ALMA SOFTWARE DEVELOPMENT APPROACH

G. Raffi , European Southern Observatory (ESO), Munich, Germany
B.E.Glendenning – National Radio Astronomy Observatory (NRAO), Socorro, NM, USA

Abstract

The Atacama Large Millimeter Array (ALMA) is a joint project involving astronomical organizations in Europe and North America. The primary challenge to the development of the software is the fact that its development team is extremely distributed geographically. In addition it has very ambitious goals covering the whole end-to-end software system.

The software development approach, based also on the experience of previous large projects is very pragmatic and contains elements of various methodologies, from the classical top-down model, to early prototyping, incremental development, and even aspects inspired to agile methodologies.

INTRODUCTION

The Atacama Large Millimeter Array (ALMA) is a joint project involving astronomical organizations in Europe and North America. ALMA will consist of at least 64 12-meter antennas operating in the millimeter and sub-millimeter range. It will be located at an altitude of about 5000 m in the Chilean Atacama desert (see Ref. [1]).

Fig.1 shows the ALMA centres located in Chile: the Antenna Operation Site (AOS) where the antennas are located, the Operation Support Facility (OSF) in San Pedro and the Santiago Operation Centre (SOC).

The peak data rate between AOS and OSF is 60 MB/s (distance 35 Km), while the sustained average data rate on the link OSF/SOC will be 4 MB/s.

PROJECT MANAGEMENT APPROACH

The ALMA Computing group is subdivided in teams located at 11 institutes around Europe and North America and consists of almost 50 people, many of them working on ALMA only part time. These have started the development of the various subsystems, namely: Proposal Preparation tools, Instrument operation, On-line calibration and reduction, and Archiving in addition to the Control and Correlator software subsystems, started some time ago. All of them are based on the ALMA Common Software (ACS – separate presentation and demo at this Conference) providing a mandatory and useful framework, which while avoiding duplications is well tested and offers common design patterns important also for later maintenance.

The work is split among 10 subsystems, with a number of common activities on top (most notably architecture, software engineering, and integration & test)

Reasons for Development Framework

The scientific requirements for the whole system were defined in great detail, complemented by Use Cases, and

Figure 1: ALMA Sites in Chile

then analyzed arriving at a proposed architectural design (classical top-down approach). This we considered as a necessary step to get started, but by far not good enough to continue for long time in the future without verification. Therefore the iterative development model was also considered from the beginning.

Additionally development of science software, where feedback from the early operations phase is essential, has been planned to be done in two separate periods, essentially upgrading the whole software over a period of 4 years.

In parallel software was developed to satisfy the needs of the first antenna prototypes to be evaluated (bottom-up work). However even for this early work separate teams provided the framework (ACS) and the control system on top respectively. This approach was hard to get started but is now fully understood and accepted and has become the conceptual basis for the whole ALMA development.

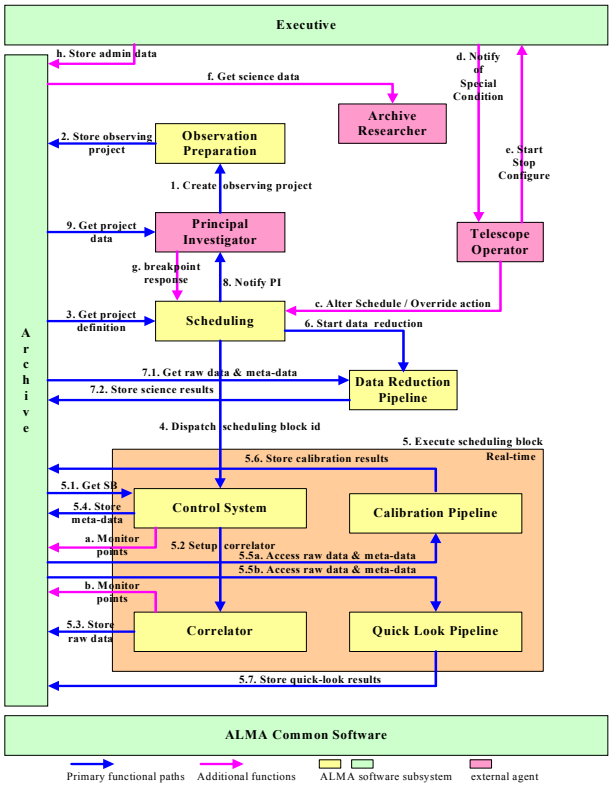

Figure 2: ALMA Software Architecture

The resulting architecture (see Fig.2) was then extended to include Container-Component concepts, resting on a communication layer based on CORBA and making extensive use of standard ORB services. This enforces the concept of separation of concerns between the domain specific features of subsystems and the technical features related to communication and housekeeping, which are delegated to ACS to be solved in a common way for all subsystems (see Ref. [2], [3], [4]).

Formal vs. Informal Design

While teams corresponding to subsystems could in principle work autonomously, we believe that it is essential to achieve a coherent global design and to make sure that this will hold in time. To obtain this we held a few extended face-to-face meetings to discuss the architecture and interfaces with subsystem leaders, cross checking with them the features required from ACS. This avoided too much paperwork in the early discussion phases with people rather carrying away simple notes and ideas. The final result for the preliminary design review was then formalized in Design, Interface Control and Plan documents (one each per subsystem). All this would have not been possible without the preliminary meetings among key designers. The various development teams are now free to use any methodology they find suitable to develop their software or even no methodology, provided they comply with the interfaces, overall architecture, and periodic milestones concerning software releases. At 6 months from PDR and after the first incremental critical design review, we assume to have in place already most of the paper work necessary. Obviously design details will be needed for the reviews and maintenance documentation, but in general paper is clearly not the goal and we will try to limit it to the minimum.

We rather intend to have periodically informal design meetings among key designers.

Iterative Development

The iterative development based on incremental design reviews (one per year) and synchronized releases (every 6 months) will allow monitoring of the development process. We intend to take advantage of this cyclic process to further detail requirements and steer the whole process, by involving users in the development process to avoid requirements developed earlier from quickly becoming obsolete.

In particular a scientist is assigned to every development team to provide advice, make sure requirements are met and possibly redefine their priority. Additionally he has an essential part in the testing process, as explained better below.

Software development sees shorter cycles with builds being produced automatically and daily from the central software repository and monthly tagged all-software integrations for checking interface integrity and reporting anomalies to the different subsystems (much before these become an integration problems at the 6-monthly release).

Milestones

An interesting aspect for control software developers is the fact that software releases are detached from hardware milestones, so that control software does not entirely depend on hardware deliverables and its possible delays (even if obviously final testing and commissioning needs hardware availability). The emphasis is on meeting milestones and developing first the global aspects that are needed for the integration work.

Testing strategy

The goal for testing is to use 20% of the development time for modular and subsystem tests that will be based on commercial and open-source unit test tools. All these tests will then be embedded in an automatic test tool called TAT (that exists already) and delivered to a separate Integration team for automated regression testing (Fig. 3).

Different kind of tests are performed. Unit tests, which verify the functionality of single programming units, are fully automated and performed with the help of JUnit, pyUnit or cppUnit respectively, depending on the development language, with a home made tool (TAT)

being used for the automatic integration tests (see Ref. [4]).

Performance test, stand alone tests and integrated user tests are meant to be added and do exist only very partially at this stage.

However there is a precise effort being made to prepare user end tests, based on user test cases. The goal here is to audit the implementation of requirements and update this after each test and before releases.

CONCLUSION

The ALMA Computing group experience shows so far shows that in spite of theories teaching that software development should be co-located and better developed in small groups, things start to move forward and the release cycle milestones are maintained.

However this is not without a big communication effort and conversely accepting quite some inefficiency.

The well understood need for periodic face to face meetings, which are expensive, is an inevitable tribute to pay to the extreme distribution of the project.

The compensation mechanism for this is that we have a much wider area of expertise and together a much better overview and capability to come up with the right idea and solution. This is what we see every time in face to face meetings. We notice also that while certain areas need a formal approach, like interfaces between software subsystems designed by different teams, a much more agile approach and even different methodologies can be applied in different teams. Our approach here is that as long as release milestones are met and the Integration team can integrate subsystems, we do not wish to prescribe how design and implementation have to be done.

ACKNOWLEDMENTS

The work reported here is the result of the effort of a group of almost 50 people scattered around Institutes located in Europe and North America.

We wish to acknowledge explicitly their enthusiasm and dedication, which in spite of the difficulties connected to their distance is keeping the ALMA software so far on schedule.

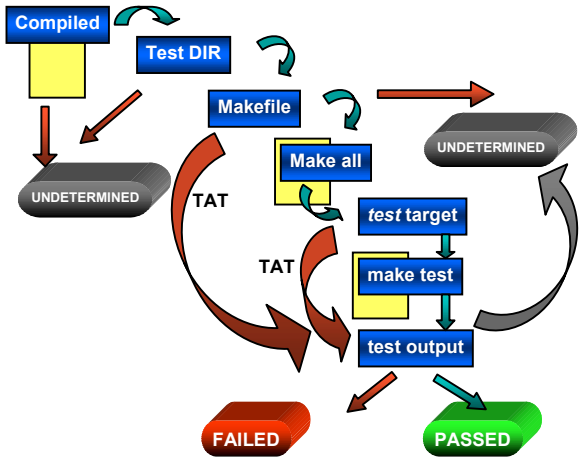

Figure 3: The integration and test cycle

REFERENCES

[1] ALMA Web page under: http://www.alma.nrao.edu/development/computing

[2] ACS Web Page, http://www.eso.org/projects/alma/develop/acs

[3] G. Chiozzi et al., "The ALMA Common Software (ACS):Status and Developments", ICALEPCS 2003, Gyeongju, October 2003

[4] G. Chiozzi et al., "Common Software for the ALMA Project", ICALEPCS 2001, San Jose, 2001

[5] Alma Software Engineering pages: http://www.eso.org/projects/alma/develop/alma-se/reference/IntegrationTools.html