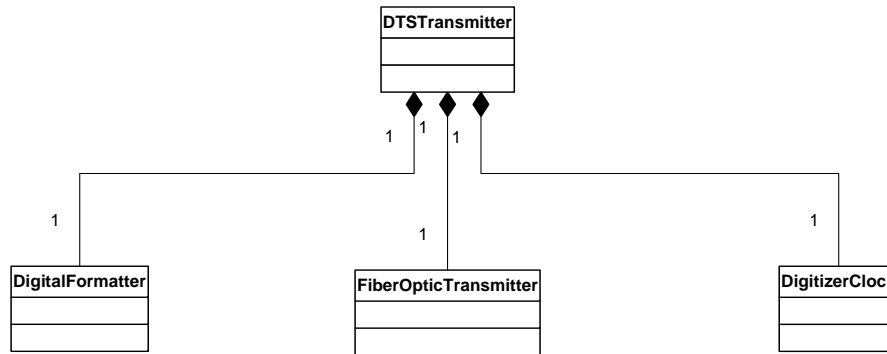



Figure 18 - DTS Structure

6.2.8 IF Monitor and Control

Also part of the Receiver is ALMA's IF Monitor and Control, depicted in Figure 19. This logical device within the Control subsystem provides for the control and monitoring of the ALMA Downconverter and the Baseband Processor.

The Down converter takes, as input, two channels from the Front End, each in the range of 4 – 12 GHz, mixes them with four channels of 8 – 14 GHz from the Second LOs, and outputs four channels of 2 – 4 GHz baseband signals, for processing by the Baseband Processor.

The Baseband Processor, in turn, processes the four channels of analog baseband signals and produces a total of eight channels of baseband signal. Four of those channels are passed to the IF Digitizer. The other four channels are sent to the IF Total Power Digitizer.

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 71 of 87
--	--	---

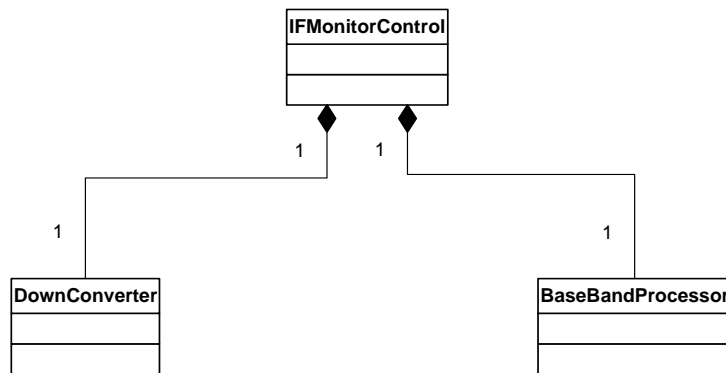


Figure 19 - IF Monitor and Control Hierarchy

6.3 Physical Device Management

One of the complexities of a system as large as ALMA has to do with device management. Given that there will be a large number of antennas, each with a collection of devices needed to make them function, how do you keep track of what receivers, etc. are in which antenna, without resorting to a cumbersome record-keeping system?

Our solution is to maintain a database of all AMB-connected devices, by serial number, but without regard to their physical location. When an antenna is brought on-line, the AMB Manager will query all connected devices for identification. Each device will answer with their node id and serial number. That serial number will then be used to look up the device in the database. This will allow us to know what type of device is connected as well as identify any device-specific tuning parameters that need to be applied to this device.

This also means that, if a device is moved for any reason, our boot-up process will take care of identifying where the device was moved to. No one will have to modify a table of device locations to account for the move.

6.4 Delay and Fringe Tracking

Delay tracking is implemented in three places in ALMA, in the station electronics, in digital delay lines before the correlator, and finally at sub sample resolution by varying the phase of the sampling clock. Each of these stages must be synchronized with the other stages and in total correct for the propagation delays across the array.

The CALC package, developed at NASA Goddard Spaceflight Center is a broadly recognized standard throughout the VLBI community. We have adopted a version of



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 72 of 87

CALC 9.1 which has been modified for use with the VLBA by Jon Benson. This engine is interfaced to the ALMA subsystem through a Java component, originally designed by Barry Clark for the EVLA and modified for our use in ALMA. Each subarray will create a “CalcServer” component. This centralized component maintains a list of the antennas within the subarray as well as the list of target sources. Well in advance (currently 10 s) the calc server calculates the delays every millisecond for the next period (again this is currently 10 s, but can be adjusted as necessary). These delays are quantized into the bulk delay (integer sample portion, to be applied at the correlator) and the fractional sample portion (to be applied as a phase shift at the digitizer clock). Only when the value of the delay changes is the value queued to be sent to the hardware device driver, in this way the required communication bandwidth is minimized. The individual device drivers are then responsible for applying the correction at the correct time.


The quantized values as well as periodic total delay values are sent via a notification channel to the correlator and antennas. The (unquantized) total delay values are used by the correlator to apply corrections in post processing and by the fringe tracking software.

Fringe tracking is done both in the first and second local oscillators. The total delay values are used to compute the phase and frequency of the local oscillators. Phase is computed both as a total, integrated value from an arbitrary starting time (to compute frequency to sufficient precision), and as the usual, modulo 2π value (to correct for quantization errors in the local oscillator signal generation). Because of hardware limitations, the corrections to local oscillator phase and frequency can be applied only once every 48 ms, which is nevertheless sufficient to meet specifications given the slow rate of change of the delay values. Fringe tracking computations are based on a continuous phase model that is independent of actual changes in observing frequency (at least, back to the arbitrary starting time mentioned earlier in this paragraph).

By using a central engine to generate the delays, the processing load on the antenna computers is reduced, and synchronization becomes a simpler problem. The tradeoff is the reliance on the Ethernet for transportation of this time critical data. In order to assure the quality of the data produced we have adopted a heartbeat mechanism, all data arriving from an antenna to the correlator is assumed to be erroneous and is flagged as such unless the data is asserted to be good. In this way network traffic and the ensuing tardy delivery of the delay information will not corrupt the final data product.

6.5 Real-Time Linux

For the implementation of ALMA’s real-time components in both the Control and Correlator subsystems, we have chosen to use RTAI – the Real Time Application Interface. This is a set of patches to the standard Linux kernel as well as kernel modules,

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 73 of 87</p>
--	---	--

libraries, and development tools. Together they provide several important capabilities needed by real time systems but lacking in a standard Linux distribution.

RTAI defines two types of processes: those running in kernel space and those running in user space. The user space processes are not real time and run in the standard Linux environment. The kernel space processes, operate at a lower level allowing deterministic response to hardware interrupts, the capability to assign priorities to processes, and greater control over scheduling. More information on RTAI is available at:

<http://mail.aero.polimi.it/~rtai/documentation/index.html>

Our design has been motivated by several requirements;

1. The number of modules running in kernel space should be minimal, both due to the difficulty inherent in development, and the instability induced by multiple real time processes running in the kernel space.
2. Existing code should be easily portable to the new system.
3. The only “hard real time” requirement is that CAN bus traffic must be able to be synchronized to the 48ms timing event, and monitor and control points must occur in the correct 24ms window of each event.

In order to conserve resources the Control group and the Correlator groups have combined efforts on this portion of the subsystem. The result is the CAMBServer which is designed to serve the needs of both the CMB and the AMB protocols. The CAMBServer is a kernel space process which processes requests for activity on either of the hardware bus architectures.

Two other tasks are running in kernel space, the teHandler task which receives the timing event interrupts and uses them to synchronize the system clock with array time, and the teScheduler which signals (through semaphores) the appropriate time for tasks to run.

In user space three high level interfaces to the CAMB server are provided; a device interface which allows requests for monitor and control points to be executed, a control interface which can be used to identify devices on the bus and control the reset lines, and a Manager interface which is designed as low level access for engineering and testing and is the union of the device and control interfaces.

The high level devices initiate a CAMB transaction by submitting a request to the interface. Requests contain the type of transaction, ancillary data as needed (such as the



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 74 of 87

data to be transmitted, or the location of the response data) and if necessary the time event with which the request is to be associated. In addition two optional methods of notification are provided, a semaphore may be supplied which will be signaled upon completion or the file descriptor of a FIFO may be passed, and the results of the request will be written to the file descriptor.

The interface in turn passes this information to the real time CAMBServer over a real time FIFO. The server maintains two queues for each CAMB channel, one for requests associated with a TE and one for requests to be processed on an “as soon as possible” (ASAP) basis. Each of these queues is serviced by a thread. The threads which service the TE queues run at a high priority, but are only woken on the 48 ms timing event and 24 ms afterwards (this is accomplished by a semaphore signaled by the teScheduler subsystem). The ASAP threads run whenever there is data in the queue to be processed and no higher priority task is running. In both cases upon completion of the request, the notification of completion is sent via a real time FIFO to the interface task in user space.

This design allows each of the CAMB channels to operate independently so a failure or delayed response on one channel will not impact the processes running on the other channels.

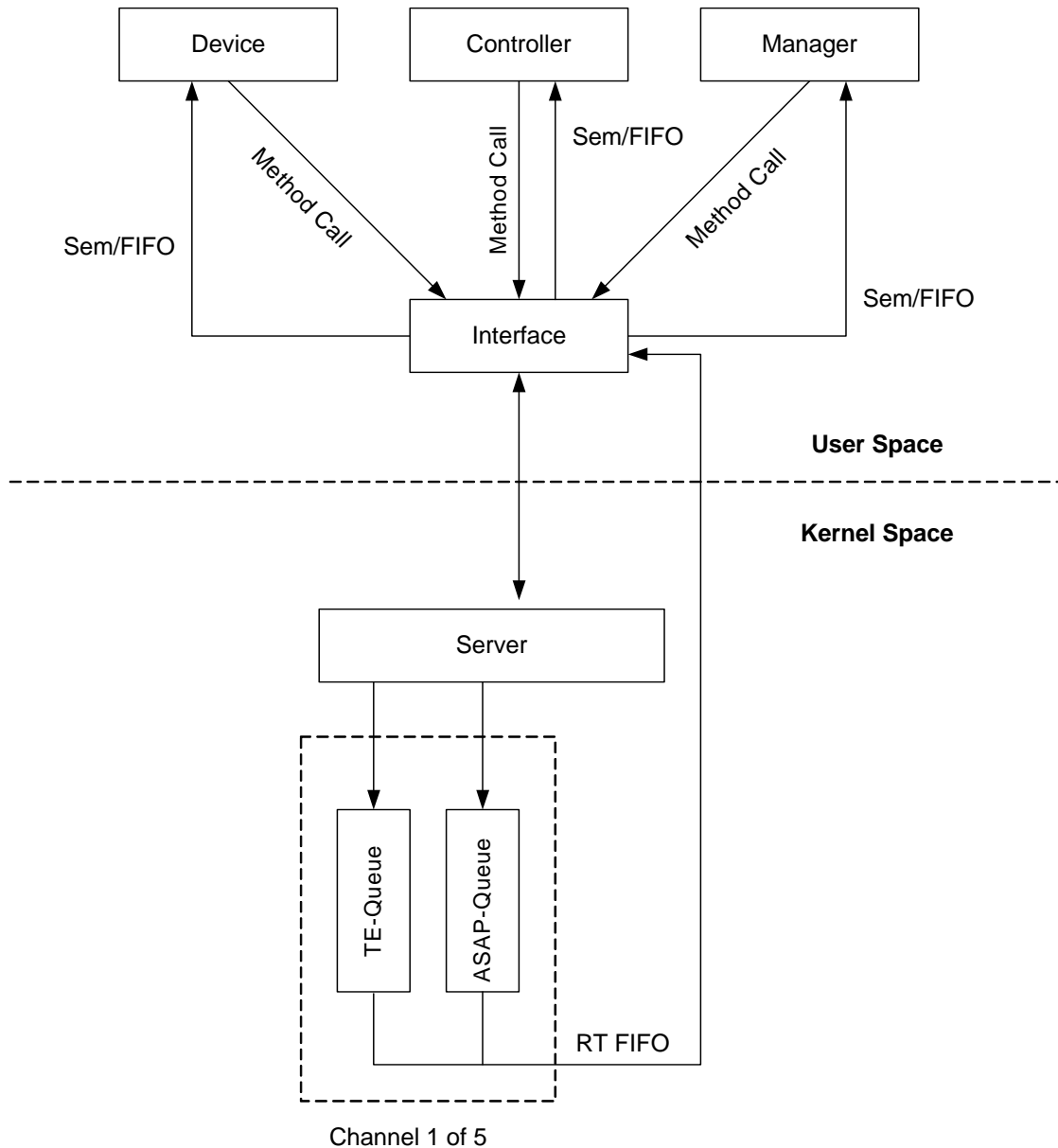


Figure 20- CAMB Design

6.6 Monitor Point Organization

The monitor point is a simple entity. It contains diagnostic and state information collected from individual hardware devices. It will be collected, collated and interpreted by a package within the control subsystem and sent to the archive.

Because of the scale of ALMA i.e., 64 antennas, there will be a lot of monitor points and an initial estimate comes to around 50,000 (this is probably accurate to within a factor of two). Using



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 76 of 87

existing documents that describe prototype ALMA hardware and the suggested sample rate for the monitor points, along with recent discussions, we can estimate that the average data rate is 2 bytes/second/monitor-point. If we further assume that no monitor point will be sample faster than twice per second, we derive a raw monitor point data rate of approximately 100 Kbytes/second, before adding the overhead of necessary bookkeeping.

Each monitor point, to be useful needs to have a name and a time associated with it. Simple XML schemes for attaching this data to each monitor point can add a large overhead to the raw monitor rate (up to a factor of ten). To avoid this, the monitor point data will be encoded in VOTables with separate tables for different data types and different antennas. This allows us to reduce the overhead, per monitor point to four bytes, two for the time and two for the monitor point name. By using the binary tables option of VOTables the raw monitor point data does not need to be converted to text and hence this does not affect the monitor point rate.

So for each computer connected to hardware i.e., each ABM and the ARTM the monitor points will be represented using a VOTable with header items:

- BaseTime – A time to be added to all time offsets in the table This will be an XML Date string eg., 2007-10-24T13:24:12
- CollectorId – An identifier for the computer that collected this data. This will be a 16-bit integer eg, 24.
- Type – The data type represented in this table. Different tables must be written for each data type and possible values include ‘D’ for 8 byte floating point, ‘S’, for string values and ‘E’ for enumerated data types.

There will be three of these tables, one per data type created per collector and 65 collectors (one in each antenna and a central one) for a total of 195 tables. These tables will be written to the archive once per minute. This leads to an overall data rate (assuming 8 byte strings and an equal distribution of monitor point data types) of approximately 200 Kbytes/sec, including the overhead of time stamps etc.

Name	Description	Data type & Size
TimeOffset	Monitor point sample time in milliseconds to be added to the BaseTime	Short Integer (2 bytes)
MonitorId	A identifier that when concatenated with the CollectorId provides a lookup into a table that describes this monitor point	Short Integer (2 bytes)
Data	The monitor point data	Double, String, or Byte (8/*1 bytes)

Table 4 - VOTable Record Format


	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 77 of 87</p>
--	---	--

Table 4 shows the format of a record within a VOTable. Each record will represent one sample from one monitor point and will contain the three fields described in the table. As stated above, a group of records will be written to the archive about once per minute.

6.7 Time Synchronization

The control subsystem will have over 66 computers (64 ABM's, 1 ARTM and 1 ACC) running and accurately synchronizing the time of them is critical to coordinating the hardware and ensuring that observations made are scientifically useful. The ALMA Control subsystem will use a time synchronization scheme based on that currently used at the ALMA Test Facility (ATF).

There will be two separate time distribution mechanisms. The first of these is what used in the majority of desktop computers namely the Network Time Protocol (NTP). This protocol allows geographically distributed computers to synchronize the time to a master computer and statistically is able to determine the propagation delays between computers. Computers that are master time servers are attached to a high accuracy clock and many are available, on the Internet, for general usage. This mechanism will be used to synchronize the time all non-real-time computers. In the control subsystem this is only the ACC.

There are some problems with using NTP in the real-time computers. One is the jitter in the actual time that occurs as statistical determinations of the propagation delay are applied. We have measured the jitter as being about a few milliseconds. The other is the time it takes for the time to synchronize. Because a statistical estimate of the propagation delay needs to be determined the time on the client computer will take many minutes before it converges on a good estimate.

To overcome these problems will use a different, home grown protocol to synchronize the time on the real-time computers. This will be based around the Timing Event, an electronic signal that is distributed to all hardware that requires accurate time synchronization. This timing will also be connected to all real-time computers and is accurately derived from a Maser that is the master timekeeper for all ALMA equipment.

Except for initialization the protocol is rather simple, the real-time computer is told the time associated with a specific timing event and it then maintains time by counting timing events and ensuring that its internal clock increments by exactly 48ms every timing event.

The only tricky bit is the initialization. To do this there will be one computer (the ARTM) that is directly connected to an electronic source of time (GPS). This computer will get its time initially from this device and then count 48ms pulses to maintain its time. The other



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 78 of 87

real-time computers will, when commanded to do so by the ACC, contact the ARTM and request the time at the next timing event. The response to this request must be received within 48ms to ensure that the correct timing event is used. The remote computer will time the length of the transaction and, if it's longer than 48ms, try again (for a preset number of times). The remote computer always knows if it has succeeded in synchronizing its time and if it does not succeed and hence can notify the ABM as to its status.

6.8 Simulation

The development schedules of control software and the hardware it controls are intimately tied. Testing of control software cannot be completed without access to the relevant hardware. It is during this testing that discrepancies in the interpretation in the interface, timing problems, memory leaks and numerous other problems are discovered and corrected. This coupling can often lead to delays. In addition control software will often need to coordinate the operation of numerous pieces of electronics and testing the software that does this will require simultaneous access to all the relevant hardware. Scheduling this can be difficult particularly as ALMA development is distributed over numerous sites on two continents.

These concerns can somewhat be addressed using simulators. A simulator is a surrogate for the actual equipment and performs, from a control software perspective, in a similar fashion. There are numerous approaches to simulation and as part of the TICS development we have used two.

Both our approaches are software simulations. In the first we write software that has an interface that mimics the interface presented by the hardware. Typically this software runs on a separate computer that has a CAN interface card. It listens to monitor and control requests on the AMB and reacts by responding to monitor requests and changing its state in response to control requests. There is one simulator for each hardware device (although we did not write simulators for every hardware device).

This approach has some advantages and some drawbacks. By using the same physical bus we test the entire call stack including the driver software. With a perfect simulator the control software would not know if it was communicating with the actual hardware or not. In addition we can use the simulator to produce failure modes that are rare or difficult to produce using the actual hardware.

However our simulators are not perfect and this is because we must trade off the time spent writing a simulator against the time saved during testing. Our simulator runs on a general purpose computer running Linux and hence cannot meet the real-time deadlines that the actual hardware would. Hence we relax the timeout parameters when test using a



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 79 of 87

simulator. Our simulator does not use the "timing event" signals on the AMB to synchronize its operation and the AMB reset signal does not reboot the simulator. More generally our simulations are not accurate because we do not mimic the precise behavior of all aspects of the electronics. Instead we concentrate on the "important" behavior and sometimes an aspect of a seemingly unimportant detail is overlooked.

Another problem with this level of simulation is that it requires specialized hardware, in particular access to a real-time computer, another computer with a CAN interface card and a cable connecting the two. This equipment is not available to all our developers, and even if it were, it would not be easily transportable. Simulation is quite valuable if it is integral to the implement-compile-test development cycle.

To address this we can use the same simulation software but bypass the CAN driver layer so that the control software communicates directly with the simulator. This allows us to run the entire system (control software and the simulator) on a general purpose development machine or even a laptop. A side benefit is that this is useful for demonstrations! One problem with our current implementation is that it requires the control system to be linked against the simulator libraries. We are currently working on a design that will remove this compile time dependency.

Another issue we have encountered with simulators occurs when the same developer writes both the simulator and the relevant part of the control software. If a mistake is made in two places it is possible that both errors will cancel each other out. Finally our experience has been that trivial simulators are not worth it. Simulators for trivial devices like power supplies rarely get used but simulators for the complex equipment like the antenna mount are invaluable.

7 Lessons from TICS

Many of the design concepts presented earlier in this document were prototyped in the Test Interferometer Control System (TICS), which was developed for use at the ATF. While primarily designed to meet the requirements of the Antenna Evaluation Group, it was also seen as an opportunity to gain experience with ACS in conjunction with much of the hardware that would ultimately make up the ALMA Control subsystem.

7.1 Software Infrastructure

The control system at the ATF is based on the ALMA Common Software (ACS). The ACS middle-ware provides a component/container structure using the Common Object Request Broker Architecture (CORBA) to connect the distributed system. Communication and lifecycle management are controlled by a Manager component.



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 80 of 87

In the implementation used at the ATF, the Manager was located on the ACC computer, and each of the ABM's had a single container running. Hardware devices (such as total power detectors or the optical pointing telescope) were implemented as components, running within the containers on each ABM.

User access to the modules is predominately through Python clients which connect to the CORBA backbone of the ACS system. The flexibility and short development cycle of Python allowed us to rapidly respond to requests from the antenna evaluation team.


7.2 Software Design

At the lowest level of TICS we have implemented a server process running on a real-time computer to control access to the AMB. This server handles the AMB CAN protocol, queues requests for CAN transactions on the AMB, and returns transaction results to the caller of each request. Transaction timeouts are handled at this level, while the timing of real-time transactions is handled by clients on the real-time computers.

The “device controllers” exist at the next higher level of TICS; these also run on the real-time computers, and are implemented in C++. The device controllers are software representations of the physical devices, and implement a CORBA interface for a hardware device. Device controller methods typically implement high-level commands (which may require coordinated access to several monitor and control points of the hardware), and also provide client access to “timing event associated” monitor and control points (which require time tags as a parameter.) The device controllers are used within the control system, and may be exposed in some way to users, but they are not typically used in engineering development and testing of the hardware (for that, the lower level AMB interface is used).

Device controllers also have “properties”, an ACS concept, which for the most part, correspond to the hardware device monitor and control points. Such properties use typed values in the interface, so that access to monitor and control values through properties is standardized and based on SI units wherever possible. The conversion of typed values from/to integer values or bit-fields for communication with the hardware, as well as the access to the hardware interface (usually the AMB) is done through subclasses of the “DevIO” class, which implements a bridge pattern. Properties also implement CORBA interfaces, which are independent of the device with which they are associated.

At the highest level, device access occurs through CORBA clients of the device controller and property servants. These clients are typically written in C++ and Python, and, in the case of Python, are used in user-level scripts. Non-real-time algorithms may be implemented at this level: for example, receiver tuning, and computation of antenna motion patterns.

	<p>ALMA Project</p> <p>Title: Control Subsystem – Design Document</p>	<p>Doc # : COMP-70.35.00.00-004-D-DSN</p> <p>Date: 2004-06-17</p> <p>Status: Pending</p> <p>Page: 81 of 87</p>
--	---	--

7.3 Experiences

Our experiences at the ATF have tested our designs and shown both the strengths and weaknesses of our design. Several issues have broad application and are highlighted here.

7.3.1 Interface Control Documents

Interface control documents (ICDs) define the interface between various subsystems in ALMA. In particular, an ICD defines the interface between the hardware devices and the control system, and ICD's were developed for all devices with a control system interface. ICDs were to have been written by hardware engineers in collaboration with members of the ALMA control subsystem group.

In practice, our experience using ICDs was decidedly mixed. When the ICDs were complete and accurate, they were very useful to control system developers; integration testing and commissioning times were reduced in such instances (in comparison to the following situation). In those cases where the ICDs were either incomplete or inaccurate, they became the source of several problems encountered during integration testing and commissioning, and would occasionally become a source of contention between hardware and computing groups.

A home-grown AMB bus analyzer (known as the “canalyzer”) was indispensable in determining which messages were on the wire, their order and their timing. This tool provided indisputable data on AMB activity, and was crucial in determining whether a problem was in the hardware or the way it was being controlled.

7.3.2 Python for Rapid Prototyping

A set of high level commands was developed in Python. These commands provide a high-level interface to the control system, and are intended to be used in scripts written by the observers. Several high-level algorithms were implemented directly in Python, which allowed for rapid prototyping and implementation changes in those cases where the algorithm was not well understood prior to deployment at the ATF.

7.3.3 Check the Cabling First

Whenever problems arose at the ATF, it became prudent on the part of the control subsystem team members to first check cabling, power switches, connectors and terminators. In a dynamic environment like the ATF, where numerous people from different groups are at work, arriving and departing at all times, hardware is repaired or



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 82 of 87

replaced without warning, and the unexpected is commonplace, it pays to double-check the physical components before looking for problems in the software or the computer configuration. Many frustrating hours of debugging can be avoided by making these simple checks.

7.3.4 Provide Software Bypass Mechanisms

In tracking down problems with devices on the AMB, it is always useful to see the data at the level of the CAN messages. Directly verifying the format of CAN message payloads eliminates layers of software between the highest and lowest levels by bypassing the device controller code. Thus, unexpected data in a monitor value or byte-ordering problems by either the hardware device or the device controller can be readily identified. Although the “canalyzer” could be used in such instances, it is much more common to use a CORBA interface to the AMB server process that we developed and named the “CAN Manager”. Using the CAN Manager from Python scripts provides a quick and easy way to do such testing.

7.3.5 Provide a Remote Reset

At the ATF, we have deployed power strips with a TCP/IP interface to allow us to power cycle the ABMs and ARTM remotely. This has proven to be very useful in supporting ATF operations remotely, as well as on site. For example, since the receiver cabins can only be entered when the antennas are pointing toward zenith, resetting the ABMs using the power strip avoids manually moving the antennas to zenith and the time which would be required to enter the cabin and reset the ABM manually.

The AMB also implements a bus-wide reset function, which is often used, and has been the only escape from a non-functioning system in many cases. While the AMB reset and network-connected power strip are both very useful for remote support, a finer-grained control for resetting parts of the control system is desirable. The inability to reset specific system components greatly adds to the duration of a debugging cycle, and impedes efficient commissioning of the system.

7.3.6 Enforce Hardware Interface Standards

While the majority of devices at the ATF implement an AMB interface for monitor and control, some do not. These exceptional devices have placed disproportionate demands on the control system developers, and never achieve the level of integration into the control system that conforming devices have. Additionally, some devices have other interfaces in addition to the AMB interface. The “extra” interfaces are typically designed by the hardware engineers for use in the lab, but inevitably are used on site as well (at least at times). Although there is no integration of these interfaces into the control system,



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN

Date: 2004-06-17

Status: Pending

Page: 83 of 87

their requirements place additional demands on the maintenance and use of network resources. The lesson here is simple: insist on standard hardware interfaces only for device monitoring and control.

7.3.7 Plan for a Significant Support Load

Support of operations at the ATF required one developer on a full-time basis. The prototypical nature of the antennas, devices, software and operating procedures at the ATF kept the support person very busy and the load often spilled onto other members of the ALMA control software development team. Allocating one person to take the brunt of this support load has allowed other developers to concentrate on the long term software development required for ALMA.

7.3.8 Real Time Operating System

The heart of a telescope control system is the real time operating system (RTOS). At the ATF the ABMs run using the VxWorks real-time operating system from Wind River. At the time of the initial decision, significant experience with VxWorks existed within both NRAO and ESO. Additionally at that point the real-time Linux operating systems were judged to be immature products with all of the associated liabilities.

In 2003 this decision was revisited, and a number of issues with VxWorks were identified. Maintaining two complete sets of compilers adds cost and complexity to the development system. Non-real time applications are developed on Intel based Linux systems. The need to maintain separate Solaris systems specifically for cross compilation added significant overhead to the development environment.

Cross-compilation was found to be very time consuming and the time even simple libraries took to compile was found to be distressing. This problem is a combination of the slower Solaris computers and the need to use the ACS middle-ware. The VxWorks compiler was also found to lag the developments in C++ standards and for most of the TICS development features such as namespaces and exception handling were absent. Compensating for these omissions was difficult and lead to complex code, not only in the real time modules but throughout the Control system.

Although VxWorks is a widely used platform for real time development, porting of tools (such as the gnu C++ compilers, or the TAO ORB used by ACS) lags the forefront of development. The Tornado suite of development tools could not be used by us because of compatibility issues.

The above reasons and the increasing stability of real-time Linux operating systems led to a decision to attempt porting portions of the control system to a real-time Linux operating



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN


Date: 2004-06-17

Status: Pending

Page: 84 of 87

system. In October 2003 we successfully demonstrated control of the telescope mount using real-time Linux. With this success the decision was made to switch the RTOS to real-time Linux.

In addition to addressing many of the concerns discussed above, the real-time Linux implementation selected by ALMA provides an additional feature. In the real-time implementation selected by ALMA the standard Linux kernel runs as the lowest priority task in a simpler, real time kernel. Thus there is a strong decoupling between the real time tasks, which run as kernel code, and the non-real time tasks which run in a standard Linux environment.

	ALMA Project Title: Control Subsystem – Design Document	Doc # : COMP-70.35.00.00-004-D-DSN Date: 2004-06-17 Status: Pending Page: 85 of 87
--	---	---

8 Appendix

8.1 CCL Program Examples

Below are two simple examples of what an ALMA observing script might look like. Each example creates a source for the array to be aimed at, directs the array to aim in the chosen direction, sets the intended observing frequency, bandwidth, etc, and sets up the desired integration and observation times. They then direct the array to begin collecting data.

The significant difference between the two examples is that the first is completely synchronous, with the script waiting for methods to complete executing before continuing. The second example takes advantage of Python's threading capabilities to allow commands to execute while long-running processes continue in the background.

As the specifications of the CCL are refined, more detailed examples will be developed to demonstrate additional features of the language. For right now, we feel that getting this document out for comment is of higher priority.

8.1.1 Example 1 – Busy/Wait Synchronization

```
# Sample ALMA observing script - 1
#
# This example is a trivial one that does not take advantage of
# threading or synchronization. Instead, it uses busy-waiting
# to wait for the antennas to arrive at the correct orientation.
import time
def slewWait(tolerance):
    inTolerance = False
    while inTolerance != True:
        errors = myArray.getPointingError()
        for val in errors:
            if val < tolerance:
                inTolerance = True
            else:
                inTolerance = False
#Get a pointing reference for Mars
planet = Source("mars")
#Move antennas in array to position and begin tracking
myArray.setDirection(planet)
#Configure correlator for observing frequency of 40 GHz,
```



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 86 of 87

```
# bandwidth of 1 Ghz, 2 Polarization Products, and
# 256 data Channels
CorrelatorConfiguration config1
myArray.setSkyFrequency(40e9)
config1.setCorrelatorSpectrum(1e9, 2, 256)
#Configure correlator for 10 second integrations over
# an observation period of 3 minutes
config1.setObservingTimes(10, 180)
#Send the configuration data to the correlator
myArray.configureCorrelator(config1)
#Check the antennas are on source
slewWait(1/60);
#Mark the beginning of a scan
myArray.startScan(FOCUS)
#Begin taking data asap
myArray.observe()
#Mark end of scan
myArray.endScan()
calData = myArray.getTelCalResults()
```

8.1.2 Example 2 – Threads and Synchronization

```
# Sample ALMA observing script - 2
#
# This example is somewhat more complex. It defines a simple
# function to be used as a thread so that the script can wait
# for the antennas to be pointed at our source, within a level
# of tolerance that we set.
import thread, time
def slewWait(tolerance):
    inTolerance = False
    while inTolerance != True:
        errors = myArray.getPointingError()
        for val in errors:
            if val < tolerance:
                inTolerance = True
            else:
                inTolerance = False
    def checkLocked():
        isLocked = False
        while isLocked == False:
            status = myArray.isLocked()
```



ALMA Project

Title: Control Subsystem –
Design Document

Doc # : COMP-70.35.00.00-004-D-DSN
Date: 2004-06-17
Status: Pending
Page: 87 of 87

```
isLocked = status[0]
for val in status:
    isLocked = isLocked and val
#Get a pointing reference for Mars
planet = Source("mars")
#Move antennas in array to position and begin tracking
myArray.setDirection(planet)
slewThread = thread.start_new_thread(slewWait, (1/60,))
#Configure correlator for observing frequency of 40 GHz,
# bandwidth of 1 Ghz, 2 Polarization Products, and
# 256 data Channels
CorrelatorConfiguration config1
myArray.setSkyFrequency(40e9)
config1.setCorrelatorSpectrum(1e9, 2, 256)
lockedThread = thread.start_new_thread(checkLocked)
#Configure correlator for 10 second integrations over
# an observation period of 3 minutes
config1.setObservingTimes(10, 180)
#Send the configuration data to the correlator
myArray.configureCorrelator(config1)
#Make sure the antennas have finished slewing and that the LO is
# locked.
slewThread.join()
lockedThread.join()
#Mark the beginning of a scan
myArray.startScan(FOCUS)

#Begin taking data now
myArray.observe()
#Mark end of scan
myArray.endScan()
calData = myArray.getTelCalResults()
```