EVLA Monitor and Control Communications Infrastructure

Version 1.0.2

Table of Contents

Version/Revision Information

Revision	Date	Author(s)	Description of Changes
1.0.0	January 31, 2005	Bill Sahr	Original version
1.0.1	February 11,2005	Bill Sahr	Revisions based on initial, limited circulation.
1.0.2	February 16, 2005	Bill Sahr	Section on UDP and multicasts added.

1 **Executive Summary**

This document is the report of the EVLA Distributed Object Communications Team. The team was formed on May 14, 2004, and concluded its work early in September 2004. The purpose of the team was to characterize and analyze the nature of the monitor and control information within the EVLA Monitor and Control System, and to recommend an approach to the distribution of this information within the system. The bulk of this document is devoted to the attempt to characterize the information to be communicated. The data inputs and outputs for a collection of core components are examined. These components include the controllers for subsystems in both EVLA and VLA antennas (MIBs and the CMP), the Observation Executor and Antennas as software entities (Antenna Objects), the Correlator and the Correlator backend, Antsol/Telcal, and Operator Screens. A possible deployment scenario for the EVLA is reviewed. At the conclusion of the characterization of the information, two possible approaches to the distribution of the information are examined – distributed objects and messaging. An analysis of aspects of the hardware architecture of the EVLA, of the nature of the information to be distributed, and of the requirements of the plan for making the transition from the VLA to the EVLA leads to the conclusion that a message-based approach is best suited to the needs of the EVLA Monitor and Control System.

2 **Introduction**

On May 14, 2004 a team was formed to consider the means by which distributed objects within the EVLA software system would communicate. The team consisted of 6 members drawn from the staff of the EVLA Computing Division (ECD). The team members were:

- Chunai Cai (Executor software)
- Rich Moeser (Java expertise, User Interfaces)
- Kevin Ryan (Antenna Monitor and Control)
- Bill Sahr (EVLA M&C Group Leader)
- Boyd Waters (e2e Software, Deployment Scenarios)
- Pete Whiteis (MIB Software)

Team members were chosen on the basis of their nominal areas of responsibility and expertise (given in parenthesis). The charge to the team was to characterize the data and information that is to be distributed within the EVLA Monitor and Control System, and to recommend an approach to communications that will satisfy the needs of the system.

3 **Characterization of Communications**

The team began by identifying processing "loci" that capture the essential elements of the EVLA Monitor and Control system. The following list was adopted as the focus for characterization of communication within the monitor and control system:

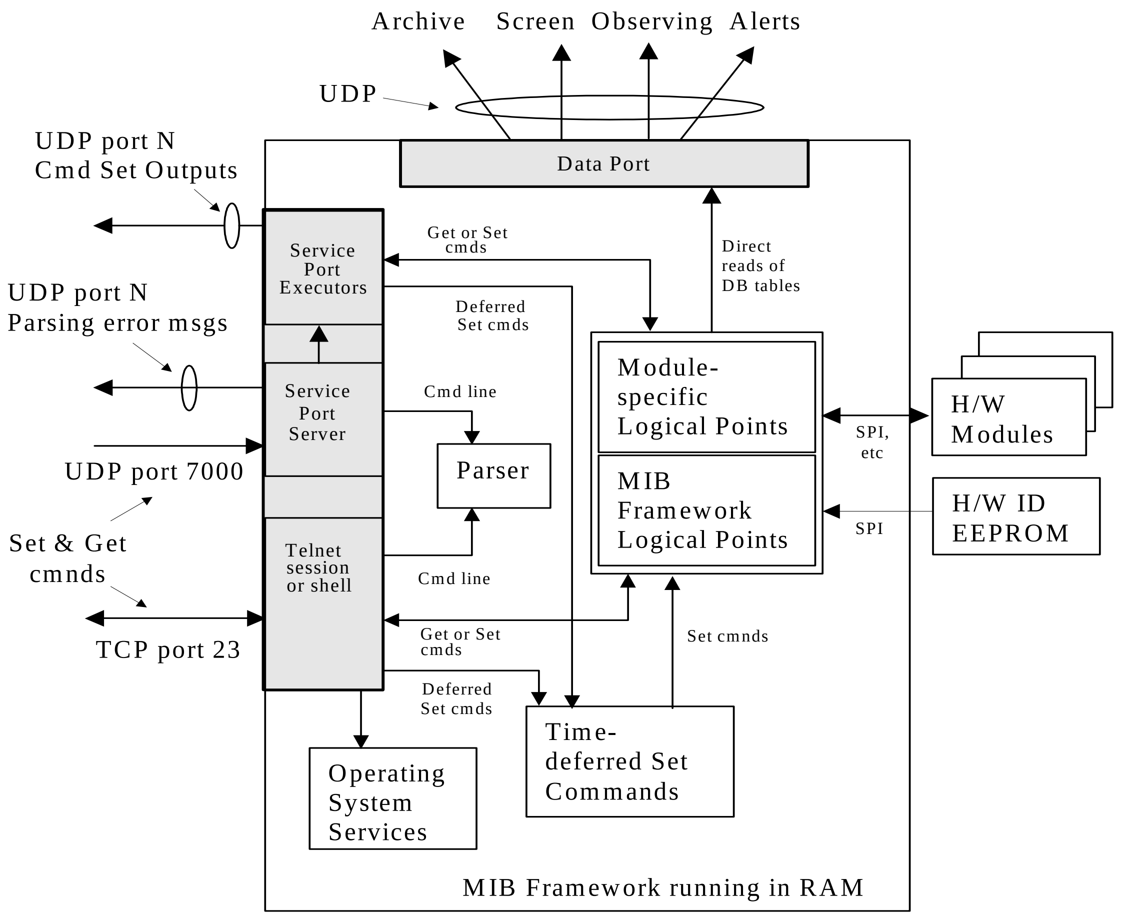
- MIBs & a collection of MIBs sufficient to characterize an antenna
- the CMP
- the Observation Executor
- Antennas, as software entities

- Correlator and Correlator Backend configuration
- Antsol/Telcal
- Operator Screens
- Deployment Scenarios

This list was not meant to be exhaustive or final, but it was agreed that an examination of the communication needs of these elements should capture both the bulk and the essence of communication requirements within the monitor and control system.

3.1 MIBs & Collections of MIBs

The figure given below is a diagram of the MIB software.



MIB Software Diagram

The internal structure of the MIB software is not of interest to the purposes of this document. The concern here is with the ports. The MIB has two ports – a service port and a data port. The service port is bidirectional, and there are two paths by which a connection may be established – a UDP-based connection that is used by the device browser and by software processes that perform operational activities, and a TCP/IP-based Telnet port that is intended for direct human interaction with the MIB and MIB-controlled devices. The same command and response syntax is used for both types of connection to the service port. Telnet sessions are limited to two simultaneous sessions per MIB.

Set and get commands are received via the service port of the MIB. Set commands are used to set the values of control points, i.e., commands to be acted upon by the hardware device(s) controlled by the MIB, and to set the value of monitor point attributes, such as the value of the high alarm and low alarm attributes that determine when a monitor point is considered to be out-of-range. Get commands are used to obtain the values of monitor points, and to obtain descriptions of the available monitor points and control points.

The data port of the MIB is a unidirectional port – output only. Monitor data, alarms and alerts, error messages, and logging messages are sent via the data port as multicasts. Three monitor data streams will be implemented (as multicast groups) on the data port – one for archiving, one for screens, and one for software processes performing operational activities.

3.1.1 MIBs, Data In

- Set commands received via the service port
    - from software processes
        - scan configurations
        - position updates
    - directly from operator screens
    - from the device browser
    - from command line interface (CLI) based telnet sessions
        - telnet sessions are limited to two per MIB
- Get commands received via the service port
    - from software processes for the purpose of obtaining a coherent, current picture of subsystem state and status
    - from operator screens
    - from the device browser
    - from CLI-based telnet sessions
- code loading, ~ 1 Mbyte per image (infrequent)
- XML device configuration files (~ 5Kbytes to 20Kbytes, infrequent)

3.1.2 MIBS, Data Out

- responses to set commands when the -v (verbose) option is set (service port)
    - to software processes
    - to operator screens
    - to the device browser
    - to telnet sessions

- responses to get commands (MIB service port)
    - used to obtain monitor data on request, and to obtain a a description of available control points and monitor points and their attributes
    - to software processes for the purpose of obtaining a picture of the subsystem state and status
    - to operator screens

    - to the device browser
    - to telnet sessions

- monitor data from the MIB data port
    - periodic multicast
    - average size of a monitor point is ~ 60 bytes
    - average number of monitor points per UDP packet is ~ 6.8
    - to the archive
    - to software processes such as the Executor and to support functionality such as flagging
    - to operator screens

- alarms and alerts (MIB data port)
    - event driven multicast messages
    - to software processes
    - to operator screens
    - to the device browser
    - to telnet sessions
    - to an archive
    - for Checker functionality
    - for Flagging functionality

- error messages, logging messages (MIB data port)
    - event driven multicast messages
    - may also be periodic if message generation embedded in a periodic task
    - generated by MIB processes
    - current maximum length is < 300 bytes Possible future enhancements may increase the maximum size to the payload of one Ethernet packet, 1280 bytes.
    - to software processes
    - to operator screens
    - to the device browser
    - to telnet sessions
    - to logging tasks

3.1.3 MIBs, Data In, Discussion

3.1.3.1 Set commands Received Via the Service Port

While it is not possible to estimate the exact frequency with which set commands will be sent to the MIBs (will vary with the nature of the observation being conducted), it is known that the frequency will not be high. The highest rate of set command generation will occur in software processes that are not tied to human rates of interaction with the system. Specifically, the Executor/Antenna software that is responsible for the conduct of an observation. For normal observing, the peak burst rate will occur at scan setup time, with each MIB getting a few (< 10) configuration commands from a software process. Once a scan has been configured, the rate of commands going out to the MIBs is low. For the test antenna, the update rate for antenna position commands and the LO system is only once every 10 seconds. This update rate will be insufficient for the New Mexico Array (EVLA Phase II), but for that case the update rate will still be very modest – once every 5 seconds. Scans typically last at least several minutes, so the rate of new scan setups will also be low. (Note, Chapter 9 of the EVLA Project Book states that the antenna position must be updated every 50 ms to maintain the desired level of pointing accuracy, but this rate is the rate at which the MIB sends commands to the ACU, not the rate at which the MIB receives set commands.)

The most extreme case of on-the-fly mosaicing (OTFM) will represent the highest rate of set command generation. For this case, the rate of pointing command updates from the Executor or its associated Antenna Objects will be 10Hz, i.e., a set command for azimuth and a set command for elevation to each ACU/FRM MIB once every 100 ms.

Set commands coming from operator screens, the device browser, and telnet sessions are all tied to the rate at which a human can interact with the system.

The data type for a set command is an ASCII string. The string length will be tens or hundreds of bytes (commands can be grouped).

3.1.3.2 Get Commands Received Via the Service Port

Get commands from software processes not associated with human interactions with the system are likely to be very infrequent. Currently, the expectation is that software processes will issue “get” commands to MIBs only when those software processes are initializing. After the current state of a MIB has been initially determined, these software processes will maintain their picture of the state of a subsystem by subscribing to multicasts from the MIB data port for monitor data, alarms and alerts, error messages and logging messages. Similar logic applies to operator screens.

The device browser is a general-purpose application intended to provide low-level access to all devices in the system. As the EVLA software matures, its chief use will be the detailed examination of device state for the purpose of troubleshooting. The focus here will be on its behavior with respect to MIB-connected devices. Unlike software processes that perform operational activities and are not tied to human interactions with the system, the device browser does poll the MIBs. It (the device browser) communicates with MIBs via the service port and does not subscribe to the monitor point multicasts emitted by the MIB data port. The reasons for this difference in behavior are bound to the intended applications of the device browser.

Initially, the device browser will send a “get *.*.*” command to the connected MIB to obtain a list of all devices, all monitor and all control points for each device, and all attributes of all monitor and control points. After the initial get command, the device browser will poll the MIB, periodically sending get commands to obtain the values of monitor points.   The default polling rate is once per second (1Hz). The user selectable set of polling rates is once every 5, 25, 100, 250, 500, 1000, 2000, and 5000 milliseconds (200 Hz, 40 Hz, 10Hz, 4Hz, 2Hz, 1Hz, 0.5Hz, 0.2Hz)

The rate of get commands for telnet sessions is governed by the rate at which a human can interact with the system.

The data type for a get command is an ASCII string. The string length will be tens to hundreds of bytes (command grouping).

3.1.3.3 Code Loading

Code loading refers to the downloading, over Ethernet, of new executable images to the MIBs. This function will be performed on only an occasional basis, when updates to MIB software are needed, and is not a part of normal operation of the array. Software to perform code loading has already been implemented. Images are transferred as S-records. This function is properly considered a special case and is excluded from further consideration of the appropriate infrastructure for communication among the distributed processes that will constitute the EVLA Monitor and Control system.

3.1.3.4 XML Device Configuration Files

Device configuration files are ASCII files, in an XML format, that are used to define the publicly exposed monitor and control points associated with devices controlled by MIBs. These files are normally static in nature. A new XML device configuration is downloaded only when the monitor and/or control points for a device are to be changed. As with Code Loading, this function is a special case and will be excluded from further consideration with respect to the purpose of this document.

3.1.4 MIBs, Data Out, Discussion

3.1.4.1 Responses to Set commands when the -v (verbose) option is set (MIB Service Port)

The output generated is quite small. If the set command is successful, a message will be generated that will be either a count of the matching monitor point or control point attributes found, or if used in combination with a time-deferred command, an indication that the time-deferred command was successfully queued and the command sequence number assigned to the command. If the set command fails, some type of informative message will be sent.

The data type for the responses is ASCII strings, and the string length is tens or hundreds of bytes.

3.1.4.2 Responses to Get Commands (MIB Service Port)

Queries from software processes not associated with human interactions with the system would be infrequent. As mentioned previously, such software processes will explicitly query MIBs only when they are initializing. For routine maintenance of state and status, it is expected that these software processes will subscribe to the MIB multicasts of monitor data, alarms, alerts, and error messages. Operator screens will also maintain their view of subsystem state by subscribing to multicasts.

The device browser will probably generate the highest level of responses to get commands. As mentioned in the discussion of data into the MIBs, the device browser does initially send a “get *.*.*” to the selected MIB, and subsequently does poll the MIBs for monitor and control point values using get commands. (Please recall that the device browser polls a MIB via the service port. It does not use the data port multicasts.) The response length for a “get *.*.*” command ranges from a few hundreds of bytes for simple devices up to tens of kilobytes for complex devices. The response length for the subsequent get commands varies from less than 100 bytes for a simple device to a few kilobytes for complex devices. As previously mentioned, the default polling rate of the device browser is once per second (1Hz). The entire set of user selectable set of polling rates is once every 5, 25, 100, 250, 500, 1000, 2000, and 5000 milliseconds (200 Hz, 40 Hz, 10Hz, 4Hz, 2Hz, 1Hz, 0.5Hz, 0.2Hz)

The rate of responses to get commands issued by telnet sessions should be quite low since the commands are issued by humans using a command line interface.

In all cases, the responses to get commands are ASCII strings, using an XML format. The largest possible response is the response to a “get *.*.*” command, i.e., a request that the MIB in question send a description of all devices, all monitor and control points for each device, and all attributes of each monitor and control point. The largest known response (as of Nov 2004) is the response of the M301 to a “get *.*.*”. The length of that response is approximately 39 Kbytes. Issuance of a “get *.*.*” command is a relatively infrequent occurrence. Typically, it would be done by the device browser or some other user interface screen or process only when it first connects to a MIB. A more typical response length would be hundreds of bytes or a few (<10) kilobytes.

3.1.4.3 Monitor Data from the MIB Data Port

Monitor data is delivered as ASCII strings, in XML format. The maximum length of a payload of monitor data would be represented by a packet containing multiple monitor points. Monitor data packets are transmitted as UDP datagrams and are constrained in size to avoid fragmentation. The maximum allowed payload is 1280 bytes.

3.1.4.3.1 Monitor Data To the Archive

Monitor Data Per Antenna

Since the exact complement of MIBs and the number of monitor points per MIB was not precisely known at the time of the team meetings, a statistical approach was used to characterize the monitor data traffic from an antenna.

The MIBs in an antenna were subdivided into three types - complex, simple, and monitor only. The current tally of monitor points (MPs) and control points (CPs) for the modules developed as of 20-Jul-2004 was then used to estimate the number of MPs and CPs to assign to each type of MIB.

Number of MPs & CPs per MIB-controlled device as of 20-Jul-2004

Device	# MPs	#CPs
ACU	92	12
D30x	13	2
L301	17	5
L302	16	10
L304	11	1
L305	11	1
L353	18	3
M301	98	22
P301	21	1
T304	13	16

Number of MPs & CPs assigned to each MIB type

MIB type	# of MPs	# of CPs
complex	125	25
simple	50	10
monitor only	25	0

These estimates for the number of monitor and control points for each type of MIB did not seem to be at too great a variance with the tally of MPs and CPs for the modules for which software had already been developed.

Next, the number of MIBs of each type that will be found in an antenna was estimated. There will be approximately 34 MIBs per antenna. The number of MIBs per type was estimated as 8 complex, 18 simple, and 8 monitor only. The number of complex MIBs was an allowance of 2 complex MIBs per IF, with 4 IFs per antenna. The number of monitor only MIBs approximated the number of power supplies expected for an antenna, and the remaining number was allocated to simple MIBs.

Simple arithmetic, laid out in the tables given below, gives a result of 2100 MPs/antenna and 380 CPs/antenna.

MIB type	# MPs	# MIBs	Total MPs
complex	125	8	1000
simple	50	18	900
monitor only	25	8	200
			2100

MIB type	#CPs	# MIBs	Total CPs
complex	25	8	200
simple	10	18	180
monitor only	0	8	0
			380

A task that archives monitor data had been running for some months. That task was metered to determine the average size of a monitor point. The figure obtained was a bit less than 60 bytes per monitor point.

One now has all of the numbers needed to estimate the amount of monitor data traffic per antenna:

$$2100\frac{MPs}{antenna}\times 60\frac{bytes}{MP}=126{,}000\frac{bytes}{antenna}\cong 123\frac{Kbytes}{antenna}$$

For 27 antennas, the total data volume for a dump of all monitor points from all antennas would be:

$$27 antennas \times 126{,}000\frac{bytes}{antenna}=3{,}402{,}000 bytes \cong 3.24 Mbytes$$

For the VLA, the average rate at which monitor data is archived is once every 15 minutes. If this same rate were to be used for the EVLA, then the average data rate to the archive would be:

$$\frac{3{,}402{,}000 bytes}{15\min}\times\frac{1\min}{60\sec}=\frac{3{,}780 bytes}{\sec}\cong 3.69\frac{Kbytes}{\sec}$$

So, for monitor data to the archive, given the above listed assumptions, the data rate is relatively low.

3.1.4.3.2 Monitor Data to Software Processes and Operator Screens (MIB Data Port)

The expectation is that software processes such as the executor and functionality such as flagging will operate by subscribing to a multicast group or groups to obtain monitor data. Of necessity, the data stream(s) involved will be separate and distinct from the multicast data stream used by the archive task because the rate at which monitor point values are sent to the archive task is too low to satisfy the needs of the executor, flagging, etc.

Within each monitor point data stream (archive, observing, and screens) the rate at which a particular monitor point value is multicast can be varied on a per monitor point basis. The rate from monitor point to monitor point will be highly variable. The system may need to see values for total power every few milliseconds, az & el values from the ACU may be needed only at a rate of 1Hz. Voltage reading from power supplies may be needed very infrequently or not at all until a power supply enters an alarm state.

3.1.4.4 Alarms and Alerts (MIB Data Port)

Alarms and alerts are messages used to flag abnormal operating states that may require correction, and could affect data quality. The MIBs will be the primary, but not exclusive source of alarms and alerts. Alarms and alerts will be multicast. Interested parties subscribe to the multicast group to receive these messages. Alarms and alerts are events, or event-driven. They are not periodic in nature. The volume of traffic generated by alarms and alerts is dependent upon system state. Alarms and alerts, as output by the MIBs, are ASCII strings, in an XML format. Each alarm or alert message contains information for only one device. An alert message is sent when a device enters an alert state and when a device exits an alert state. The alert message is not repeated on a periodic basis while the device is in the alert state. Optionally, the rate at which monitor data is multicast may increase while a device is in the alert state.

3.1.4.5 MIB Error Messages, MIB Logging Messages (MIB Data Port)

Error messages are messages from software processes indicating that the software process itself has entered an abnormal state. For the MIBs, logging messages are informational messages originating in software processes that are used as milestone markers and for debugging.

For the MIBs, error and logging messages use the same implementation. Both are event driven, but may exhibit periodic characteristics if the message generation is embedded in a periodic task. Error and logging messages are multicast. Interested clients receive these messages by subscribing to the appropriate multicast group. The messages are ASCII strings. As currently implemented error and logging messages are ASCII strings, less than 300 bytes in length. Error and logging messages do not currently use an XML format. A possible XML format for error and logging messages has been defined. If implemented, error and logging messages will be constrained to fit within one Ethernet packet, giving a maximum length of 1280 bytes.

3.2 The Control and Monitor Processor (CMP)

The CMP is the means by which the EVLA Monitor and Control System will monitor and control VLA antennas. Physically, it consists of two processor boards (in a VME chasis), with one processor utilizing four Industry Pack (IP) mezzanine modules. The processor that hosts the IP modules connects to one of the two available ports on the VLA Serial Line Controller (SLC). This connection gives the CMP access to the entire VLA monitor data stream coming from all VLA antennas, and provides a means to send commands to the VLA antennas. The processor connected directly to the SLC is a classic, hard real-time system running the VxWorks operating system. The second processor presents the interface of the CMP. It converts VLA-format monitor data to EVLA-format monitor data, EVLA-format antenna commands to VLA-format antenna commands, and presents a set of VLA antenna devices to the EVLA monitor and control system.

Communications traffic for the CMP has been characterized as follows:

Commands from the Observation Executor

- Initial antenna configuration at the start of an observation – tens to hundreds of bytes per VLA antenna
- During an observation, within scans – position & LO settings updates. Normally, 10's of bytes per VLA antenna, once every 10 seconds. Worst case – the most extreme case of OTF mosaicing producing tens of bytes of position data per VLA antenna, at a rate of 10Hz.
- During an observation, scan changes – probably not more than a few tens of bytes of data per VLA antenna

Commands from Operator Screens

- Minimal and occasional – on the order of minutes – chiefly for administrative purposes, at rates set by the speed with which a human can interact with the system

Commands from the device browser or equivalent

- The worst case would be the same polling rates as described for get commands sent to a MIB by the device browser.

Commands to the CMP will be delivered as ASCII strings.

Monitor data from VLA antennas. The exact method is as yet undecided. Possibly polling or multicast data streams as for MIB-controlled devices. Data sinks for VLA antenna monitor data would include:

- Software processes such as the Executor, and those processes involved in the checker and flagging functionality. If polling is used, it would make no sense to poll at a rate that exceeds the rate at which new data is available. For most VLA monitor points this rate is 0.1 Hz. The synch detector values and round trip phase measurements are available at a rate of 1 Hz. ACU status is available at a rate of 5 Hz, and ACU position is available at a rate of 10 Hz.
- Operator Screens. Data rates, per screen, are anticipated to be in the range of hundreds to thousands of bytes per second.
- Low-level screens – device browser screens or the equivalent thereof. Either a scenario similar to the MIB responses to get commands generated by the device browser, or data rates similar or equal to the VLA "monitor word display". The monitor word display transfers one monitor word to the screen every waveguide cycle, with a monitor word occupying approximately 20 bytes. Approximating the waveguide cycle rate as 20Hz gives a data transfer rate of approximately 400 bytes/second. The monitor word display is not run during normal observing or as a part of normal operations. It is used to diagnose problems.

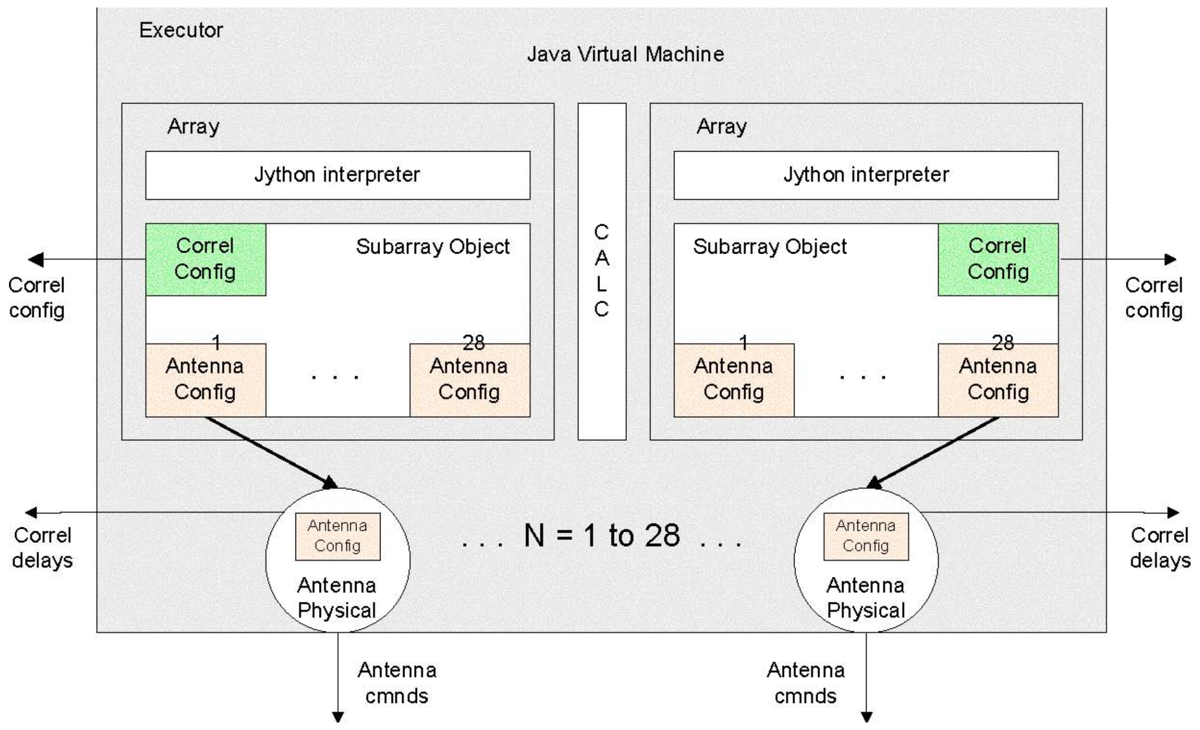
Monitor data from the CMP will be delivered as ASCII strings using an XML format.

3.3 The Observation Executor and Antenna Objects

The communication requirements among the components of the (Observation) Executor and between the Executor and antennas depend upon the relationship of the components. The diagram given below is a sketch of some of those relationships as embodied in the test antenna software developed by Barry Clark. Since the details of the design of the production version of the Executor are still being settled, Barry's design will be used here as a proxy for the final version.

The diagram indicates that the Executor uses a single process, multithreaded model, with all of the components sharing the same address space. Very briefly, the Executor instantiates 28 AntennaPhysical objects. These objects are persistent and represent the actual antennas in the physical array. Their lifetime is the same as the lifetime of the Executor. Array objects come and go. These objects represent the first level of subarraying. The AntennaPhysical objects assigned to an Array represent the actual antennas that are associated with a particular observation and control script. The control script cannot move antennas among Arrays. A control script can create subarrays within an Array. This capability represents a second level of subarraying, under the control of the observer. An Array object may, therefore, contain one or more Subarray objects. Correlator configuration is handled at the level of the Subarray objects. Correlator delay models are generated at the level of AntennaPhysical.

Each Array object has 28 Antenna Configuration or Antenna data objects that are mapped, as disjoint sets, into the one or more Subarray objects contained in the Array objects. In the diagram given below, since each Array is shown with only one Subarray all 28 Antenna Configuration objects are allocated to the single Subarray object in each Array.



Observation Executor

Antenna Configuration objects are mapped to AntennaPhysical objects or to null pointers if a particular antenna has not been assigned to the Array in question. Cloning an Antenna Configuration object in the appropriate AntennaPhysical object specifies the desired state of an antenna. In the current implementation of the Executor, the AntennaPhysical objects issue commands directly to the EVLA MIBs to configure an antenna for an observation.

One important aspect of the diagram of the Executor is the relationship of CALC to those components needing its results. CALC produces both pointing angles and delays. It is called by the Antenna Configuration objects to produce an initial set of three pointing angles and delays, and by the AntennaPhysical objects to obtain current pointing angles and delays. The rate at which results are needed from CALC for the most extreme case of OTF mosaicing (OTFM) is a driver for the architecture of the Executor. Benchmarks show that the EVLA control system host computer cannot currently execute CALC quickly enough to supply results for 27 antennas at the rate needed for the worst case of OTFM. There is a need, therefore, to optimize the execution time of CALC.

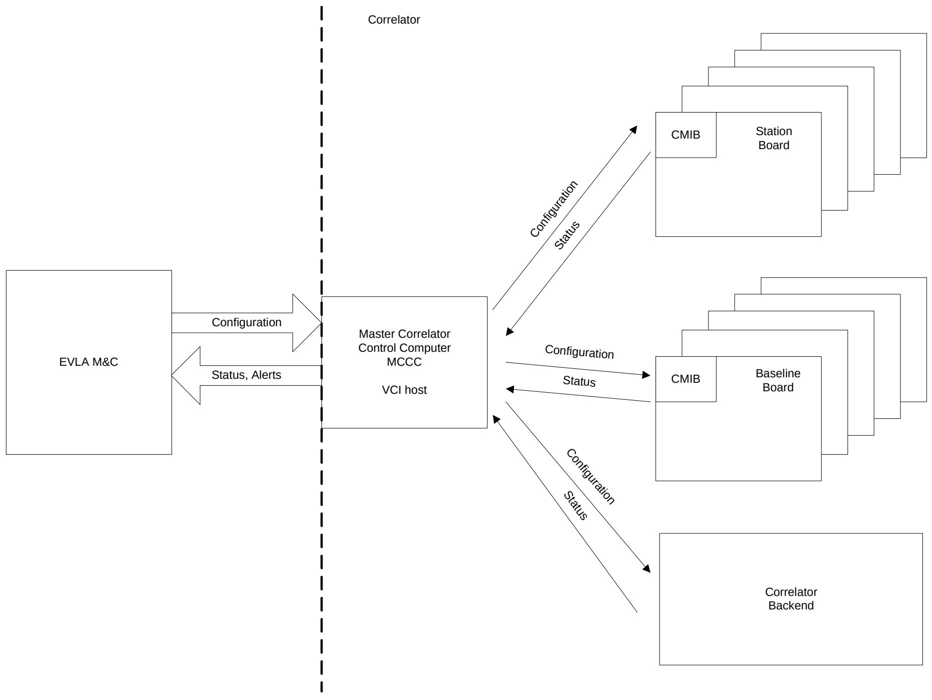
The major overhead for CALC is incurred when it is passed a new timestamp. Successive calls for different antennas that use the same timestamp execute much more quickly. The difference in processing time can be as much as 90%. However, successive calls achieve this higher level of efficiency only if CALC is not allowed to terminate execution between the first call with a new timestamp and successive calls that contain the same timestamp as the first. CALC, therefore, lives in a persistent execution space.

Experience with the VLBA shows that when CALC is implemented as a server and called remotely it spends as much as 20% of its time on communications overhead. To minimize communications overhead CALC is run in the same address space as the Executor and is treated as a function call.

While this brief description glosses over quite a number of issues, it does describe the relationships among the components of the Executor. Since the entire set of components share the same address space the major issue with respect to communication among distributed processes is the commands emitted by the AntennaPhysical objects. The driver here is still the requirements for OTF mosaicing. For the most extreme case of OTFM, the AntennaPhysical objects must issue position updates to the ACU and polynomials to the correlator at the rate of 10 Hz.

3.4 Correlator and Correlator Backend Configuration

The interface to the correlator and the correlator backend will be a software entity known as the Virtual Correlator Interface (VCI). The figure given below is a diagram of the architecture of the VCI. Briefly, the EVLA Monitor and Control System will send configuration requests to and receive status, error, alarm and alert information from an interface that will be hosted by the Master Correlator Control Computer (MCCC). The VCI transforms the received configuration requests into requests appropriate to the components of the correlator – station boards, baseline boards, phasing boards (not shown), and the correlator backend. Configuration messages will then be sent to the correlator module interface boards (CMIBs) located on each correlator board, and to the correlator backend end (CBE). For a 32-station correlator there will be 128 station boards and 160 baseline boards. The correlator backend will be a cluster, currently estimated to contain 64 processors. The EVLA Monitor and Control system need not address the correlator boards or CBE processors individually. That task is handled by the VCI.



Virtual Correlator Interface

The VCI is concerned only with monitor and control. Astronomical data and meta-data relating to an observation are not transferred over the VCI.

The VCI messages to be used for correlator configuration have not been sufficiently defined at this point in time to enumerate the message lengths or rates of message delivery. It is know that VCI messages will use an XML format.

Some work has also been done on configuration messages for the correlator backend (CBE). Current thinking is that CBE configuration will be specified using an XML format. The CBE prototype is currently configured in this manner. The use of XML to specify a CBE configuration for 27 antennas, 8 basebands, and 16 subbands will result in a file size of hundreds of kilobytes.

The VCI is being designed and developed by Sonja Vrcic of the Dominion Radio Astrophysical Observatory (DRAO). After much work and thought she has decided that communications with the VCI should be messaged-based. When asked for the rational behind her decision to use a message-based approach, she replied as follows:

A decision to use message based interface for communication between the EVLA Monitor & Control System and the correlator is based on the following:

loose coupling: The message based interface allows loose coupling between the M&C System and the correlator. Loose coupling will allow the EVLA Monitor & Control System and the correlator software to evolve independently. Also, loose coupling will allow the EVLA M&C System(s) and the correlator (MCCC) to become active independently of each other, which is highly desirable in the environment where the same M&C System may be managing more than one correlator and the correlator may be controlled by multiple users.

simplicity: A message based interface will result in simpler software system compared to systems that use distributed objects. Use of standard communication protocols and techniques may simplify system development and allow for faster development.

performance: Message based communication is likely to provide better performance than a distributed object system that uses remote procedure invocation. Performance is a concern since a large number of messages will be transferred over the VCI when full system re-configuration is required, or in the case of a major failure. VCI, as a single point of access to the correlator, must be able to handle large number of messages efficiently in order to keep up with requests generated by multiple users. It is likely that delay and phase models will be transferred over the VCI, which only increases requirements related to performance.

testing and debugging: A message based interface allows operators and developers to monitor content of the messages exchanged over the VCI, and to generate VCI messages. This ability will facilitate testing and debugging of the system.

phased development: Use of a standards based messaging interface will simplify testing and debugging of software components. Use of human readable XML messages will allow designers to generate and interpret messages relatively easy, which will allow testing of a software component if the development of its counterpart (on the other side of the interface) is delayed.

The distributed object system is often used to mask the fact that a system is distributed over several computers and/or locations. In the EVLA System, the VCI messages will be generated by the M&C System which is operated by highly trained personnel. Here, the intention is not to hide the distributed nature of the system, but to facilitate diagnostics and troubleshooting. In the case of the EVLA system, and in particular the EVLA correlator, decoupling of the correlator from the Monitor & Control System and exposing the system architecture is desirable. Such approach will allow each system to function and evolve relatively independently.

The message based interface based on the standards based infrastructure is perceived as the best solution for the VCI interface.

3.5 Antsol/Telcal

Antsol is a software process that solves for the complex gains of the antenna. These results are then used for a variety of purposes, including:

- data quality displays (the “F” display in the VLA system)
- operational purposes including Tsys estimates
- real time determination of focus & pointing offsets
- phasing the array
- as a post-hoc diagnostic tool to determine when an antenna began to misbehave

- the contents of the Antsol files are piped to the AOC and placed in a flux calibrator database for use by astronomers

Antsol is a component of the VLA online software system. Telcal (referred to as Real-time Calibrator Analysis in the EVLA e2e Science Software Requirements document) is the EVLA successor to Antsol. Most of this section is devoted to Antsol, as currently implemented for the VLA, with the assumption that it will not differ much for the hybrid array using the VLA correlator. Telcal becomes an issue when dealing with output from the WIDAR correlator (or, more accurately, from the CBE). The design parameters relevant to Telcal have not yet been specified.

Antsol, Inputs

The amount of visibility data transferred into Antsol's address space for continuum is the product of the number of baselines with the number of IF pairs with the number of polarization products.

Baselines	IF pairs	Polarization Products
378	2	4

and, for spectral line,

$$line = continuum \times nchs(\#channels)$$

with nch as follows:

IF pairs	Pol Products	Correl Mode	nch
1	1	1A – 1D	16 - 512
1	2	2AB – 2CD	8 – 256
1	4	Px	4 – 128
2	2	4 IF mode	4 - 128

The visibilities are expressed as complex numbers, i.e., as a pair of floating point (REAL) numbers. Taking a floating point number as 32 bits means that each complex number will occupy 64 bits (8 bytes).

For continuum,

$$378 \times 2 \times 4 = 3{,}024\ complex\,numbers \times 8\frac{bytes}{complex\,\#} = 24{,}192\ bytes = 23.625\ Kbytes$$

For the spectral line modes, the worst cases are 1 IF pair, 1 polarization product with 512 channels, and 2 IF pairs, 2 polarization products, with 128 channels,

$$378 \times 1 \times 1 \times 512 = 193{,}536\ complex\#s \times 8 \frac{bytes}{complex\#} = 1{,}512\ Kbytes \cong 1.5\ Mbytes$$

$$378 \times 2 \times 2 \times 128 = 193{,}536\ complex\#s \times 8 \frac{bytes}{complex\#} = 1{,}512\ Kbytes \cong 1.5\ Mbytes$$

Antsol currently uses only the channel 0 summation – one (1) complex number per IF. However, all of the above given data is transferred into Antsol's address space to permit a more sophisticated mode of operation that was never implemented.

The visibility data can be viewed as being organized as arrays.

In addition to visibility data, Antsol also requires:

- the correlator mode
- # of channels per IF
- the calibrator flag
- flagging data (both antenna-based and baseline-based)
- U-V limits
  - U & V are expressed in nanoseconds
  - the limits are expressed as 2 radii defining an annular region, with the radii currently set to 0 and infinity
- the reference antenna
- the scan start time
- the previous set of solutions (already present in Antsol's address space)

This additional data can be represented as a handful of bytes consisting of integers, reals, and a short ASCII string for the correlator mode. This data comes from the archive record, which gets it from the program named Dump (the VLA equivalent of DCAF), which gets it from the VLA software global common regions.

Frequency of data input

Every integration – long-term integration, not correlator dumps. For the VLA, the relevant integration period is the long-term integration implemented in the array processor. For the EVLA, the relevant integration period would be the integration implemented in the correlator backend (CBE), not the correlator long-term accumulator (LTA). For the VLA, for the worst-case spectral line modes, the shortest integration period that is allowed is 6 2/3 seconds.

Antsol Outputs

I. Complex antenna gains consisting of one complex # for each IF for each antenna = 27 antennas X 4 IFs/antenna = 108 complex #s X 8 bytes/complex # = 864 bytes

II. A set of residuals (the closure errors) for each IF for each antenna. The residuals are represented by 1 complex # for each IF, for each antenna so, again, the volume of the output is 864 bytes.

Frequency of data output

Every integration. As with the input data, the relevant integration period is the long-term integration implemented in the array processor (VLA) or the correlator backend (EVLA).

Antsol results are distributed to:

- the VLA software commons They are written into a single buffer, which is overwritten each time a new set of results are available
- a file

For Telcal in the EVLA, the volume of visibility data will be much higher, and it will be distributed among various nodes of the correlator backend cluster. The team put the question of the possibility of parallelizing Antsol in some manner to Ken Sowinski. His reply was as follows:

```
Consider that the computational part of Antsol can be divided in two parts;
first collect all the relevant data into a single array, and second use that
data to come up with a least squares estimate of the antenna gains.  The second
part requires relatively little data and by current standards only a little
computation (a few tenths of a second on a Modcomp).


The first part involves essentially no computation but only data selection.  In
the current Antsol, data selection amounts to first being sure that the baseline
being considered is not flagged and lies within specified u-v limits, and
secondly selecting the ‘continuum’ channel from the frequency spectrum for the
baseline.  Considering just these actions it would be to our advantage to
reverse them; select the data from the frequency spectrum relevant to Antsol and
pass it, then in Antsol suppress it if the data is flagged or not in the
required u-v range.  To do this requires that nodes processing correlated data
know the frequency range of interest to Antsol, but nothing more.


A complication we will face over the current system is that we don’t want to do
a vector sum over frequency until any residual phase slope is removed.  This
requires significant processing before the data integrated over frequency is
sent to Antsol.  Alternatively, is it always acceptable to simplify the
computation by using amp-scalar summing?
```

In summary, I think not much data need be passed to Antsol if we are willing to accept that the backend nodes (if that is where the data is coming from) know something about what Antsol wants to see.

In other words, it may be possible to split Antsol into two parts, one of which resides on each correlator backend node. This component can decimate the data before transferring it downstream to the second-stage of Antsol. Exactly how much data must be transferred is an unknown at this point, but the strategy outlined above does bring the solution of the problem of getting the needed visibility data into Telcal's address space via parallel or high-speed serial I/O into the realm of the possible.

3.6 Operator Screens

At the time the team was meeting relatively little was known about operator screens. However, some general statements relating to traffic to and from the screens could be made.

Operator screens are custom displays used to monitor and control the array. Operator screens will run, primarily, on operator workstations located in the VLA Control Room at the site, but will have the capability of running on any workstation, at any location, as long as the machine in question has access to EVLA resources such as antenna MIBs, the monitor and control host machine(s), etc. The monitor portion of their functionality will predominate over control, and the amount of data received by the displays will far outweigh the amount of data sent by the displays.

Two models exist for getting information to the displays. One would be to actively poll EVLA resources; the other is for the resources to multicast the needed data, with the screens taking the role of passive listeners to the multicast. The screens output stream from the MIB data port is an example of the multicast approach. The probable data type for data received by screens is ASCII, in an XML format, as per the MIB service and data ports. For data sent by operator screens, the likely data format is ASCII, not XML encoded, as per the get and set commands received at the MIB service port.

If polling is used, and if we assume that a typical screen will contain the data from 40 monitor points, and use the figure of 60 bytes per monitor point, then a screen's worth of input data would be 2,400 bytes (ignoring headers). If 5 screens, each containing 40 points, poll for data at a rate of 1 Hz, then the data rate would be 12,000 bytes/second.

Polling is actually the less likely of the two alternatives for getting data to operator screens. More probable is the use of multicast, from the MIBs and from whatever other processes have data of interest to operator screens. If multicast is used, then the traffic to operator screens is simply a function of the multicast traffic from the sending processes. It is not a function of the complexity of the screens or of the number of screens. Operator screens would simply subscribe to the appropriate multicast group(s) and filter for the data of interest. The use of multicast has been assumed in all other sections of this document.

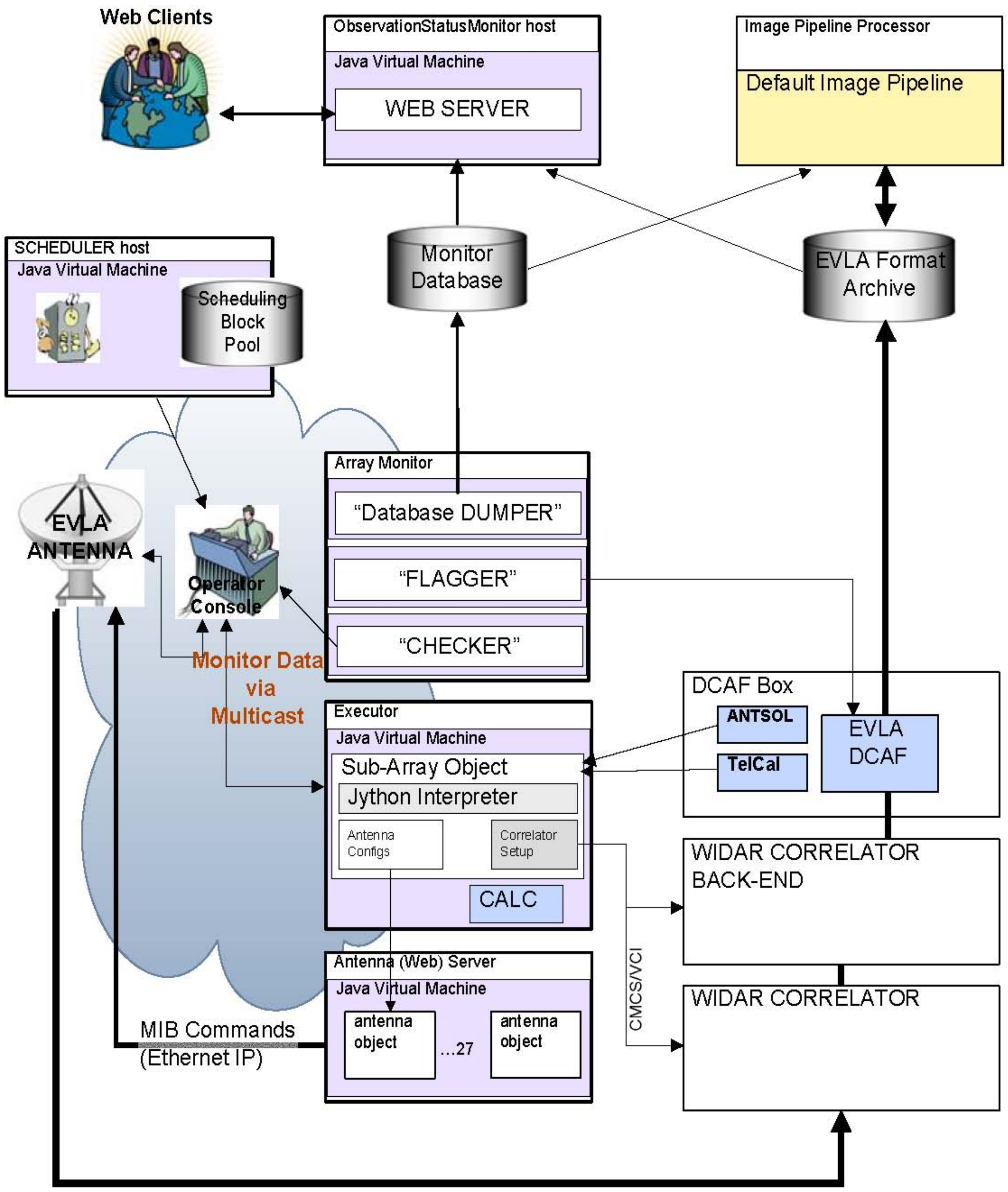
4 Deployment Scenarios

Deployment scenarios are important to a consideration of communications infrastructure because deployment determines which processes are local or remote with respect to one another. The term local is meant to denote communication within the same address space. Distributed or remote communication refers to communication across address spaces, with the further distinction of communication across address spaces on the same machine versus communication to other address spaces on different machines. In terms of speed and overhead, local communication is generally the fastest and has the lowest overhead, and communication to address spaces on other machines is usually the slowest and has the highest overhead.

The diagram given below shows one possible EVLA deployment scenario. It is presented as a sanity check to help determine if some significant aspect of the possible communications scenarios has been overlooked. Possible omissions with respect to communication across address spaces on different machines are particularly important.

In this diagram, discrete boxes represent separate computing platforms. The components of the EVLA Monitor and Control System occupy, basically, the bottom two-thirds of the page, i.e., from the box labeled "Array Monitor" down. Likely communications traffic among most of the components shown in this diagram has already been discussed. For example, MIB commands to an EVLA antenna was discussed in section 3.1.3. The operator screens that would run on the Operator Console were discussed in section 3.6, the Executor in section 3.3, Telcal & Antsol in the DCAF box have been discussed in section 3.5, the WIDAR correlator and Correlator backend in section 3.4. One significant difference between this diagram and the material so far presented is the diagram shows an Antenna Server, sending commands to EVLA antennas, residing on a computing platform separate and distinct from the computing platform that hosts the Executor. In the discussion of the Executor, commands were sent to EVLA antennas by the AntennaPhysical objects, and these objects were shown as co-resident with the other components of the Executor. This deployment diagram, therefore, implies an additional communication requirement – between the Executor and an Antenna Server, across address spaces on different machines. As is the case for sending commands to antenna MIBs, the driver for this communication would be the most extreme case of OTF mosaicing - position updates for the ACUs. Presumably, delay updates for the correlator would not be an issue for an antenna server, but delay updates must still be communicated across the boundary between the Executor and the WIDAR correlator.

For the EVLA test antenna software, the "Database Dumper", "Flagger", and "Checker" functions run in the same machine as the Executor. Is it not unreasonable to assume that these functions may eventually be pushed onto a separate computing platform. "Database Dumper" is the monitor database archiving task. It subscribes to the monitor data archive multicast discussed in section 3.1.4.3.1, unpacks the XML data and sends it off to an Oracle database. "Flagger" listens to the multicast alarms and alerts and forwards the relevant data to DCAF for the flagging of archive records. "Checker" also listens to alarms and alerts, and displays the relevant information to the array operators. Alarms and alerts coming from the MIBs were discussed in section 3.1.4.4. Not previously mentioned is that a few other processes in the EVLA Monitor and Control System will also produce alarms and alerts. For example, the AntennaPhysical objects in the Executor are likely to emit alerts at source changes. Deployment of the monitor database archiving task, flagging, and Checker on a machine separate and distinct from the host machine for the Executor will certainly have no impact on the communications overhead for alarms and alerts produced by the MIBs. That these tasks may need to catch alarms and alerts multicast by the Executor should not be a significant factor.



Possible EVLA Deployment Scenario

5 Additional Considerations

In addition to the foregoing material, the following items are also relevant to a consideration of the nature of the communications infrastructure for the EVLA Monitor and Control system:

- Extension of the EVLA Monitor and Control System to Phase II of the EVLA
- The possibility of moving EVLA Operations from the VLA site to the AOC in Socorro
- The possibility of eventually extending the EVLA Monitor and Control System to include operation of the VLBA antennas
- The transition plan, i.e., the creation of software to operate the hybrid array, first with the Modcomps still present in the system, and then with the Modcomps retired, but still with VLA antennas and the VLA correlator

5.1 Extension of the EVLA Monitor and Control System to EVLA Phase II

EVLA Phase II includes the addition of new antennas (8 new + 2 converted VLBA antennas) at fixed locations relatively distant (~200Km to 250km) from the VLA, and use of the WIDAR correlator as the VLBA correlator. The relevant considerations here are 1) geographical dispersion, and 2) the need to modify and extend the EVLA Monitor and Control System software to support these new capabilities. The goal is to create now an EVLA Monitor and Control System that can be modified and extended with minimal impact on the software already deployed.

5.2 Moving EVLA Operations from the VLA Site to the AOC in Socorro

A group known as the Transition Plan Committee, chaired by Frazer Owen, is currently examining the possibility of moving EVLA operations from the VLA site on the Plains of San Augustin to the Array Operations Center (AOC) in Socorro, NM. If this possibility is to be accommodated by the EVLA Monitor and Control System without requiring a major rewrite of the software then the approach taken now must be inherently remoteable, and have a high degree of deployment configurability. The ability to handle a geographically dispersed system is also required.

5.3 Extending the EVLA M & C System to Include Monitor and Control of VLBA Antennas

This step has not been formally proposed as a requirement for the EVLA Monitor and Control system, but the possibility has been discussed, and it would be desirable to create an EVLA Monitor and Control System that does not preclude doing it. The relevant characteristics for the software would be the ability to handle a very widely geographically dispersed system with diverse software platforms. The software must be remoteable and must scale.

5.4 Transitioning from the VLA Control System to the EVLA Control System

A document entitled *VLA/EVLA Transition Observing System Development and Re-engineering Plan* (*available at http://www.aoc.nrao.edu/evla/techdocs/computer/workdocs/index.shtml as document #37*) outlines the steps by which the transition will be made from the present VLA Monitor and Control System to an EVLA Monitor and Control System that will then control the hybrid array. In very broad terms, this plan specifies, first, an exchange of information & events between the VLA Control System and the EVLA Control System, and then a re-allocation of control functionality from the VLA Control System to the EVLA or EVLA accessible elements, followed by retirement of the VLA Control System.

The exchange of information and events required by the Transition Plan includes items such as flagging information for EVLA antennas sent to the VLA Control System, synchronization events marking the start of a scan sent from the VLA Control System to the EVLA Control System, antenna complex gains sent from the VLA Control System, etc. The important point to note is that one endpoint of this exchange of information and events, the VLA Control System, is a pre-existing, legacy system that is not object-oriented, is CPU-cycle and memory limited, and will be retired in less than three years.

Two subsystems currently under development play a crucial role in the transition plan. One subsystem, known as the Control and Monitor Processor (the CMP) will be used by the EVLA Monitor and Control System to monitor and control VLA antennas. The other subsystem is a new controller for the current (VLA) correlator.

The current plan for the public interface presented by the CMP is to implement the MIB service port interface and the MIB data port multicasts. The advantage here is reuse of code developed for the MIBs and consistency of interfaces among devices.

The new VLA correlator controller is an embedded, resource-constrained subsystem that must meet tight, hard real-time deadlines. The code is written in C, and is not object oriented. For its initial deployment, it will be controlled by the Modcomp computers and the VLA Control System, using a message-based interface.

An intermediate step in the transition plan calls for the migration of

1) the software that solves for complex antenna gains (Antsol),
2) the software used to develop pointing offsets for reference pointing, and
3) the archiving of metadata and visibility data

from the VLA Control System to a new, separate subsystem that is accessible by both the VLA Control System and the EVLA Control System. Because this new, separate subsystem must be accessible to the VLA Control System it will be message-based, using sockets over Ethernet.

6 **Conclusions & Recommendation**

So far, what has been presented is a logical analysis, underpinned in many cases by actual deployed and running software, that attempts to characterize the types, rates, and volumes of information and data exchanges among the processes that will constitute the EVLA Monitor and Control system.

With respect to the matter of communications infrastructure, the question becomes how is this information to be distributed? There are two basic approaches to this problem – distributed objects and messaging. Speaking very loosely, we define a distributed object system to be a system composed of objects that communicate with one another via method invocations, be they local or remote. At least at the level of the communicating objects, the distributed object approach requires that objects be used throughout the system. We define a messaging system to be communication among processes by (short) ASCII-based strings of some defined format. These strings are, in effect, a request for some transformation of state in the receiving process.

6.1 Objects Everywhere Is Not Possible

The distributed object approach presents an immediate problem for the EVLA Monitor and Control system. The EVLA Monitor and Control system will not and cannot use objects throughout. To start, there is the matter of the MIBs. The MIBs are a resource limited, embedded processor. Early in the course of EVLA design and development a decision was made to prohibit the use of off-chip RAM in the MIBs in order to mitigate RFI. All code in the MIBs must execute from on-chip RAM. The total amount of RAM available in the MIB processor is 1.5 Mbytes. 1.5Mbytes of RAM is insufficient to support an object-oriented approach for the MIB applications software. An object-oriented approach for the MIB framework software was attempted. Early versions of the framework software were written in C++ and did use objects. It was quickly discovered that the memory requirements for a C++ implementation were much higher than the memory required for equivalent functionality written in C. Among other issues, the dynamic nature of memory allocation and deallocation that is associated with the object-oriented approach is a poor fit to the tight RAM budget of the MIB. Additionally, the available RAM in the MIB is totally inadequate with respect to the requirements of the middleware, such as CORBA, that is used to implement communication among objects in a distributed object system. Messaging systems are simply less demanding of system resources than is the distributed object approach. Using a message-based approach, it has been possible to develop a very clean interface to the MIB that supports all of the required functionality within the resources available.

There are a number of additional subsystems within the EVLA that also use a message-based approach.

For sound and sensible reasons already cited, a decision was made to use a message-based approach for the VCI.

Those elements of the transition plan which require communication between the VLA control system (Modcomp computers) and the EVLA control system are message-based, and must be message-based. It is simply not possible, or, at the least makes no sense, to attempt to retrofit the VLA control system with a distributed object approach. To implement a distributed object approach for the exchange of events and information, be it CORBA-based or some other method, would require that a middleware layer or some sort of adapter or adapters be implemented for the VLA Control System. Such an undertaking seems in no way reasonable or justified by either the life expectancy of the VLA Control System or by a comparison of the effort required by a distributed object approach to the effort required by a message-based approach using sockets over Ethernet.

The CMP, the means by which the EVLA will monitor and control VLA antennas, is message-based. While it is theoretically possible to expand the resources available to the CMP to support a distributed object, remote method invocation type of interface, doing so would require a significant investment of time, make reuse of the MIB Service port, Data port and Framework software impossible, and would create an interface that is different from and incompatible with the interface used for EVLA antennas.

The new VLA correlator controller, required for control of the hybrid array, is message-based, and must be so if it is to communicate with the VLA control system. An attempt to re-engineer this subsystem to permit of a distributed object type solution would require extensive modification of the work that has already been done. (It should be noted that initial development of the new VLA correlator controller predates the EVLA project.)

Those elements of the transition plan that call for the piecemeal migration of functionality from the Modcomp-based VLA control system to a system accessible by both the VLA and EVLA control systems are message-based. Again, since the system on which these capabilities will reside must, during a period of its existence, communicate with the VLA control system, a distributed object approach is not possible unless one writes two control interfaces – a message-based interface for the VLA control system and a distributed object interface for the EVLA control system.

Unless one is willing to undertake a very substantial modification of the VLA Control System, a thorough reworking of the CMP and VLA controller software, and a reformulation of the plans for migration of VLA functionality to the EVLA, a message-based approach using sockets over Ethernet seems to be virtually mandated for transition from the VLA to the EVLA.

6.2 Best Fit

The information and data to be distributed by the monitor and control system consists chiefly of

- commands (MIBs) and configuration requests (VCI)
- responses to commands and configuration requests
- monitor data
- alarms and alerts
- error messages
- logging messages
- data structures containing configuration or model information
- metadata relating to an observation
- visibility data
- a few special cases such as loading code or XML configuration files into a MIB

Commands and configuration requests, responses to commands and configuration requests, monitor data, alarms and alerts, error messages and logging messages are all ASCII strings, most but not all in XML format. The configuration files for the correlator backend are basically data structures, presented as ASCII strings in an XML format. Metadata, while not discussed in this document will probably be ASCII strings, probably in XML format. Visibility data will almost certainly not be ASCII strings, but rather binary data. Code loading uses a special (S-record) format, and the MIB configuration files are ASCII strings in an XML format. Virtually all of the monitor and control information and state information that is to be distributed within the monitor and control system is ASCII strings, most of it tens or hundreds of bytes in length. With the exception of correlator backend configuration, the largest anticipated size is in the range of a few tens of kilobytes. Messaging seems to be a natural fit to the distribution of monitor and control information within the system. Messaging is not a reasonable solution for the distribution of visibility data within the system.

6.3 Lack of Pervasiveness & Increased Complexity

The total processor complement in the EVLA Monitor and Control system for Phase I of the EVLA will be roughly as follows: ~ 1000 MIBs, 3 to 5 systems in the VLA Control Building (not including operator terminals), the Master Correlator Control Computer (the VCI host), a Correlator Power Control Computer (CPCC), ~ 300 CMIBs, the CBE multiple-processor cluster (estimated at

64 processors), and, during the transition, 2 processors in the CMP. In round numbers, that's approximately 1375 processors. Of that total, only the 3 to 5 processors in the Control Building are actually suitable candidates for a distributed object approach. Even if one argues that the core functionality of the monitor and control system resides on those 3 to 5 processors, the fact that the remaining 1370 processors all use a message-based approach to communication among distributed processes undermines the utility that might be had from using a distributed object approach. It might be possible to write adapters or translators of some sort that would allow the use of distributed objects communicating via CORBA-mediated remote method invocations at the higher levels of the Monitor and Control system while continuing to use a message-based approach for the MIBs, VCI, and the CMP. However, there seems to be no technical or engineering advantage to doing so, and more than a few disadvantages. A very considerable and significant amount of complexity would be added to a real-time system with no payoff in terms of increased system functionality or capabilities. Since complexity would be added, the resulting system would be less robust, more difficult to maintain, modify and extend, require longer timelines and more manpower to develop, and need more heavily resourced, and, therefore, more costly computing platforms.

6.4 Suitability

In addition to the arguments and issues outlined in the preceding sections, the team identified the following characteristics of a message-based approach that makes it well suited to the needs and requirements of the EVLA software:

A message-based approach provides strong support for loose coupling. Loose coupling provides a host of advantages. It is a robust approach to the organization of a real time software system that helps to isolate errors. It increases modularity and decreases the degree of interdependence within and across subsystems. Increased modularity and decreased interdependence will help to maximize the probability that it will be possible to extend the EVLA control system to EVLA Phase II and to control of VLBA antennas without a major rewrite of the EVLA software.

The message-based approach will result in a simpler software system, requiring fewer resources and less time to develop than a distributed object approach.

It will provide a level of performance that is likely to be superior to a distributed object approach that uses a heavier-weight solution for communication among objects.

A message-based approach, using ASCII messages, maximizes the visibility of the interactions among distributed components, which should facilitate debugging.

It is scalable to and beyond the degree required.

A message-based approach is agnostic with respect to computing platforms, operating systems, programming languages, and programming paradigms. The EVLA software is currently running the Nucleus Plus operating system on the MIBs (a TC11IB processor). The CMP uses TimeSys Linux running on a PowerPC processor and VxWorks running on a 68040 processor. Both processors run in a VME chassis. Core monitor and control processes, such as the Executor, flagging, monitor data archiving, etc. run under Red Hat Linux on a Pentium-based system (mchost). C, Java, and jython are used. Some of the EVLA software is object-oriented, and some is not. The VLA control system runs on Modcomp computers, which run a proprietary Modcomp operating system. The VLA online system is written chiefly in Fortran and assembler. None of it is object-oriented. All of these platforms and components must be accommodated. Additionally, it is wise to preserve future options for the use of other platforms, operating systems and languages, both during development of the initial software for the EVLA and after deployment.

Very importantly, a message-based approach provides strong support for the independent evolution of components within a system. For the EVLA, the independent evolution of components does not mean the parallel development of components within the context of an agreed upon common infrastructure, along similar tightly coupled timelines, in the same or different locations. For the EVLA, the situation is more complex. For example, the EVLA control system must be able to evolve independently of the VLA Control System, even while interfacing to it, and interacting with it to differing degrees and in differing ways at various stages in the Transition plan. The VLA Control System is a legacy system. It already exists, and to a large degree must be taken as a given. The MIB interface evolved independently with respect to other components of the EVLA control system because of the need to support antenna hardware development at the earliest phase of the project. The MIB interface is now a relatively mature and fully functional interface. It is doubtful that project timelines, manpower resources, and the available dollars would support a major rework of the MIB software. And, certainly, it is clear that EVLA Phase II software and possible control of VLBA antennas will happen on timelines that are quite independent of the timeline for development of the control software for EVLA Phase I.

6.5 Requirements of the Transition Plan for Control of the Hybrid Array

The Transition Plan for control of the hybrid array has a major impact on and is a major constraint upon the decision concerning the approach to take for the EVLA communications infrastructure. Various aspects of this plan require the exchange of information and events between the VLA and EVLA Control Systems and the development of subsystems that can be controlled by both the VLA Control System and the EVLA Control System. The VLA Control system is a short-lifetime, resource-limited, legacy system running a proprietary Modcomp operating system with applications written chiefly in Fortran and assembler that are not object oriented. An attempt to retrofit it with a distributed object approach to communications seems very ill advised. A message-based approach using sockets over Ethernet is by far the more efficient approach to the issues of inter-system communication and mutually accessible subsystems for the specific case of the VLA-EVLA transition software and control of the hybrid array.

If the transition software is to be message-based, it becomes much more difficult to argue in favor of a distributed object approach for the mature form of the EVLA software. Two separate, mostly discontinuous, efforts would be required – one for the transition software and one for the final form of the EVLA software. EVLA software components could not be grown from or evolved from transition software components. Indeed, even most of the physical components of the hybrid array would be useless with respect to testing EVLA software – only the EVLA computing platforms and EVLA-converted antennas would be workable testbeds. Further, it would be necessary to unload the transition software and load the developing EVLA software to conduct tests. Scheduling of test time then becomes an issue, and test time itself becomes a scarce resource. Operators would require training for two different types of systems. The amount of throw-away software would increase dramatically to include virtually all of the software developed to address transition issues. And the list does not end with the items so far mentioned. Many, many more issues would arise.

A strong and distinct dichotomy between the approaches taken by the transition monitor and control software and the mature form of the EVLA monitor and control software simply does not provide reasonable software development scenarios. It does not make efficient use of the available manpower, it would make it much more difficult, if not impossible, to meet the current EVLA project schedule, and it makes ensuring continuous operation of the array very problematic.

6.6 Disadvantages of Messaging

One of the chief disadvantages in message-based interfaces is that software processes wishing to send messages to a receiving process must know both the syntax and semantics of the messages. In other words, the sending process must be given knowledge of what a receiving process will consider a valid message. If a new message or message type is added, both the receiving process and the clients must change. The burden of this disadvantage can be considerably reduced by paying careful attention to the message syntax, and by making the interface self-describing. The message-based interface that has been implemented for the MIB is a case in point.

There are only two basic commands for the MIB – set and get. The basic command format is:

```
set

or      <device name>.<monitor or control point name>.<attribute name>=<value>

get
```

The syntax is simple and straightforward. Not a great burden on a sending process. Additionally, the sending process (client) need not be have a built-in awareness of the devices, monitor points, and control points for each and every MIB-controlled device because the MIB interface is self-describing. A "get *.*.*" command will return to the process that issued the query a list of all devices connected to the MIB, all monitor and control points for each device, and all attributes for each monitor and control point. That the MIB interface is self-describing allows a single device browser, with no built in knowledge of the devices, to be used to command and to determine the state of any MIB-connected device in the system. In this manner, the burden of requiring a client to have detailed knowledge of the vocabulary of the device or process with which it wishes to communicate can be substantially reduced.

Another disadvantage of the message-based approach is that it reduces the opportunity for reuse of ALMA software. It is understandable that, from a management point of view, it is very desirable to reuse as much of the ALMA software as possible, and to have as much uniformity across NRAO as can be achieved. The key phrases are "as possible" and "as can be achieved". The ALMA Common Software (ACS) uses the distributed object approach. Unfortunately, that approach is either a poor fit or simply unworkable with respect to many of the engineering and technical requirements and constraints found in the EVLA project.

6.7 An Aside, The Use of UDP and Multicasts as a Message Protocol and Transport

Messages can be transported in many ways – TCP/IP, UDP unicast, and UDP multicast to name just a few. While the issue of the transport mechanism used for messaging is quite independent of the issue of messaging versus distributed objects, some concerns have been voiced over the use of UDP and multicast in the EVLA Monitor and Control software.

The main issue with respect to the use of UDP seems to be that it does not guarantee delivery of the datagram. UDP packets can be dropped, with no notification to the sender and no retransmission of the dropped packet. EVLA monitor and control software takes two approaches to this issue. First, the EVLA monitor and control network is a fiber-optic, full-duplex, switched network. This statement means that each and every node on the network has a separate point-to-point connection to a switch for both the receive and transmit paths to and from the node in question. In theory, that the network is full-duplex with each node connected directly to a switch should guarantee that there will be no collisions, and, as long the packet rating of the switch is not exceeded, there should be no dropped packets. However, theory and practice are never perfectly matched. While it may be true that there will be no collisions, for a number of reasons packets will be dropped. Use of a full-duplex, switched network, with the bandwidth available on each segment of the network chosen to exceed the expected traffic on that segment, will minimize but never completely eliminate dropped packets.

The second step taken to address the issue of dropped packets is to design the software to be tolerant of packet loss. Two examples are offered to illustrate this point. First, with respect to command delivery, the EVLA monitor and control software entity that sends commands to an antenna will also receive monitor data from that antenna and compare the commanded state to the reported state. Discrepancies between the two states will be reconciled by retransmission of the commands needed to bring the state as reported by the antenna into compliance with the commanded state.

The second example concerns alerts transmitted as multicast datagrams. Originally, an alert-on message was sent only once, when a device entered the alert state, and an alert-off message was sent only once, when a device exited the alert state. It was found that in a few cases alert-off messages could not always be matched with alert-on messages event though direct query of the device showed that the device had exited from the alert state. Presumably, but not conclusively, the mismatch can be attributed to dropped packets. It has now been decided to include the alert state of each and every monitor point in the multicasts of the monitor point values that are sent on a periodic basis to the archive, to screens, and to software processes performing operational activities. This method should prove quite robust in the presence of dropped packets and make the detection of alert-on and alert-off states easy and straightforward.

A reasonable question is, if the use of UDP requires extra effort why not use TCP/IP. The reply is that the steps taken are not extra effort, and, in the context of a real-time system, TCP/IP has a number of disadvantages that are best avoided unless the use of a TCP/IP based approach can be demonstrated to have significant advantages in other respects. That TCP/IP would guarantee the delivery of commands to a network stack and a software application associated with a device in no way guarantees that those commands actually reach the correct hardware registers in the lower levels of the device or that the software and hardware in the device will act on the command in the expected manner. In the monitor and control system it will always be necessary to monitor the reported state of the antennas, compare it to the commanded state, and take steps if the two states do not match regardless of the transport or protocol used to deliver commands. Similarly, the use of TCP/IP to deliver alerts in no way addresses the issue of broken or stalled connections, abnormal termination of the receiving processes, or a host of other conditions, other than dropped packets, that would prevent actual delivery of the alert. Sensible real-time system design dictates the use of methods and techniques that allow system functionality to be robust and recoverable in the face of these possibilities.

TCP/IP does complicate real-time system design in a number of ways. TCP/IP is a connection-oriented protocol, and the sending process blocks on the send until it receives an acknowledgement that the message has been received. That it is a connection-oriented protocol means that connection management will be required. It can be difficult, time-consuming, and complex to break and reestablish a TCP/IP connection that, for any number of reasons, has stopped functioning. Often, the necessary corrective actions cannot be accomplished in a purely automated fashion, by software running in the system. Human intervention may be required, sometimes taking the form of manually aborting and restarting one or both processes at the endpoints of the connection. Connection management is undesirable in a real-time system if a simpler approach is available.

That TCP/IP is a blocking protocol adds complexity. First and foremost, blocking has the potential to destroy timing in a real-time system. Consider the situation of the software entity responsible for sending commands to VLA and EVLA antennas. If UDP is used, the command(s) for any one antenna are sent and the software moves on to the next antenna. If TCP/IP is used, the command is sent and the software processes blocks on the TCP/IP send awaiting acknowledgement. An unknown number of retries may be required, and, if the connection is down, the software process may hang. For the TCP/IP scenario, determinism is decreased or destroyed. For the case of a hung connection, the observation, or some portion of it, may be lost. For TCP/IP, to get a non-blocking return, one would be required to generate a new thread either for each message or have a series of persistent threads, one per antenna. This requirement leads, in turn, to the need for additional code that checks on the status of the threads to insure that the system is not swamped by hung threads. While the multiple thread scheme outlined for the use of TCP/IP is potentially workable, it is to be preferred over UDP only if the use of a TCP/IP-based approach offers advantages that offset the increased complexity.

The EVLA Monitor and Control group is using TCP/IP in some contexts. In particular, it appears promising in the context of exploiting web-based technologies as the basis of user interfaces to some software processes. Some consideration is also being given to the use of HTTP, which is TCP/IP based, for the communication of antenna commands at some levels in the system, although perhaps not at the level of final delivery of commands to the hardware. For now, the method of choice is UDP. UDP is not an unreasonable choice. That it is connectionless and non-blocking reduces system complexity and helps to maintain determinism. Adoption of a more complex approach is justified only if that approach offers some compensating advantage. It remains to be seen if the use of TCP/IP based transfer of information offers advantages that outweigh the disadvantages.

The main objection to the use of multicast that has been voiced is that multicast is not a standard. Multicast is a capability that allows a single copy of a packet to be addressed to a group of nodes that have expressed a desire to receive it. It depends upon routers and switches within the network to forward the packet to the networks containing the receiving nodes.

The Internet Assigned Numbers Authority (IANA) has assigned a group of IP addresses to be used for IP multicasting. The assigned address range is 224.0.0.0 to 239.255.255.255. Within that address range, certain addresses or ranges of addresses have been reserved. For example, the addresses 224.0.0.0 through 224.0.0.225 are used by network protocols on local network segments. Network protocols use these addresses for automatic router discovery and to communicate routing information. 224.0.1.0 through 238.255.255.255 are globally scoped addresses. They can be used to multicast data between organizations and across the Internet. Some of these addresses are reserved for specific multicast applications. The Network Time Protocol (NTP) is a multicast application that uses the address 224.0.1.1. NTP is used by the MIBs in the EVLA antennas to initially set the time. The addresses in the range 239.0.0.0 through 239.255.255.255 are limited scope addresses. They are defined by RFC 2365 to be constrained to a local group or organization. And, so on, and so on. The point is that standards exist for multicast addressing.

The Internet Engineering Task Force (IETF) has also been developing a set of standards to address the issues of dynamic registration and multicast routing. RFC 1112 defines the Internet Group Membership Protocol (IGMP). IGMP specifies how a host may join a multicast group, leave the group, and provides a means for membership queries and reports. There are several standards available for routing IP Multicast traffic. RFC 1075 defines the Distance Vector Multicast Routing Protocol (DVRMP). RFC 1584 defines the Multicast Open Shortest Path First (MOSPF) protocol, and extension to OSPF that allows it to support IP Multicast. And, there are now two Internet standards-track drafts describing Protocol Independent Multicast (PIM), a multicast protocol that can be used in conjunction with all unicast IP routing protocols.

The standards cited are just a sample. They were chosen to illustrate that standards exist that specify the fundamental components of multicasting – addressing, group dynamics, and multicast routing. For a much fuller and more complete list of the relevant IETF documents and discussion of many other issues related to multicasting, a good web site is SWITCH, the Swiss Education and Research Network, http://www.switch.ch/network/ipmcast/references.html. Of course, one can also consult the IANA site, http://www.iana.org/, and the RFC archives, http://www.faqs.org/rfcs/. A google search on keywords such as “IP multicasting” produces a wealth of resources.

Finally, let it be emphasized, once again, that the material in this section is basically a digression. It is not germane to the basic issue of messaging versus distributed objects that is the focus of this document.

6.8 Recommendation

The ALMA Common Software (ACS) was considered for use in the EVLA Monitor and Control System. ACS is an example of what this document has termed the distributed object approach. ACS has many strengths. It supports strong typing and compile-time discovery of incorrect method invocations. The use of an Interface Definition Language (IDL) compiler provides a language-independent means of describing interfaces. ACS in particular has great strengths in dealing with a complex distributed system for which the deployment of and the interactions among components cannot be predicted in advance. ACS is a good fit to a project that has geographically dispersed software development teams working on components in parallel with similar, relatively tightly coupled timelines.

All of the team members, with the exception of Pete Whiteis, had some degree of exposure to the ALMA Common Software that is basis for communication among distributed objects in the ALMA system. At the time the team was formed, Bill Sahr had attended the three-day ACS Workshop held in Garching (March 2004). Chunai Cai had attended a 3.5 day course given in Socorro, had successfully installed ACS 3.0 on her workstation and had written and tested a number of components that used ACS to communicate. Rich Moeser, and Kevin Ryan, had taken a two-week ACS course given in Garching. Boyd Waters had attended a two-week ACS course in Garching, had done some work on porting early versions of ACS to compilers, and had authored a comparison of ACS to other methods of distributed object communication that was included in the document set prepared for the AIPS++ Technical Review.

The team feels that, for the general case, both the distributed object and messaging approaches are workable solutions for the problem of real-time control in a distributed system. However, the pertinent question for the EVLA is which approach best fits the staffing, timelines, engineering, scientific, and functional requirements of the EVLA. It is the conclusion of the team that the distributed object approach is a poor fit to or unworkable for many of the requirements that must be satisfied by the EVLA Monitor and Control System. The team members unanimously recommend that a message-based approach be used for the communications infrastructure of the EVLA Monitor and Control System. It should be clearly stated that acceptance of this recommendation means that the EVLA Monitor and Control System will not use the ALMA Common Software.

It should be further stated that since the interfaces between the EVLA on-line systems and the other components of the end-to-end software, other than the data archive, are sufficiently simple and straightforward, the use of a message-based approach for the EVLA Monitor and Control System does not preclude the use of ACS elsewhere in the EVLA software.