EVLA Architecture and Design

Snapshot #1: 2001-03-26

Contributors: Steve Blachman, Wayne Koski, Bill Sahr (ed.), Ken Sowinski, Boyd Waters

1. Introduction

This document presents a snapshot of the still evolving EVLA architecture and design document. It is very much a preliminary draft – many sections will be added, and the material already included will change. In its present form, it is a loose mixture of philosophy, requirements, architecture, and design. Its purpose is to stimulate discussion and comments that can then be used to refine and expand what is written here. Anyone who has contributed ideas that are included in this document is listed as a contributor, but responsibility for errors and/or misrepresentations lies with the editor.

2. Goals

There are several general goals that the architecture and design of the EVLA must attempt to fulfill. To date, the goals that have been articulated are:

- Software that achieves a high degree of hardware/platform independence
- Robustness w.r.t. modifications
- Extensibility
- Software reuse

Among these goals, hardware/platform independence is first among equals. What is meant by hardware/platform independence is the ability to change the underlying hardware with minimal impact on the software. The phrase "minimal impact" cannot be given an exact definition, and what is possible will vary from situation to situation. However, it is absolutely essential that the EVLA be able to upgrade the hardware used for its computing systems without requiring major rewrites of the software. The EVLA will be developed and deployed incrementally, over a period of many years. It is certain that some of the hardware and software technologies initially used will become obsolete and unsupportable over the course of further development and deployment. The EVLA must be able to incrementally upgrade and replace those obsolete technologies without perturbing the entire system. The EVLA must be able to continually adapt to changes in the marketplace, price/performance ratios, available capabilities, and user expectations without requiring major software rewrites to do so. A high degree of hardware independence in the EVLA software is a crucial, strategic component needed to address these issues.

A few examples to illustrate steps that can be taken toward the goal of hardware/platform independence may be useful. One of the most obvious points is the use of multiplatform operating systems and languages. For example, the use of VxWorks as the operating system in the real-time subsystems of the EVLA will help achieve a measure of hardware independence. VxWorks can use Sun Solaris, Windows NT, and RedHat Linux systems as a host platform, and the PowerPC, Pentium, MIPS, and ARM processor families as target CPU architectures. Use of VxWorks makes it possible to move within a CPU architecture, and to switch from one CPU architecture to another, while minimizing the changes needed in the application code.

The applications are written to the API presented by VxWorks, and that API is a constant across platforms. Use of Java is another obvious and often cited example. GUIs written in Java for Sun Solaris need not be totally rewritten to run, for example, under Windows NT. Yes, some coding changes and additional debugging may be required, but the time needed to implement the same GUI for a new platform is substantially decreased. Yet another example might be the software library created to handle monitor and control requirements within an antenna. If the software library for communication among modules consists of a few primitives such as read and write, with a few, generic parameters such as the number of words to transfer and a starting address, and if the library is layered to isolate and encapsulate the code specific to a particular fieldbus, then the underlying hardware can be changed without requiring changes in the software which uses the library primitives. The layer of the library that interacts directly with the fieldbus must be rewritten, but as long as the API seen by the applications remains unchanged, the applications that use the library do not require modification.

The need for robustness in the face of modifications and extensions to the system seems obvious, but a few comments may be merited. The issue here is one of degree. Hopefully, the EVLA will be a continually evolving entity not for the 5 or 7 years that pass during development, deployment, and refinement of its capabilities, but for the 20 or more years that will mark this phase of the life of the instrument. As initially deployed, the EVLA will use only the same 27 antennas now used by the VLA. However, the software must be written in a manner that allows the EVLA to be extended to include eventual use of VLBA antennas and the hoped for new antennas that will comprise the New Mexico Array. These extensions are stated goals of the EVLA project. Clearly, significant modification and extension of the baseline code and computing systems will be required to achieve these goals. The EVLA software and computing systems must not become fragile under the impact of the required changes and extensions to the initially deployed system. The decision to use object oriented analysis, design, and programming techniques for the EVLA was made, in part, with these considerations in mind. Object oriented techniques, if correctly applied, can bring a higher degree and new forms of modularity to a software system. That modularity, in turn, contributes to the robustness and extensibility of the system. Interfaces can be used to isolate the impact of changes in subsystem implementations. The use of commercial-off-the-shelf (COTS) hardware and software can also make important contributions. For example, the use of Ethernet and standard protocols for communication among the components of a distributed system provides a well-defined and relatively inexpensive upgrade path, and a standards compliant API to which applications may be written. The use of commercial or open source middleware for communication among functional units in a distributed system not only substantially reduces the in-house effort required for a portion of the software development, but can also greatly enhance the degree to which components of a system can be modified, relocated, and extended without requiring extensive changes in other components of the system.

Software reuse can be a thorny and misunderstood issue. Managers sometimes tend to treat it as the proverbial silver bullet that will cut development time, costs, and manpower requirements. Developers tend to view it as an infringement on independence and creativity. Both positions are probably wrong. It is simplistic and impractical to interpret software reuse as consisting solely or in large part of the literal reuse of coding. The first, best, and most frequent example of software reuse is the reuse of software analysis and design elements. Most software and system developers welcome the opportunity to study the designs developed for projects similar to the one on which they are working. Time can be saved, and pitfalls can be avoided, but the literal reuse of unmodified code from one project in another is often impossible. Reuse of analysis and design is already happening for the EVLA. Members of the EVLA design group have already visited Green Bank for the purpose of studying the design and code that was then being developed for the Green Bank Telescope. There is little doubt that there will be additional visits and exchanges of information. Computing staff members working on the EVLA have attended and will continue to attend both informal and formal presentations of various aspects of the software being developed for ALMA, and many of the computing division staff who are involved with the EVLA subscribe to the ALMA Software Announce mailing list, and read the ALMA documents announced via that list. Recently, it was decided that the ALMA use cases document will serve as the starting point for a similar effort for the EVLA.

To say that the literal reuse of unmodified code is often impossible is not to say that code reuse does not happen. If analysis and design converge, then code from one project may serve as an excellent starting point for the code used in another project. If, for example, the manager object used in the Green Bank Telescope is a good fit for some class or classes of problems in the EVLA, there would be no good reason not to use the code for that object as a starting point for the code that addresses the same or similar issues in the EVLA. However, it is likely that the Green Bank code would be a time saving starting point, rather than a final destination. For a variety of good and telling reasons, such as differences between the two classes of problems and/or the development of new software technologies with new capabilities, substantial modification of the Green Bank code might be required. For example, the use of RPC++ as a method of communication among distributed objects might be replaced by CORBA, and/or the parameter class or classes might be rewritten to use XML. These coding changes, if they proved to be productive, might then serve as a starting point for future changes in the Green Bank software. Literal code reuse is not only impossible, but to be avoided. Past mistakes would remain uncorrected and present opportunities for new capabilities and standards compliance would be missed.

Of course, there is another type of code reuse that has not so far been considered. “Reuse” is the wrong term. Substitute the word “use”. The design group working on the EVLA has no qualms about the use of AIPS++ and glish in the EVLA software. Indeed, the use of AIPS++ and glish is viewed with relief. The issues addressed by the

considerable and significant functionality being developed by the AIPS++ group removes a large burden from the EVLA development effort.

3. System Overview

Figure 3-1, the EVLA Computing System Conceptual Diagram, is neither architecture nor design, but may serve to help orient the reader. It shows the EVLA to be conceived as a heterogeneous, distributed system, i.e., multiple CPUs connected by a network. It is heterogeneous because the CPUs will not all be of the same type or family, and because the various platforms connected to the network will not all run the same operating system.

Starting at the top left of the diagram, the various subsystems of an antenna are shown as connected to an antenna controller via a fieldbus. The antenna subsystems will be custom devices, developed by NRAO. It is possible that the more complex subsystems will contain microprocessors and may include a small real time kernal. Hopefully, most of the software for these subsystems will be written in a higher-level language such as C rather than assembler.

The antenna controller will be a COTS general-purpose computer. Likely candidates for the computer are either a Pentium or PowerPC CPU in a VME or CPCI crate. For reasons of costs, it may be desirable to look at ruggedized PC hardware as a candidate for the antenna controller. The likely OS for the antenna controller is VxWorks. The antenna controller and possibly some of the antenna subsystems will include ports for connection of a laptop computer to support work done at the antenna.

The fieldbus may or may not be a COTS device. Likely candidates for the fieldbus are CAN, a modification of the NRAO-developed MCB, and Ethernet running TCP/IP. CAN is to be used by ALMA, and the MCB is currently used by the VLBA, the GBT, and NRAO's Orbiting VLBI ground station. As noted earlier, care must be taken that communications software developed for the fieldbus isolates hardware dependencies and presents a generic API to applications using the fieldbus (clients) that allows the fieldbus to be replaced without requiring a rewrite of the client applications.

The antenna controller will communicate with the array controller in the control building via a conventional network, probably gigabit Ethernet, running conventional protocols, probably TCP/IP & UDP. The physical connection between the antenna and the array controller will be fiber optic cable.

The combination of antenna subsystems, fieldbus, antenna controller, and array controller defines the path for monitor and control of the antennas.

In addition to the connection between the antenna controller and the array controller, a

connection is shown between the samplers in the antenna and a filter bank in the control room. This connection is the path for the raw scientific data. The physical connection will

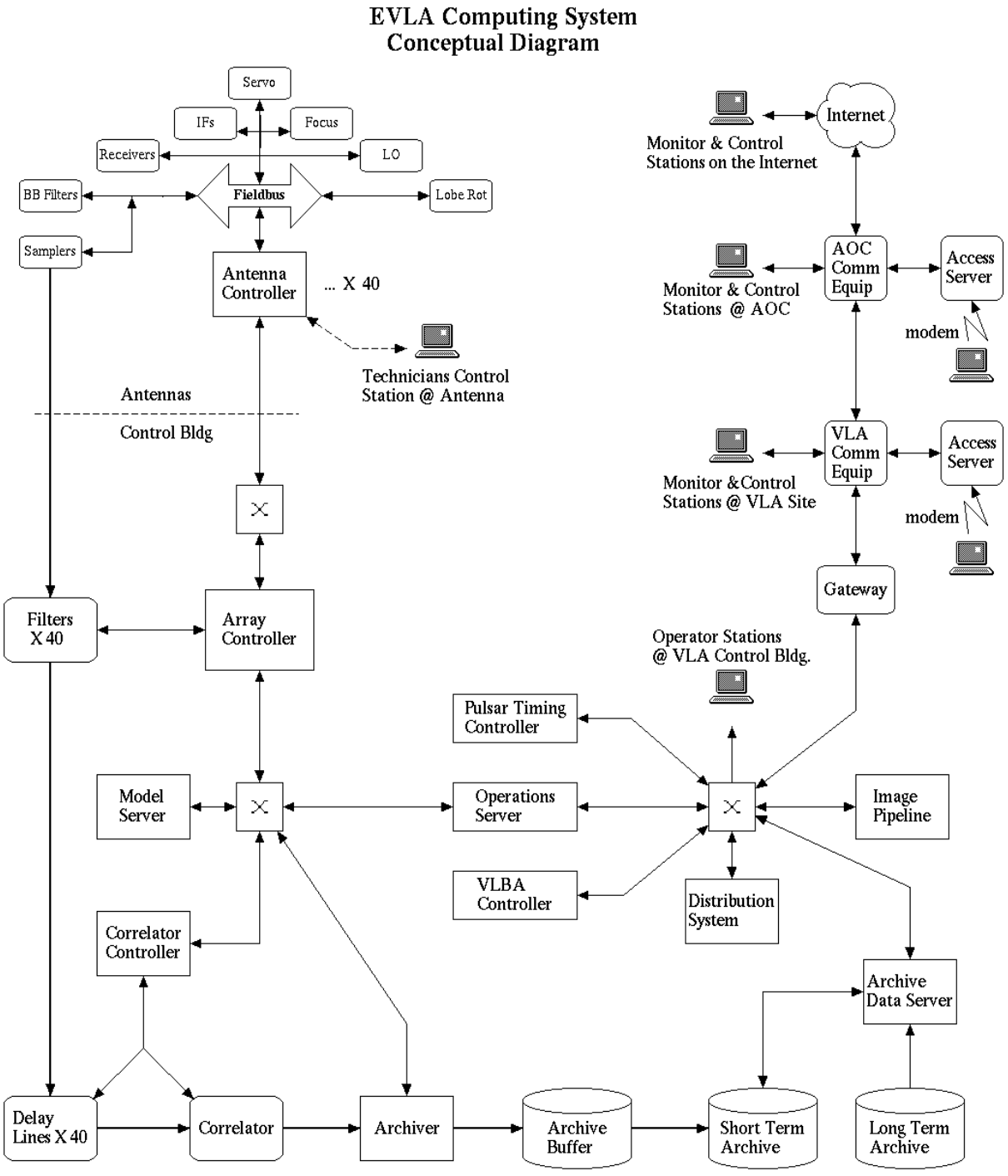


Figure 3-1 EVLA Conceptual Diagram

be fibre optic cable, however this path will not use conventional network protocols.

Most of the hard real-time control tasks will be contained in the cluster of devices shown in the lower left hand portion of the diagram - array controller, model server, correlator controller, and the correlator. The monitor, control, and data interfaces to the correlator are still being defined. To the degree that the other components are distributed, they will be interconnected via a conventional network running industry standard protocols, probably gigabit Ethernet running TCP/IP. Communications within this cluster will be isolated from other traffic, probably by use of a switched network.

The right hand side of the network is devoted, chiefly, to showing components of the system whose functions are related to control on human timescales - operator stations located at the VLA control building, other buildings at the VLA site, the AOC building, and locations connected via the internet. That the ability to monitor and control the array is to be possible over all of these domains will necessitate a very thoughtful approach to the issue of user interface software. It is strongly to be desired that only one suite of user interface software be written and maintained. Across these domains, the expected platforms will be Sun Solaris, Linux, and some variant of Windows. Some combination of an Internet browser with downloadable java classes, and/or a java "fat client" which must be installed on a user's system are strong possibilities. The need to support ASCII terminals is to be avoided. Currently, an ASCII based control program is used by both the VLA operators and by off-site personnel who need to view the status of the array. It may be necessary to insure that NRAO employees who may need to log into the site from their homes for first-order troubleshooting have the hardware needed to run the graphical control software. In the long run, money spent on such equipment will be more than recovered by the elimination of the need to develop and support multiple versions of the user interface to the array.

The center of the diagram shows an operations server, and two high-level controller devices. The operations server links the hard real-time portion of the system to the domain of human interactions with the array. That portion of the system that has knowledge of observing strategies and operations will reside in the operations server. As for the pulsar timing controller and for the VLBA controller, all that is meant by their inclusion in the diagram is the fact that the system must be designed to accommodate special purpose "engines" which can drive the array according to a set of needs peculiar to their purposes. The diagram shows these engines as not connected directly to the hard real time cluster to emphasize the need to keep the hard real time portion of the system isolated from the impact of non-realtime traffic.

Finally, the bottom and lower right hand side of the diagram is devoted to the archive and to the image pipeline. "Archive" is a term that is not yet defined w.r.t. the EVLA.

Portions of it will be memory resident, yet other portions stored on disks, other portions on tape, and the total archive may eventually span multiple physical locations. Data to support current operations will be drawn from the archive. The task of the image pipeline is to produce first order &/or default images, on a timescale of minutes, that can be used by the observer both to refine further observations to be made during the current observing run, and as the scientific results of the observation. The image pipeline will be implemented using AIPS++.

EVLA software will be designed and written using an object-oriented approach. Middleware will be needed for communication among objects that are running on different processors. Corba, NDDS (a real-time publish-subscribe system), and solutions using XML are all being considered. Middleware to handle communication among distributed objects is a good candidate for a COTS solution.

4. Software Layering

Figure 4-1 presents a diagram of the high level layering of the software for the EVLA software. Again, the archive must be considered a currently unresolved object that plays a central role. What the diagram does show is a separation of functionality into distinct layers, with defined paths of communication among the layers. The goal will be to write the EVLA software in a manner that adheres to the layering as strictly as possible.

An observer interacts with the system via the boxes labeled “Programmatic UIs” and “Observing Languages/Scripts**”.** These layers include items such as monitor and control GUIs, card files produced by JOBSERVE, output from a dynamic scheduler, goal-oriented descriptions of an observing run, and GUIs/programs used by engineers and technicians. With the exception of the technician and engineering GUIs, these entities will communicate only with the Observing System layer. It is in the Observing System layer that objects with knowledge of observing and observing strategies will reside. Examples of Observing System knowledge would include methods for reference pointing, strategies for mosaicing, special considerations relevant to solar observing, etc.

The Observing System Layer communicates with the Control System layer. Control system objects will understand devices. Communication with actual devices will be via only those objects that reside in the Control System layer.

A box labeled “Image Processing” is also shown in Figure 4-1. This box represents the Image Pipeline. It is shown as having connections to those layers used to drive the array. Those connections represent the requirement that feedback from the image pipeline will be used in conjunction with goal-oriented observing to drive the array toward satisfaction of the stated observing goals.

VLA Expansion Software Layers

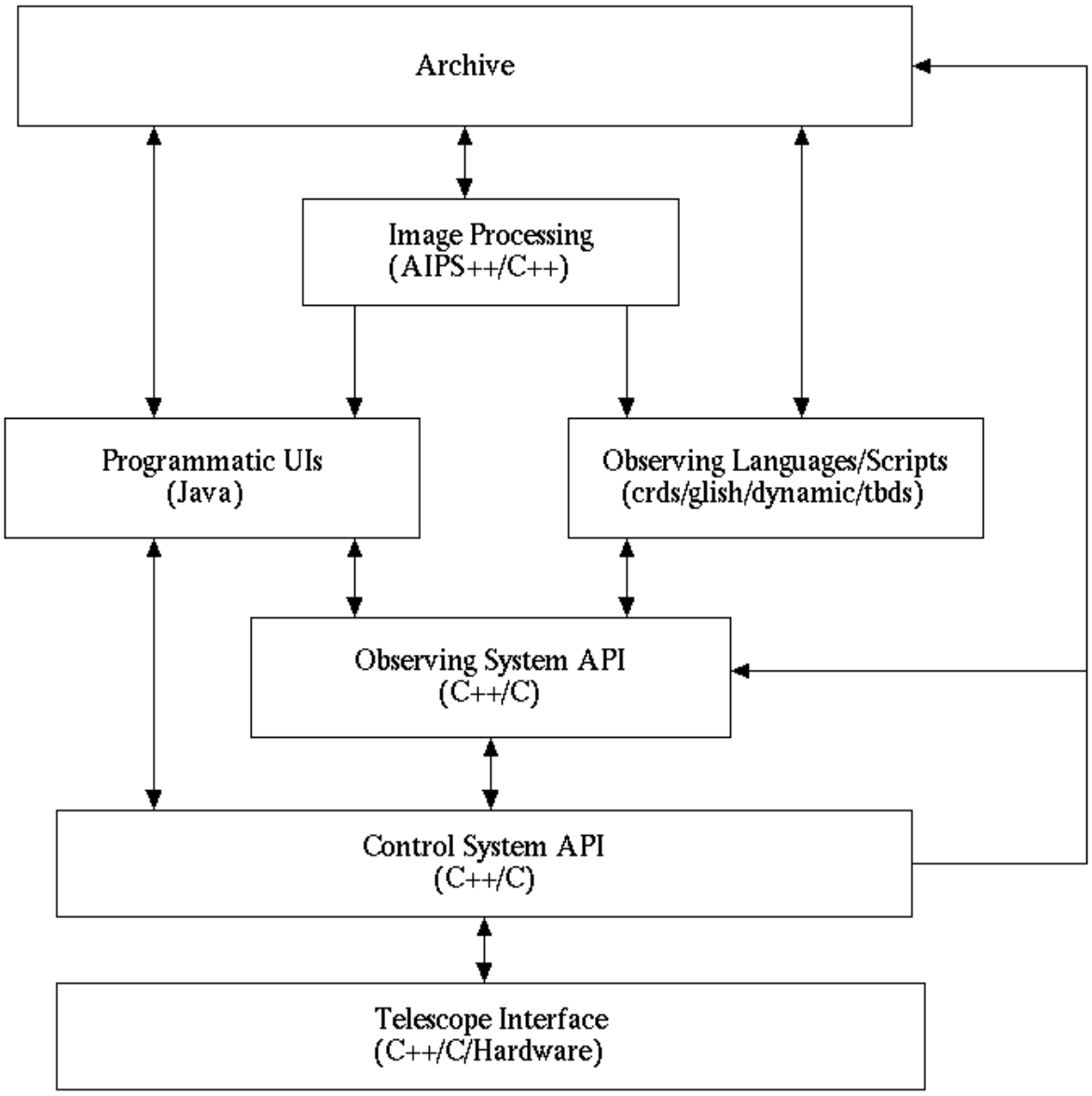


Figure 4-1 EVLA Software Layers

5. The Antenna Object

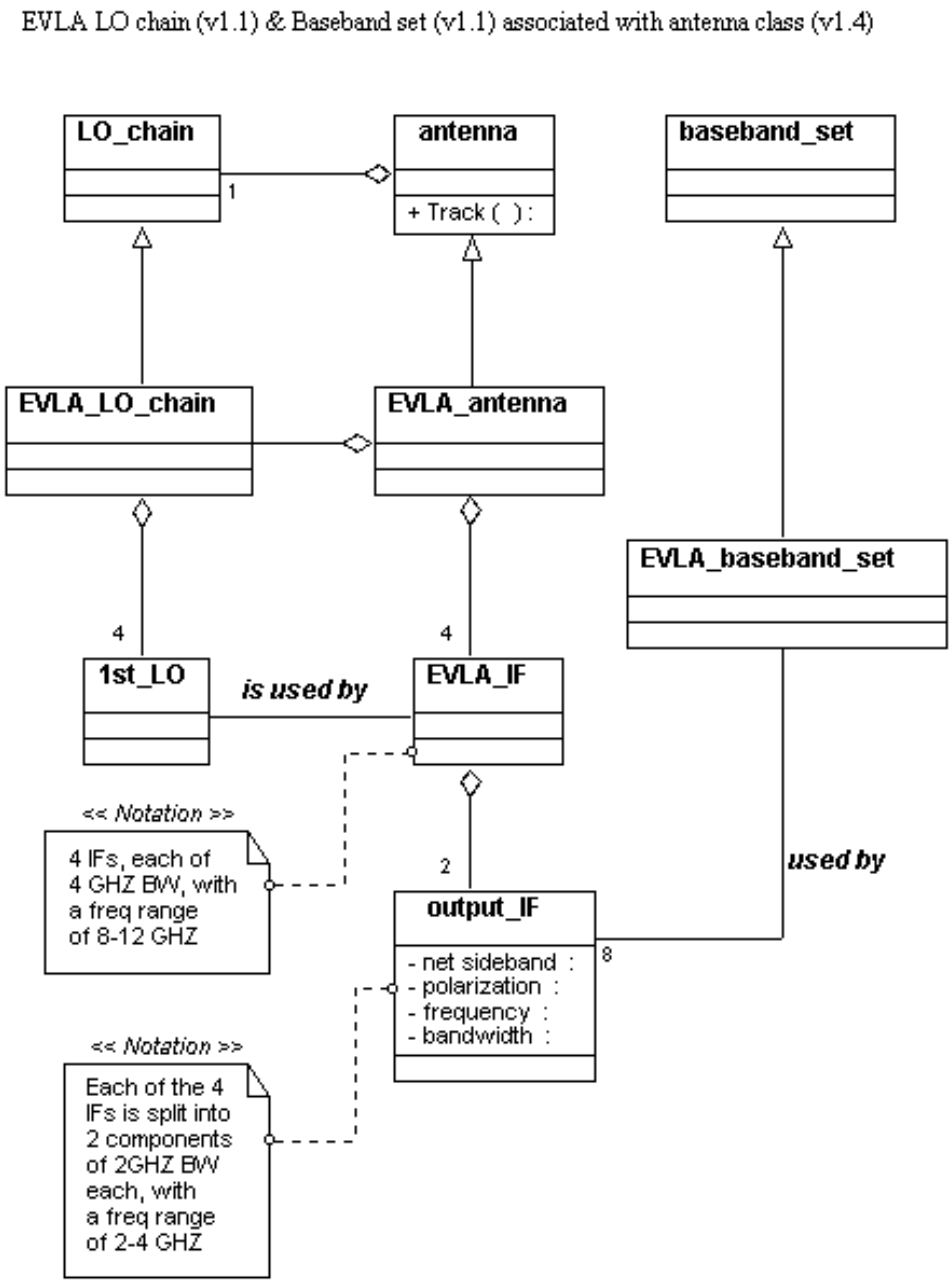
Over the life of the project, the EVLA will be required to support as many as four different types of antennas –

- The current VLA antennas
- Upgraded VLA antennas
- VLBA antennas
- New antennas for the New Mexico Array configuration

Support for multiple antenna types is an EVLA requirement, and the strategy used to satisfy this requirement must be an integral part of the EVLA architecture and design. The definition, creation, and use of antenna classes and antenna objects is seen as central to providing the ability to monitor, control, and acquire data from different types of antennas.

It would be highly desirable for the observing system to be able to use different types of antennas without requiring the observing system to have detailed knowledge of the special characteristics of each antenna type. For this goal to be achieved, all antenna objects must present the same API to the observing system. The goal is not to hide the type of an antenna and its special characteristics. Rather, the goal is to encapsulate the differences among antennas within the antenna classes. All antenna classes will inherit from a common base class. Differences among types of antennas will be contained in the derived classes, and in the derived classes associated with an antenna. For those cases where detailed knowledge of the specific capabilities of an antenna is needed, query interfaces can be provided. Figure 5-1 illustrates the point being made. It shows base classes for an LO chain, an antenna, and a baseband set, from which classes specific to the EVLA have been derived. Please note that this diagram is meant only for illustrative purposes.

At this point in time an antenna is not conceptualized as a collection of devices. (An "is-a" relationship.) Rather, an antenna is viewed as "having" devices. This distinction translates, in terms of UML, to defining an antenna as an aggregation of component parts rather than as an entity composed of parts. Aggregation is a less stringent relationship than composition. This distinction is being made because it may have an impact on code design and implementation. This same distinction may be applied to the correlator.



.

Figure 5-1 Base & Derived Classes

6. Antenna Objects and the Hybrid System

Figure 6-1 illustrates how antenna objects might fit into a hybrid system, i.e., a system composed of old, unmodified VLA antennas, upgraded VLA antennas, and even VLBA antennas. Functionally, antenna objects are sources for monitor data and sinks for commands. For the case of the EVLA, the antenna objects will almost certainly run in the computers identified as "Antenna Controller(s)" in the conceptual diagram presented in Figure 3-1, i.e., the antenna objects will reside in computers that are located in the antennas, and will communicate with devices in the antenna via the fieldbus. This arrangement is not possible for unmodified VLA antennas because there are no computers in the antennas. For the unmodified VLA antennas, the antenna objects could run in a computer or computers that are positioned between the observing system and the serial line controller. The VLA antenna objects would obtain their data from the serial line controller, but still serve as sources of monitor data with respect to the observing system. Similarly, the VLA antenna objects would serve as sinks for commands from the observing system, forwarding the commands to the actual antenna via the serial line controller. In effect, the serial line controller would be the "fieldbus" for the VLA antennas. VLBA antennas are still something of an unknown. For the VLBA antennas, the antenna objects could run in a computer(s) in the control building, or in a computer at the antenna, or both.

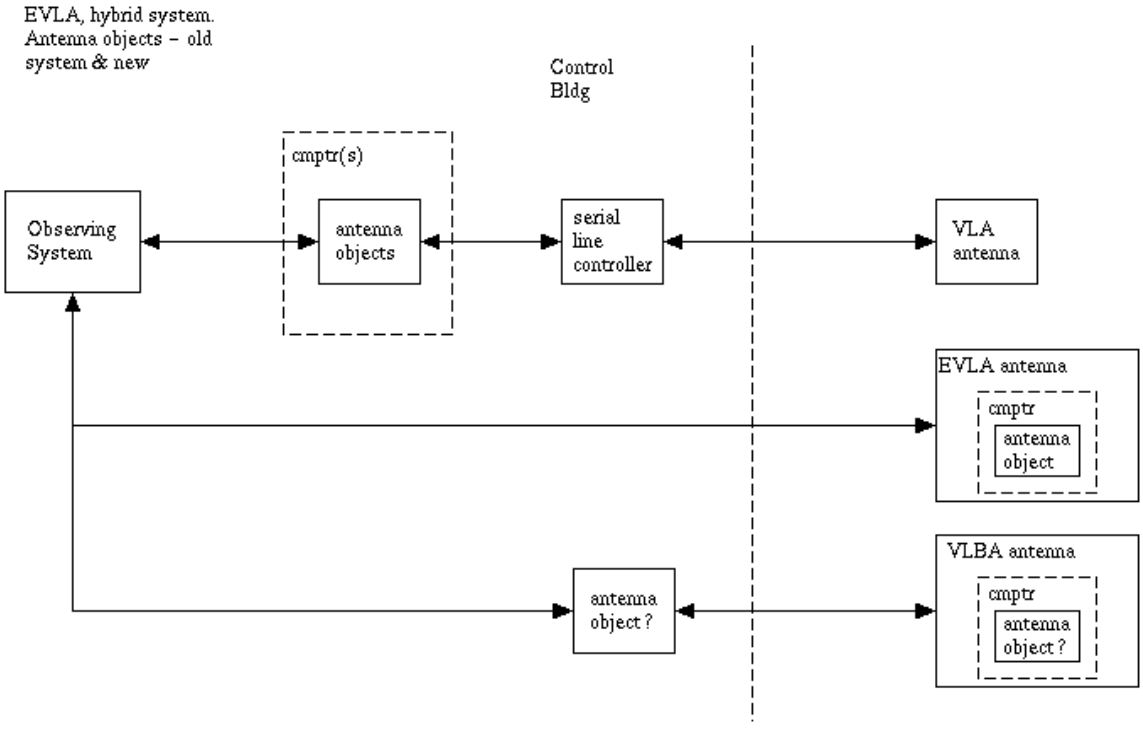
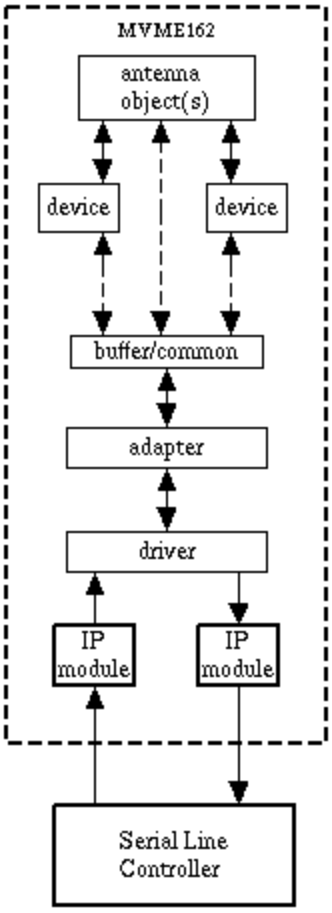


Figure 6-1 Antenna Objects, Hybrid System

6.1. Serial Line Controller Interface

The Serial Line Controller Interface (SLC/IF) is a project currently under development. Its purpose is to provide the means by which antenna objects can be connected to unmodified VLA antennas. The figure to the left (Figure 6-2) gives one possible architecture for the combination of antenna objects, the Serial Line Controller Interface, and the Serial Line Controller. The MVME162 is the CPU board being used for development of the Serial Line Controller Interface. The CPU board, the IP modules, and the Serial Line Controller are hardware. All other elements are software entities. Before proceeding to a more detailed description, it should be noted that the Serial Line Controller used at the VLA is a dual ported device. One of the two ports is currently unused, allowing the SLC/IF to be connected to the Serial Line Controller with no impact on VLA operations.



Starting, more or less, in the middle of the diagram, the buffer/common receives monitor data from the Serial Line Controller, and commands from the antenna object. The format of the buffer is oriented toward the needs of the antenna object. The adapter handles the format requirements of the driver and perhaps other issues.

The devices, which are shown as "attached" to the antenna object, will contain methods for all R/W parameters, and will encapsulate the knowledge of device behaviors and peculiarities. The devices are also the point at which "pseudo registers" used to remember commands sent to devices containing write only registers will be implemented. Devices will communicate with antenna objects. It has not yet been

Figure 6-2 Serial Line Cntrllr IF

decided if devices will communicate directly with the serial line controller interface via the buffer/common. Devices may be restricted to access to the buffer/common only via the antenna object.

At this time, hardware development and the software driver for the Serial Line Controller Interface is nearly complete. The ability to obtain monitor data from the Serial Line Controller has been demonstrated. No attempts have yet been made to send commands. It is likely that development of the software for the SLC/IF beyond the stage of a driver will be stopped until an antenna object has been developed in sufficient detail

to allow specification of the remaining components. It is unlikely that the EVLA will simply duplicate the memory resident database of monitor data as currently used by the VLA. The issue must be re-examined. It is entirely possible that the current format used for monitor data will not map well into an object-oriented system.

7. Middleware

Middleware can be loosely defined as software that handles some or all of the work required to achieve communication among elements in a distributed system. For an object-oriented system, middleware would be responsible for some or all aspects of method invocation on remote objects. In a publish-subscribe system, middleware would handle the distribution of publications to subscribers. There are a number of characteristics that are desirable in middleware. A list of such characteristics would include:

- Elimination of the need for software developers to write networking code.
- Location independence, i.e., objects or entities are not required to know the location of other objects or entities in order to communicate with them.
- Low latency, with low variability. A definition of latency is to some degree context dependent, but it could be roughly defined as the total elapsed time between a method invocation and the response or the total elapsed time to transport a publication from publisher to subscriber.
- Adequate throughput
- Rollover/failover features. This characteristic would include provisions to detect network failures, to provide alternate suppliers of a service or publication, and to gracefully handle the dropout and addition of nodes.

So, what are the subsystems within the EVLA that could make use of middleware, and what are the characteristics of the communication needed by these subsystems? Referring back to the conceptual diagram (Figure 3-1), and starting at the antenna –

- The antenna controller and the antenna subsystems must exchange monitor and control data over the fieldbus.
- The antenna controller and the technicians control station, connected at the antenna, must exchange monitor and control data.
- Monitor and control data will be communicated between the array controller and each of the antennas.
- The model server will send information to the array controller (and thence to the antennas), the correlator controller, and to some element of the archive system.
- The array controller and the correlator controller will exchange monitor and control data with the operations server.
- Monitor data, and perhaps control data will be sent to some element of the archive.

- Ancillary subsystems (represented by the pulsar timing controller and the VLBA controller in the conceptual diagram) will exchange monitor and control data and probably other forms of data with the operations server.
- The image pipeline will exchange data and control information with the operations server and some element of the archive.
- Operator stations at the EVLA Control Building, at other locations on the VLA site, at the AOC Building in Socorro, and at other locations around the world will exchange monitor and control data with the operations server, and data of many types with the archive server.

Notable omissions from this list are the transmission of visibility data from the antenna, and the output of data from the correlator. It is unlikely that middleware will have a role to play in these particular data paths. There is simply no reason to add the complexity and additional overhead. They are likely to be straightforward in the sense that they will be point-to-point connections.

Reviewing the list of subsystem communication exchanges for which middleware might be useful, it is clear that the exchange of monitor and control data is the most frequently occurring element.

There are two items in the list that involve user interfaces – the technician control station connected at the antenna, and operator control stations that interact with the operations server and the archive. Will middleware be used as a component of user interfaces?

For the technician control station, the issue is the functionality of the station. Will it connect directly to the antenna controller, directly to the fieldbus, and / or directly to each of the subsystems individually? Obviously, middleware would serve a useful role only for those connections that see complementary software on the other end of the connection. If the antenna controller uses Corba or real-time publish-subscribe, then the connection between the antenna controller and the technicians control station must also do so if the use of a special port and / or special software for antenna controller access is to be avoided. However, that requirement does not speak to the possibility of a direct connection to the fieldbus or to each of the antenna subsystems. It is conceivable, but unlikely that the individual subsystems of the antenna will run a middleware package. If they do not, then test / diagnostic software that has been tailored to each subsystem must also avoid the use of middleware. These requirements appear to conflict with one another – a connection to the antenna controller which does require complementary middleware and subsystem test suites which do not. If the fieldbus is a true network, all that is really needed by the technician control station is a connection to the fieldbus. The technician control station can then communicate with the antenna controller, using the same middleware, if any, that is used by the antenna controller, and with the antenna subsystems using whatever scheme is used by those subsystems to communicate with the antenna controller.

If, in the course of developing the antenna subsystems, special ports (non-fieldbus) running software used for development, debugging, and diagnostics are created, then the technician control station should be equipped with hardware capable of connecting to this port, and with the software that uses this port. Two implications follow – 1) the special ports must use a protocol / standard which can be installed / implemented on the technician control station (probably a laptop computer) and 2) in accordance with the system requirements stated in the NSF proposal for the VLA Expansion Project, the functionality of the software which makes use of the special port must be duplicated in the software which uses the fieldbus. Section 6.3.2 of the project proposal, entitled System Requirements, in the 2nd paragraph of the subsection entitled "Antenna Monitoring and Control" (p. 70) states "No separate, privileged interface to the electronics will be developed. All monitor and control capabilities needed to test and support a device should be available over the standard monitor and control interface."

For the operator stations, the issues are somewhat different. One balks at the thought of requiring software that includes Corba or a proprietary real-time publish-subscribe system for Internet access to the system. Corba is very complex, and proprietary software raises the issues of distribution, licensing, and fees. There is also the issue that the machines used for network / Internet access may be running Solaris, Linux, varieties of Windows, or products not yet anticipated. It is not known that middleware which runs on all of these platforms and that also satisfies other functional requirements could even be found. Even if such a product does exist, will it continue to be a viable technology for 10 or 20 years, will it be extended to new platforms over time, etc, etc, are all questions with unknown answers. Additionally, multiplatform support for both installation and versioning would be required. Attempting to support middleware-based user interfaces, for remote access via a variety of platforms would seem to be a major mistake. The required support would soon become a huge and endless timesink. This logic pushes one to the conclusion that network and Internet access to monitor and control data, and to the archive be pushed into the arena of Internet-mediated communication that uses generic methodologies that exist in the commercial marketplace. Currently, there are two approaches to this issue – the J2EE (Java 2 Enterprise Edition) framework, and the .NET framework. J2EE is a single language, multiple platform approach, while .NET is a multiple language, single platform approach. It is likely that J2EE will be the framework of choice, but the development of .NET must be monitored for possible expansion to non-Windows platforms.

In the foregoing paragraph no distinction is made between operator stations used by the VLA Operators and operator stations at remote locations. This lack of such a distinction is deliberate. The current plan is to use the same software suite for both purposes. Suitable provisions for authorizing and revoking authorization of control functions must be provided. It would be desirable to develop some mechanism that would give the operators in the EVLA control building priority access with respect to all other operator

stations. The level(s) at which this issue can be most effectively addressed is currently unclear.

One interesting result of this discussion of middleware is that communication among the elements of the EVLA begins to be characterized as falling into two domains – one for which middleware is a relevant issue, and one for which it is not. Middleware, be it corba, publish-subscribe, XML messaging or whatever, has a role to play in the real-time elements of the system – the antenna controller, array controller, model server, and correlator controller, with the operations server demarcating a boundary at which middleware stops and the domain of network / Internet solutions begins.

7.1. Middleware Candidates

Discuss candidates for middleware: Corba (multiple orbs to get the language bindings), NDDS (proprietary, licensing issues), Java RMI (By then will it require Corba?), RPC++, XML messaging, other?

8. User Interfaces

8.1. Engineering Interfaces

Test suites which can be used during development, troubleshooting, and system maintenance must be developed for each device in the system. Development of this software will necessarily involve close collaboration among hardware and software engineers. The goal is write once and use everywhere - the same test suite can be run on an operator station, regardless of location and on a technician control station, again regardless of location.

The technician control station raises some interesting issues. One desirable goal is that the technician control station be useable in the absence of a functioning antenna control computer. If CAN or Ethernet is chosen as the fieldbus, one component of the solution is to provide a port on the fieldbus for connection of the technician control station. As discussed in the section on middleware, this connection will allow testing of all devices on the fieldbus, including the antenna controller. However, there is the issue of the addressing scheme to be used for devices. This issue will be considered in the context of Ethernet as the fieldbus, to help make the discussion more concrete. The question is what addressing scheme can be used that will satisfy the requirements of troubleshooting and maintenance? These requirements include dealing with the issues of swapping modules and replacing modules. For example, in each VLA antenna there are two L6 modules. Sometimes it is desirable to swap the modules, for example, to see if an intermittent problem follows the module. So, to what is the address used by test suite software anchored? The answer must be that the address used by the software to address a functional unit of a subsystem must be anchored to a location, not to the module. The

software that tests L6A and L6B must always look at the locations used for the L6A and L6B modules regardless of the particular board occupying that location. So, for the case of Ethernet, IP addresses and names must be anchored to locations with some method of mapping the MAC on the network interface to the proper IP address &/or name. This issue raises the larger issue of addressing within and across antennas. Wayne Koski estimates that from 20 to 50 devices will be present on the fieldbus within each antenna. So, how is the address space for some 30 antennas to be configured or partitioned? There are a number of possibilities. One that appears attractive, at this point, is to assign unique IP addresses to all devices, system wide, with all devices within each antenna on the same subnet. As an aside, the point has been raised that this approach may give rise to the opportunity for certain efficiencies. For example, a subnet specific broadcast message could be used to request that all devices in an antenna report their status.

If the device test suite software is written with careful attention to modularity it should be possible to include it in the antenna controller. Inclusion in the antenna controller of the same test suite software that is used in the technician control station would make it possible to run the same tests on any or all of the antenna subsystems without requiring the presence of someone at the antenna. (Assuming, of course that the antenna controller is up and running normally.) This capability would be very useful for first order troubleshooting from the AOC, or from someone's home.

9. Data Characterization

9.1. Visibility Data

The WIDAR correlator, as currently planned, will have 240 baseline boards. Each baseline board will be capable of an output rate of 100 Mbytes/sec. By simple multiplication, the maximum aggregate output rate of the correlator will be 24 Gigabytes/sec. The "aggregate" output rate is taken to be the summed output across all baseline boards. For 86,400 seconds/day, if the correlator were to run at its full rated output for an entire day, the result would be roughly 2 petabytes of data.

The initial specification for the EVLA is to accept data from the correlator at a maximum, aggregate output rate of 25 Megabytes/second. While that specification may sound modest compared to the capabilities of the correlator, it is roughly a X25 increase over the VLA and the current correlator. Ken Sowinski, in an email of 12/21/2000, provided a set of estimates and calculations that relate this figure of 25Mbytes/sec to the VLA and the EVLA.

Begin with the 27 antennas that comprise the VLA. For the 27 antennas, there are 351 baselines. Now suppose the goal is to handle 16 subbands of 1024 (2048 ?) channels each, for a total of 16,384 channels (32,768 ?). Again, this goal is modest in comparison to WIDAR's capabilities. It subdivides the entire 16 GHZ bandwidth per antenna into 16

subbands rather than making use of the correlator's ability to subdivide each 2GHZ baseband into 16 subbands. Multiplying 351 baselines by 16 subbands results in 5,616 1024-point real to complex FFTs per integration. The amount of data that must be moved is given by 351 baselines * 16 subbands / baseline * 1024 channels / subband * 8 bytes = 43.875 Mbytes of data. This amount of data must be accepted by the correlator backend processor, and then output by it, so the figure of 43.875 Mbytes must be multiplied by 2 for the total amount of I/O required per integration. The result is a total I/O requirement of 86.75 Mbytes. Call it 100 Mbytes to keep the calculations simple. Now, for the case of a 10 sec integration time, the backend processor will be required to perform the 5,616 1024-point complex FFTs within one integration period. Dividing the integration time by the number of FFTs gives 1.78 ms per FFT, and dividing the 100 Mbyte I/O requirement by the 10 seconds available results in 10 Mbytes/sec of sustained I/O.

The table given below illustrates the logic of the preceding paragraph for several integration times. The 1.78 ms per FFT is rounded to 2ms and then halved to allow for other work that must be done by the backend processor.

Integration Time	FFT time	I/O Bandwidth
10 sec	1 ms	10 Mbytes/sec
1 sec	100 microsec	100 Mbytes/sec
0.1 sec	10 microsec	1 Gbyte/sec

Table 1. Integration Time, FFT time, and I/O Bandwidth

For a system that accepts data from the correlator at a rate of 25 Mbytes/sec, a 10 second integration time is never a problem. A 1 sec integration time will require a reduction in the number of channels by a factor of 4. A 0.1 second integration time requires a reduction by a factor of 40. The total of 16,384 (32,678 ?) divided by the factor of 40 gives 409.6 channels, or, rounded to the nearest power of two, 512 channels per baseline to remain approximately within the I/O spec of 25 Mbytes/second.

25Mbytes/second if sustained for a full day would result in 2 Terabytes of data. This figure ignores slews and setup times during which no visibilities are produced, but it counts only visibilities and ignores other data product.

The next question to answer is what fraction of the time will the VLA run at the rate of 25Mbytes/second. For the A array configuration, only experiments which look at objects that vary quickly will require integration times less than 1 or 2 seconds. The integration time is set by time-smearing considerations and will shrink inversely with increasing baseline length. Widefield continuum mapping requires delta nu/nu to be less than approximately 1/10,000. This figure implies about 4000 channels at K band to cover 8 GHZ of total bandwidth; probably the worst case. 4096 channels at a 1 sec integration

time is about 25 Mbytes/sec. The conclusion is that the throughput goal of 25 Mbytes/sec supports all continuum work, and except in the A array configuration, it will never even come close to being a problem. The spectral line case will not be considered at this time, except to note that spectral line work (1/2 to 2/3 of the allocated time) is done in D array. Very little spectral line work (less than 1 day per month) is done in A array, and it is not obvious that the new correlator will change this fact. Observations at wavelengths longer than K band cannot provide a full 8 GHZ of bandwidth, so the 25 Mbytes/sec specification is adequate as long as there is no requirement to use the otherwise unused baseband pairs to produce additional channels.

The initial goals of the EVLA w.r.t. to the correlator capabilities are modest, but still an enormous improvement over the VLA. By the time the EVLA is deployed, and the correlator requirements are more fully understood, technology may have advanced to the point where phase II of the EVLA may be considered: 40 antennas, 780 baselines, antennas farther away by a factor of 10, requiring 0.1 sec integration times and 40,000 channels for untainted widefield continuum mapping.

A second approach to characterizing the correlator output is to apply a few simple estimates to the output rate. The month of October 2000 was used as a basis for the estimates. October has 31 days, which is 744 hours. During that month maintenance, startup, move/operations and software test time consumed 138.5 hours. Assuming that data was not taken during 95% of the time allocated to those activities, there remains a total of 612 hours during which data was recorded. October 2000 had more antenna moves than the average month, so the figure of 612 hours is biased in the direction of fewer hours than might be the average case.

Next, one must estimate the correlator output rates for an average month. No attempt was made to analyze the observing schedule. Instead, a completely arbitrary utilization model was used. The assumptions are 1) that the correlator will be run at the maximum initial spec of 25 Mbytes/sec for 10% of the time, at ½ the maximum initial rate 30% of the time, and at 1/5 the maximum initial spec for the remaining time (60%). The results of these assumptions are given below.

Data rate	# of hours per month	Data produced, Mbytes
25 Mbytes/sec	61.2 hours	5,508,000
12.5 Mbytes/sec	183.6 hours	8,262,000
5 Mbytes/sec	367.2 hours	6,609,600

Table 2. Correlator Output, per Month

The total for the month is 20,379,600 Mbytes = 19,901 Gbytes = 19.44 Terabytes. Dividing

by the 31 days in the month gives approximately 642 G bytes / day, or 26.75 Gbytes / hour.

The most likely conclusion to be drawn from these figures is that the usage model is incorrect. The attempt to develop a more realistic usage model will be deferred until the correlator, especially the spectral line modes, are more fully understood.

The calculations developed by Ken Sowinski, and the statistical approach are both "back of an envelope" types of estimates. As plans develop and knowledge increases, estimates that better reflect the realities of anticipated correlator usage will be developed. However, the estimates given in this section are of use in forming a sense of the size of the problem. Obviously, a very thoughtful approach must be taken to the issue of archiving the visibility data.

10. Future Sections

Many additional sections will be added to this document. The chief impediment to doing so has been time. Sections for which some amount of material has already been accumulated include

- The EVLA Network
- Data Characterization (data rates, data volume)
    - Monitor Data
    - Visibility Data (some initial work already done)
    - Other Data Products

Sections which are anticipated, but as yet undeveloped include

- Time, Timing, and Synchronization
- Time System and Representation
- Device Control
    - Antennas
- The WIDAR Correlator
    - Front End setup
    - Backend
        - Setup
        - Interface
- Correlator Operation and the Hybrid Array
    - The new correlator controller
- Other Subsystems
    - Weather
    - Site Interferometer
- Monitor Data
    - Antennas
    - Correlator
    - Other Subsystems

        - Weather
        - Site Interferometer
- Logging, Errors, Alarms
- Security
- Data Production
- System Tests and Tuning
- Image Pipeline
- Failure Analysis (Identification of points of failure, with contingency plans.)

Index