Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

The ALMA Software System

Joseph Schwarz, Heiko Sommer

European Southern Observatory, Garching, Germany

Allen Farris

National Radio Astronomy Observatory, Socorro, NM

Abstract. Prospective users, instrumentation and location of the Atacama Large Millimeter Array (ALMA) all present its software developers with major challenges. The development of this software will be distributed among many institutes on two continents, mimicking the software itself, which will have to function in a distributed environment, spanning the 0.5-10 km baselines between antennas, as well as the much larger distances that will separate the array site at the 5000m-high Llano de Chajnantor, the Operations Support Facility in San Pedro de Atacama, the Santiago Central Office, and the ALMA Regional Centers in North America and Europe.

To make distributed development successful, we have defined interfaces that allow separated groups to work independently of their counterparts at other locations as much as possible. We have defined a common architecture and infrastructure, so that work done at one location is not unnecessarily duplicated at another, and that similar tasks are done in a similar way throughout the project. A single, integrated Archive attends to the needs of all subsystems for persistent storage, and hides details of the underlying database technology. The separation of functional from technical concerns is built into the system architecture through the use of the Container-Component model: application developers can concentrate on implementing functionality in runtime-deployable components, which in turn depend on Containers to provide them with services such as access to remote resources, transparent serialization of value objects to XML, logging, error-handling and security. The resulting middleware, which forms part of the ALMA Common Software (ACS), is based on CORBA and XML.

1. Introduction

The Atacama Large Millimeter Array (ALMA) is a joint project of the North American and European astronomical communities, with Japan likely to join in 2004. It will consist of 64 antennas each 12 meters in diameter which will work as an aperture synthesis telescope to make detailed images of astronomical objects. They will be positioned as needed with baselines from 0.5 to 14 kilometers so as to give the array a zoom-lens capability, with angular resolution reaching 10 milliarcseconds. ALMA will represent a leap of over two orders of magnitude in both spatial resolution and sensitivity, making it ideal for medium scale deep investigations of the structure of the submillimeter sky. It will operate at an altitude of ~ 5000 meters on the Llano de Chajnantor in Chile's Atacama desert.

1.1. ALMA Schedule

The first of the 64 antennas is expected to be on site in 2006, followed by the first production receiver (4-band) in Q2 of 2007. A few months later, when 6-8 antnnas have been commissioned, early ALMA science operations will start; the full antenna complement will have arrived in early 2012, at which point ALMA will begin full science operations.

2. Software Scope

2.1. From the cradle to the grave... and beyond

The ALMA software needs to handle all phases in the life of an observing project, *i.e.*, 1) Proposal Preparation; 2) Proposal Review; 3) Program Preparation; 4) Dynamic Scheduling of Programs; 5) Execution of the Observations themselves; 6) Calibration & Imaging; and 7) Data Delivery & Archiving. Even after the data has been delivered to the PI, the work of the software is not done. The system must support Internet-based archival research once any proprietary rights to the data have expired. In this regard, the ALMA Science Research Archive will be one of the first to have VO Compliance designed-in from the start.

At the same time, the software is required to make millimeter interferometry accessible even to the uninitiated (while preserving the expert's ability to exercise full control). As ALMA's Science Software Requirements state:

> "The ALMA software shall offer an easy to use interface to any user and should not assume detailed knowledge of millimeter astronomy and of the ALMA hardware.
>
> "The general user shall be offered fully supported, standard observing modes to achieve the project goals, expressed in terms of science parameters rather than technical quantities..."

Of course, when complexity is swept under the rug for the sake of the neophyte user, the resulting lump remains. What is made simple for the user will therefore be complex for the software developer. One goal of the system architecture should be to relieve the developer of *unnecessary* complexity. Our primary vehicle for achieving this is the separation of functional from technical concerns, which is discussed below.

2.2. The numbers

ALMA's baseline correlator will produce ~ 1 Gbyte/s, which the software must Fourier-transform from the time to frequency domain and reduce to average/peak data rates of 4/40 Mbyte/s. The data that results from the later imaging of this raw *uv*-data will increase these figures by about 50% to 6/60 Mbytes/s, implying ~ 180 Tbyte/y to archive. (As the experience of the Hubble Space Telescope shows, Archive *access* rates could be a factor of ~5 higher.) The wish to support recent enhancements to the hardware of the baseline correlator has produced a proposal to raise these figures to 25/95 Mbyte/s, demonstrating the need for the software to be flexible and scalable enough to adapt quickly to dramatic changes in requirements.

Amidst this storm of incoming data, the online calibration software must be able to calculate pointing & focus corrections, phase corrections & average phase noise and feed these results back to the observing process in ~ 0.5 s, so that antenna pointing & focus can be adjusted in near-real-time and the time spent observing target and phase calibrator can be adjusted to match rapidly changing atmospheric conditions (Lucas 2004).

Once observations belonging to a given Observing Program are complete, the science data processing, *i.e.*, the production of images via calibration, Fourier transformation and deconvolution, must keep pace (on average) with the rate of data acquisition.

Another important number to keep in mind is the dozen or so institutes on two continents that take part in the development of software for ALMA. The fact that this first number exceeds unity by an order of magnitude conditions many of the architectural and process-related choices we have made, as we will discuss below.

Table 1. Run-time Issues

Challenge	Response
Changing observing conditions	Dynamic Scheduler
High data rates	Integrated scalable Archive
Diverse user community (novice to expert)	Flexible observing tool, GUIs
Distributed hardware & personnel	
AOS: antennas at 0.5-14 km from correlator	High-speed networks
AOS-OSF: operators are 50 km from array	Distributed architecture
OSF-SCO-ARCs: PIs, staff, separated from	CORBA & CORBA services
OSF by 1000s of km, often by many	Container/Component model
hours in time zone	XML serialization

3. Development Approach

3.1. Separation of Concerns

Expressing the complexity in software of operating a mm-wavelength interferometer is difficult enough for the developer without the additional burden of having to know in detail the computer science domains of remote access, network protocols, and database technology. The *separation of functional from technical concerns* is a strategy for enabling the application developer to concentrate on the physics, algorithms, and hardware details of aperture synthesis interferometry, while a specialized, system-oriented team provides an easy-to-use *technical* infrastructure of communications, database, and security facilities.

The functional architecture further divides these interferometry-related tasks into subsystems that can be developed in relative independence from each other.

The technical architecture furnishes these subsystem developers with simple and standard ways to 1) access remote resources; 2) store and retrieve data; 3) manage security needs; and 4) communicate asynchronously with other subsystems and components. In the end, separation of concerns is no more than a variant of the divide-and-conquer strategy that Caesar used to conquer Gaul 2000 years ago.

3.2. The difficult issues

The preceding discussion gives an idea of the difficulties that ALMA software development must confront. Tables 1 and 2 summarize these problems and how we are addressing them. The most important of these will be discussed later in this paper.

Table 2. Development-time Issues

Challenge	Response
Evolving requirements	Iterative development
Changing data rates	Modular, flexible design
New observing modes	Scriptable observing procedures
New hardware (ACA)	Generic, parameterized design
IT advances	Isolation of system-dependent S/W
Distributed development	Unified architecture (HLA)
Different s/w cultures	Functional subdivisions aligned to existing project organization
	Common Software (ACS)
	Don't do it twice (but if you really must do the same thing, do it the same way everywhere)
	E-Collaboration tools

3.3. Why dynamic scheduling?

To maximize the scientific return of ALMA, it is essential that the observatory be able to exploit the somewhat rare and unpredictable moments when the water vapor level in the atmosphere is low enough, say between 0.2 and 0.5 mm, to permit observing in the largely unexplored frequency range of 500-1000 GHz. This will be done by operating ALMA in service mode, using a dynamic scheduler (Farris & Roberts 2004) that can react quickly to changing atmospheric parameters, choosing observing programs of the highest scientific priority that will exploit the conditions of the moment.

The principle construct that makes dynamic scheduling possible, is the *Scheduling Block [SB]*, a software object defined to be an indivisible unit of observing activity. Once execution of an SB has started, it can be aborted but not restarted in the middle. An SB will be self-contained to the extent that it will contain all *project-specific* observations necessary to allow calibration of the data that it produces (typically phase and possibly bandpass calibration; more general calibrations, such as baseline determination and determination of the pointing model, are the responsibility of the ALMA Observatory and lie outside the province of a PI's SBs). The dynamic scheduler will be able to query each

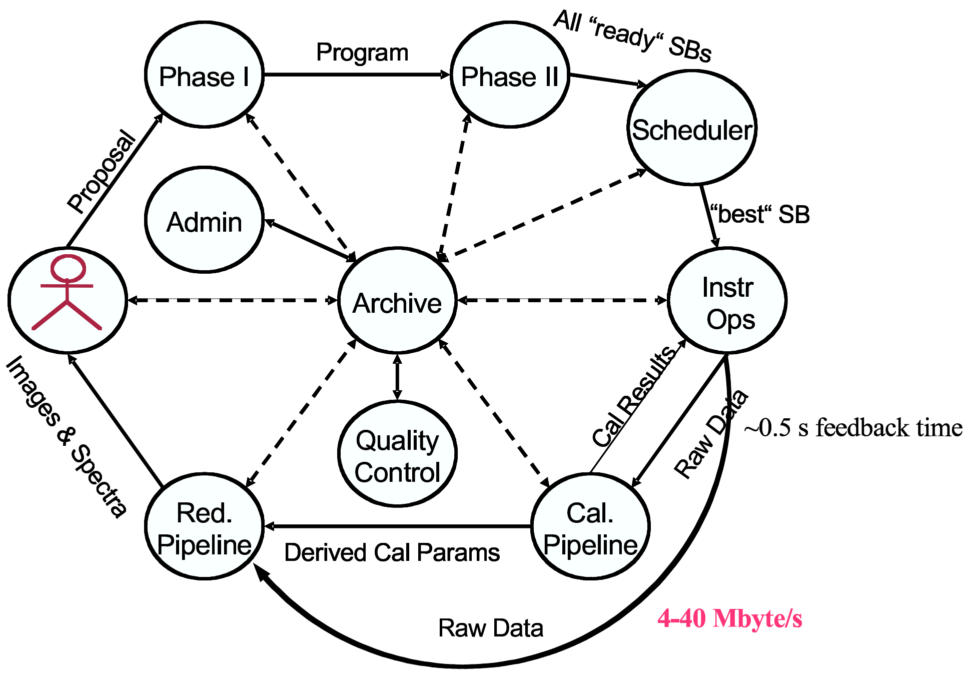

Figure 1. A very schematic view of the ALMA system data flow.

SB to answer questions such as 1) What array hardware (*e.g.,* antenna configuration, receiver bands) do you require? 2) What atmospheric conditions do you need?

The nominal execution time for an SB will be about 30 minutes; PIs can specify longer SBs, but the dynamic scheduler will tend to prefer the shorter ones, all other factors being equal, since these enhance its ability to react to changes in observing conditions. As many, if not most observations of interesting targets will require longer integration times than a single SB allows, multi-SB observations will be the rule, rather than the exception. Quite often, these SBs will not be executed consecutively, but this only presents problems for rapidly varying (pointlike or solar-system sources), for which use of longer SBs will be unavoidable. When an SB has finished execution, either because it has achieved its performance goals or because it has reached its time limit, the scheduler will repeat its selection process.

4. System data flow

Figure 1 shows, in an admittedly oversimplified fashion, the flow of ALMA data from Observing Proposal through Scheduling, Data Acquisition, Calibration, Imaging, Archiving and final delivery to the PI.

4.1. The Archive at the Core

As can be seen from Figure 1, practically all data acquired or generated by the ALMA software system will be saved in its integrated Archive (Wicenec 2004, Meuss 2004). Much more than what we usually think of as a "science archive," the ALMA Archive will hold not only raw and processed scientific data, but observing proposals and programs, a history of site environmental conditions, and hardware characteristics and calibrations. To accommodate the rapidly inflowing raw data, the Archive will be optimized to handle high streaming input/output rates, but will be able to support only significantly lower random access rates. In general, the Archive will deal with three types of data: 1) Bulk data, characterized by high volume and a moderate number of records; this will be stored as binary attachments to VOTable headers; 2) Monitor ("engineering") data, consisting of a moderate volume but large number of records; and 3) "Value objects," low volume, complex searchable structures such as Observing Projects & Scheduling blocks, Configuration information and Meta-data providing link to bulk data (*e.g.*, via VOTables).

A Data Access Layer interface to these types of data has been defined, allowing the underlying database technology to be hidden from subsystem developers. The existence of this interface will give the Archive developers the ability to replace this technology when and if it becomes necessary or desirable to do so.

4.2. Components & Containers

The Container/Component model (Völter 2003) furnishes us with the framework for the separation of functional from technical concerns in the development process. It is similar to frameworks provided commercially by Sun's Enterprise Java Beans, Microsoft's .NET and the OMG's CORBA Component Model. Briefly, a *Component* is a unit of software with a well-defined functional (or "service") interface that is deployable inside a *Container*, upon which it depends for the technical services it needs. This division lets subsystem developers focus on functionality, rather than the details of, for example, remote communication and deployment. The Component also implements a *lifecycle* interface, accessible only by the Container, that allows the Container to manage the Component's initialization, execution and shutdown. Both the functional and lifecycle interfaces are defined in CORBA IDL, which gives developers at least the theoretical option to implement their components in any language which has an IDL mapping. The ALMA architecture in fact provides Container implementations (and therefore allows Component implementation) in C++, Java and Python. A subsystem may contain an arbitrary number of components.

The Container's job is to handle technical concerns centrally and hide them from application developers. It provides convenient access to other components and resources, selected CORBA/ACS Services (Error, Logging, configuration, ...) and enables decisions about deployment (which component(s) should run on which hardware) and start-up order to be deferred until run-time. Should we see a need for additional technical services in the future, these can be integrated into the Container, minimizing any modifications that must be made to the application (Component) software.

Figure 2 illustrates the relationships between Components and their Containers, and between Components and other Components with which they communicate.

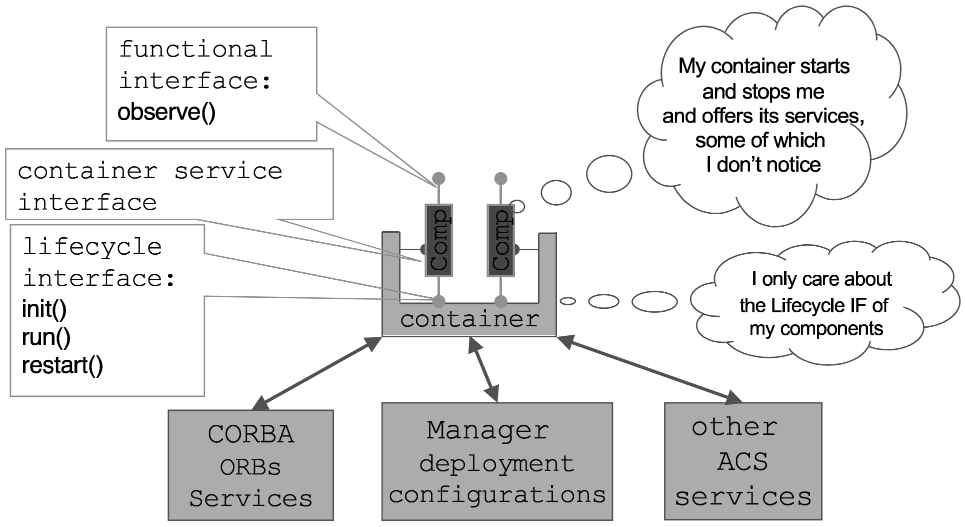
functional interface: observe()
container service interface
lifecycle interface: init() run() restart()
Comp
Comp
container
My container starts and stops me and offers its services, some of which I don't notice
I only care about the Lifecycle IF of my components
CORBA ORBs Services
Manager deployment configurations
other ACS services

Figure 2. The Container/Component environment.

Role(s) of XML An important contribution to the capabilities of the Container/Component framework comes from the multifaceted use of the eXtensible Markup Language or XML (Sommer 2004). We define the structure and content of "Value Objects", complex data objects such as Observing Projects, Scheduling Blocks and VOTable headers via XML schemas. Binding classes that allow type-safe native language access to these data structures and that can validate compliance with these schemas are generated automatically by an open-source framework. The Containers provide transparent serialization of these Value Objects to XML, enabling subsystems to exchange and modify them in a language-independent way. Moreover, the Archive has been designed to accept such schema-conformant XML documents directly.

We are currently investigating the utility of generating the XML schemas themselves from an overall observatory data model expressed in the Unified Modelling Language (UML). This approach holds out the promise of more efficient maintenance of the data model itself, drastically shortening the time needed to incorporate the (inevitable) changes to the model that the evolution of ALMA and of our own understanding will make necessary.

4.3. Data reduction pipelines

We will not attempt to recreate or rewrite the vast body of software that exists to process and analyze the data from radio aperture synthesis telescopes, but will rely on the reuse of such software, in particular, on AIPS++ as the data reduction engine for ALMA (Davis 2004). A subgroup of ALMA's Science Software Requirements Committee has audited AIPS++ for compliance with the requirements of ALMA, and has identified those areas where functionality is lacking or performance is inadequate. A joint group of AIPS++ project members and ALMA developers are addressing these concerns. Another joint effort, in this case by IRAM and AIPS++ staff, has verified the suitability of AIPS++ for the reduction of mm data from the interferometer at Plateau de Bure. Systematic benchmarking of AIPS++ has led to major performance improvements during 2003, so that we are now confident that the package will be able to meet ALMA's goals for rapid processing of the acquired data.

Meanwhile, AIPS++ itself is evolving to use a standard, open-source scripting language (Python) to replace its existing, bespoke scripting language (glish). A proof of concept effort is underway to recast AIPS++ tools as Components (in the sense discussed previously), using the C++ and Python Containers provided by ACS. The first phase of this three-phase project has completed successfully, giving us reason to hope that AIPS++ will integrate seamlessly into the ALMA software environment.

4.4. ALMA Common Software (ACS)

The ALMA Common Software (ACS) implements the separation of functional from technical concerns, especially the Container/Component model. More generally expressed, ACS is a framework for a distributed object architecture that is used from the highest level software all the way down to the device level in the subsystems that control antennas, receivers and correlator (Jeram 2004). Built on CORBA, ACS hides CORBA's considerable complexity, wrapping those CORBA services used by ALMA.

ACS has been carefully designed to be independent of commercial software, for example through the use of high-quality open-source ORBs such as TAO and JacORB. It is constantly evolving to meet developers' needs, having reached, as of this writing, Release 3.0. The most recent developments have included the implementation of the Python Container, needed by the subsystem responsible for pipeline data processing, and an all-Java version (albeit with reduced functionality), which is necessary for the developers of the Observing Tool to provide software that can be installed on almost any astronomer's desk- or laptop computer.

At the start of the ALMA software effort there was considerable resistance by developers to the discipline and standardization that is a consequence of ACS. It is testimony to the soundness of the concept of common software, the quality of its implementation and the dedication of the ACS team that developers are now asking that ACS offer more features and support for their activities.

4.5. Avoiding nasty surprises

The history of software development is littered with the ruins of projects that failed to meet the promises of their managers or the needs of their intended users. It is generally recognized that it is only the rare project that *re-implements* an existing system that can hope to freeze the requirements early in its lifetime. As an experiment in the truest sense of the word, ALMA is certainly not such a project. We have already encountered changes in our requirements and as the characteristics of the hardware become clearer and an operations plan is finally agreed upon, we must expect more. To mitigate the impact of these changes, we follow an iterative development cycle: integration of all subsystem software is performed by a dedicated Integration & Test team every month. Significant additions to the system's functionality come with bi-annual releases, interspersed with annual design reviews (somewhat iconoclastically termed "Critical Design

Review No. 1, 2, ...") that concentrate on the functionality to be developed during the coming year.

To date, the ALMA Software System has completed an Internal Design Review, a Preliminary Design Review with reviewers external to the computing group, the first "Critical" Design Review, and two code releases. The first of these code releases, R0, tested the build procedures and tools and a minimal amount of code, while the second release, R1, successfully implemented a skeletal end-to-end data flow, testing interfaces, communications mechanisms, and developers' understanding of them. The R1.1 release will concentrate on eliminating the problems and mismatches uncovered in R1, while R2 will aim at providing a useable system for the Antenna Test Facility at the site of NRAO's Very Large Array (VLA) in New Mexico. Later releases will build towards support of Early Science Operations in 2007 and, with the benefit of experience in the use of earlier versions of the software, of Full Science Operations in 2012.

It remains to learn from the sad fate of projects whose end product evokes the following from their customers: "It's beautiful software but not what we wanted." We believe that the key to avoiding such a regrettable epitaph is to ensure that user participation doesn't end with the publishing of a requirements document. To this end, a scientist expert in the problem area is assigned to every subsystem development team with the task of 1) providing advice and help to solve problems during development; 2) making sure that requirements are met and that the requirements themselves are up to date; 3) evaluating subsystem progress and redefining requirement priorities when appropriate; 4) ensuring that subsystems interface properly; and 5) helping to develop a test plan and to perform periodic testing and evaluation of the software from a scientific user's perspective.

The user tests represent the culmination of a comprehensive testing strategy. At the lowest, or most detailed level, subsystem developers are responsible for unit tests that verify the correctness of pieces (typically, classes or small groups of collaborating classes) of code. Tools such as JUnit, pyUnit and cppUnit are commonly employed to automate and standardize these tests. A "test first" attitude is encouraged. Subsequent automatic performance tests will be used to ensure that timing constraints are met and that data throughput is adequate. The stand-alone subsystem user tests referred to earlier will be performed before subsystem releases with adequate time to allow subsystem developers to respond. Finally, integrated user tests that exercise the complete system will be executed as soon as possible after integrated subsystem releases.

4.6. High-level Analysis & Design

The authors of this paper form the ALMA Computing IPT's High-level Analysis & Design (HLA) group. HLA develops and maintains the system architecture and fosters its implementation in close cooperation with the ACS team. Overseeing subsystem-subsystem interfaces and guiding the planning for incremental releases constitute another part of HLA's activities. Finally, the HLA group collaborates with the Integration & Test (ITS) and Software Engineering (SE) groups to improve the ALMA software development process.

Under the guidance of a management team anchored on both sides of the Atlantic, these collaborations among HLA, ACS, ITS and SE will play an important part in ensuring that the distributed development of the ALMA software, by bringing the scientific and software expertise of many diverse participants to bear on the challenges facing it, can outweigh the added communications effort needed and ultimately prove a major factor in the project's success.

References

Bridger A. et al. 2004, this volume, 85

Davis, L. et al. 2004, this volume, 89

Farris, A. & Roberts, S. 2004, this volume, **??**

Jeram, B. et al. 2004, this volume, 748

Lucas, R. et al. 2004, this volume, 101

Meuss, H. et al. 2004, this volume, 97

Sommer, H. et al. 2004, this volume, 81

Völter, M. et al. 2003, Server Component Patterns, (New York: Wiley)

Wicenec, A. et al. 2004, this volume, 93