ACS from development to operations

Alessandro Caproni[a], Pau Colomer[b], Bogdan Jeram[a], Heiko Sommer[a], Gianluca Chiozzi[a], Miguel M. Mañas[b]
[a]European Southern Observatory, Karl-Schwarschild Str. 2, D-85748 Garching, Germany
[b]GTD System & Software Engineering, Paseo Garcia Faria 17, E-08005, Spain

ABSTRACT

The ALMA Common Software (ACS), provides the infrastructure of the distributed software system of ALMA and other projects. ACS, built on top of CORBA and Data Distribution Service (DDS) middleware, is based on a Component-Container paradigm and hides the complexity of the middleware allowing the developer to focus on domain specific issues.

The transition of the ALMA observatory from construction to operations brings with it that ACS effort focuses primarily on scalability, stability and robustness rather than on new features. The transition came together with a shorter release cycle and a more extensive testing.

For scalability, the most problematic area has been the CORBA notification service, used to implement the publisher subscriber pattern because of the asynchronous nature of the paradigm: a lot of effort has been spent to improve its stability and recovery from run time errors.

The original bulk data mechanism, implemented using the CORBA Audio/Video Streaming Service, showed its limitations and has been replaced with a more performant and scalable DDS implementation.

Operational needs showed soon the difference between releases cycles for Online software (i.e. used during observations) and Offline software, which requires much more frequent releases.

This paper attempts to describe the impact the transition from construction to operations had on ACS, the solution adopted so far and a look into future evolution.

Keywords: Alma Common Software, CORBA, distributed programming, software infrastructure, framework

1. INTRODUCTION

ALMA, an international partnership of Europe, North America and East Asia in cooperation with the Republic of Chile, is one of the largest astronomical observatories in existence. It operates a single telescope composed of 66 antennas located on the Chajnantor plateau in Chile. All antennas are now at the ALMA site and the observatory is transitioning towards full-scale science operations.

The ALMA Common Software (ACS), developed by the European Southern Observatory (ESO) in collaboration with its partners, provides the infrastructure of the distributed software system of ALMA and other projects, and is available under the open source LGPL license. ACS, built on top of CORBA and DDS middleware, is based on a Component-Container paradigm and provides an API that hides the complexity of the middleware allowing the developer to focus on domain-specific issues. ACS adopts a different interoperable implementation for each of the three supported languages, C++, Java and Python and provides services for distributed programming like transparent remote object invocation and activation, component lifecycle management, publisher/subscriber, distributed error logging, alarm management, configuration database, simulation, deployment, testing and debugging tools.

When the observatory entered in operation, part of the technical time previously assigned to engineering had been assigned to science; moving towards full scale science, technical time will be reduced more and will be eventually allocated 24/7 to science. In that context, the stability of the software system is of primary importance, especially that of the common infrastructure. Being ACS almost complete, we focused primarily on scalability, stability and robustness delaying the delivering of new features, apart those required by operational needs. During this transition phase, an incremental software delivery release schema has been adopted associated to a more extensive testing at the observatory.

The be ready to the delivery of antennas at the observatory and the amount of data they produce, as well as the introduction of subarray, ACS focused on the scalability where the most problematic area has been the CORBA notification service, used to implement the publisher subscriber design pattern. Problems in that area are intrinsically difficult to debug, because of the asynchronous nature of the paradigm and in ALMA, notification channels have been also used for point-to-point communication. Run time failures in the publishers, in the subscribers or a sudden crash of the service were very difficult to recover and subsystems were not able to properly restore their state: the only way to recover from an error was a restart of the entire software system that could require more than 30 minutes. Therefore, to reduce the downtime, a lot of effort has been spent to improve the notification service stability and recovery from run time errors.

An investigation of the replacement of CORBA notification channels with DDS was conducted times ago and looked promising, but further development has been put on hold until the ALMA software will be considered stable enough. A comparison of DDS with a messaging library like ZeroMQ is also foreseen.

As the amount of scientific data and the number of senders and received grew, the ACS Bulk Data (BD) tool that allows to send big amount of scientific data from senders to receivers, initially developed with the CORBA Audio/Video Streaming Service, showed its limitations especially recovering from errors, therefore it has been replaced with a high-performant, scalable DDS implementation.

During operation, we realized that the release cycles of Online software (i.e. used during observations) differs from that of Offline software, which requires much more frequent releases to quickly adapt to the changes in operation. As a consequence, offline subsystems started to reduce their dependency on ACS that could not ship libraries and tools common to all the subsystems without substantially impacting their release cycles. The updating of some of the strategic tools like Java must now be done more frequently, e.g. to promptly respond to security issues of web applications.

2. ACS RELEASES FOR ONLINE AND OFFLINE SYSTEM

The transition of the observatory from development to operation moved the focus to the stability of the software more than on the delivery of new features. Since ACS is the framework for the entire ALMA software, any change in ACS potentially affects all the other subsystems and it is therefore of primary importance to keep a very good configuration control and to avoid any change that could introduce unexpected and unplanned disruption. The need to keep such changes under strict control is particularly important for the external tools, i.e. the tools and libraries (including but not limiting to the CORBA and DDS providers), shipped as part of ACS. In the past, ACS was the central point to collect and distribute libraries shared by different ALMA subsystems, even if they were not directly used by the ACS layer itself. The purpose was to better control what libraries and tools were installed in operation, ensuring consistency of versions across the different subsystems. But the side effect of this approach was that ACS releases were shipping a lot of libraries and on which ACS had no real control making the distribution very heavy; an update of an external tool, used possibly only by one or two subsystems, had to be coordinated with ACS and required a new release. Each change in the external tools had to be discussed and explicitly approved by all the subsystem leaders before being implemented. As consequence any little change in the external tools required a long time to be approved, implemented and finally delivered.

We realized that the development cycles of the software running Online software (i.e the software used during observation), and Offline were very different. The former must be kept very stable in order not to disrupt operation; the latter requires more frequent releases to quickly respond to changes in observational requirements or to install security patches for web applications. Online releases are needed less frequently because the new functionalities to be delivered in each observation cycle have to be ready once per year, before the new cycle starts. To overcome the mismatch of release cycles, Offline subsystems progressively reduced their dependency on ACS.

To reduce the impact of changes in the external tools and the dependencies among subsystems, the ACS distribution has been made lighter by taking out all the tools and libraries that for many years ACS collected and distributed on behalf of other subsystems. In this way each subsystem is able to upgrade such tools without following the release cycle of ACS. Shared libraries and tools have been moved in a centralized dedicated area, built and installed immediately after ACS; responsibility of upgrades and consistency had been given to the Software Engineering group.

An incremental software release cycle has been adopted[7]. For the Online software, the features of the observation cycle are not delivered all at once before the cycle starts, but in several, about 4, releases during the year. Each incremental release is installed and deeply tested at the observatory with the real hardware, during testing time, but only the last one is validated at the telescope to be used for the next observation cycle. From a practical point of view, it means that the Online software running at the telescope has been developed about one year before being used.

The different life cycles of Online and Offline releases becomes more explicit looking at the number of releases as shown in Table 1: the number of yearly Offline releases is the double than that of the Online releases. One ACS release is delivered for Online plus Offline, and one for Offline use only. Generally, an ACS release for Offline does not provide any new functionality with respect to the previous Online release, being simply a placeholder for the building system: apart of renaming, the same version of ACS is used by 2 Offline releases.

Table 1: 2015 ACS releases

Release name	**Delivery date**	**Offline/Online**
ACS-2015.1	26/01/2015	Offline
ACS-2015.2	09/02/2015	Online+Offline
ACS-2015.3	30/03/2015	Offline
ACS-2015.4	24/04/2015	Online+Offline
ACS-2015.5	29/06/2015	Offline
ACS-2015.6	31/07/2015	Online+Offline
ACS-2015.7	12/01/2015	Offline
ACS-2015.8	23/10/2015	Online+Offline

In the revision control system, a branch-release schema had been adopted to follow the incremental release schema: development is done in the trunk and a new branch is created for each new release. ACS release names have been changed from a progressive number to a naming schema that better reflects the observation cycle: ACS-YYYY.N, where YYYY is the year the release has been delivered and N is the progressive number of the release during the year.

To better keep under control the changes in each release of ACS, one ticket is created for each task with a defined testing and acceptance procedure to be performed at the telescope to accept the fix. New features and improvement are delivered as part of a release, bug fixes are provided as patches of existing releases. Each patch must also be tracked by a ticket and is associated with a definite patch level available in the ACS root folder and fully documented in the release notes. In this way it is possible to know what ACS version is in use at any time at the telescope and to know exactly what features and fixes it contains. Installation of patches at the observatory is discussed in the project tracking software including possible side effects, the improvements it provides and fully tested at the observatory before being approved.

The advantage of the incremental software delivery consists in dividing each feature in a set of progressive steps each with a testing procedure to be performed in the real system that ensures that the fix is working as expected. As described, this schema requires now more extensive testing at the telescope since each step is deeply tested instead of waiting for the entire feature to be ready but this is in conflict with the fact that the transition of the telescope to full science will substantially reduce the available testing time in favor of science. Telescope time, to be agreed between science, computing and engineering, will become a shared scarce critical resource: having a simulated environment for testing will be strategic in a near future[8].

This release schema is still not perfect as it does not cope very well with the case of a feature that does not work as expected. In this case the responsible subsystem has to provide the fix but in the meantime is delaying the subsystems that depends on it. A new release schema will be introduced soon to overcome this problem by accepting, as part of the released software, only the features that work well. The focus will be moved from the software subsystems to the single features it provides.

3. PUBLISHER SUBSCRIBER WITH CORBA

ACS provides the publisher/subscriber design pattern with CORBA notification channels. The ACE/TAO implementation was chosen because it proved more stable and performant than other implementations and provides useful extensions.

Publisher/subscriber pattern is widely used in ALMA: in operations, ACS starts 16 notify service instances, each of which has an allocated number of notification channels, between 1 to 20, but there can be more as there is no limitation on the number of notification channels that can be dynamically activated in a notify service. The number of subscribers and publishers per notification channel is also very variable. There are channels with 1 consumer and more than 200 suppliers, others with about 130 consumers and 55 suppliers and others with an even number of subscribers and publishers.

The performance of the current implementation is good enough but notification channel problems (due to the service implementation itself or misbehaving applications) are often hard to identify and debug because they are intrinsically asynchronous and involve several independent partners (senders, receivers, ACS API and the CORBA notify service). The problems reported in operation range from missing events, to slowness in delivering up to the crash of the notify service itself; the number of problem reports increased with the number of antennas in operation showing that notify services do not scale well.

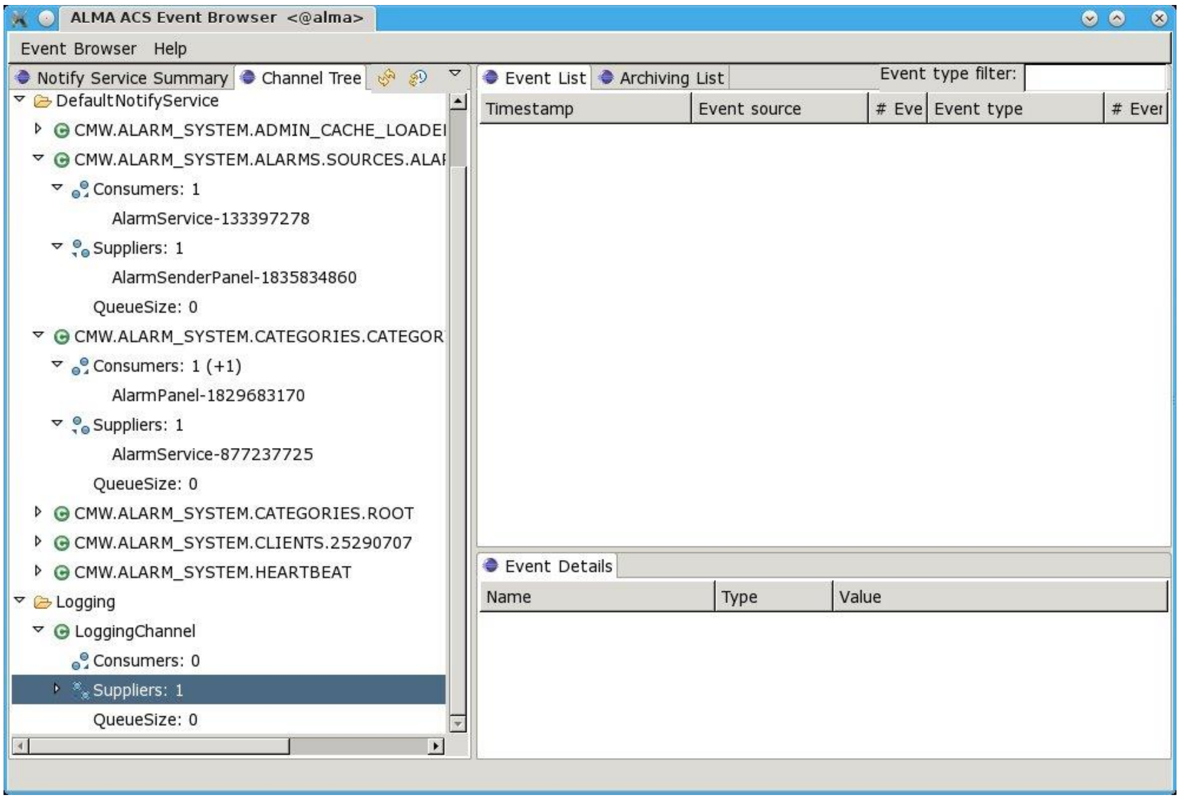
A subscriber that disappears without properly releasing the resources, or a great number of subscribers that cause the allocation of too many threads, or subscribers that are too slow in handling the received events, are example situations that can trigger notification channel problems that are hard to isolate and that could bring, in the worst case, the crash the notify service and the unavailability of all the channels running on it. Even if ACS daemons monitor and restart a misbehaving notify service, such a crash of a notify service was a dramatic event in operation since subscribers were not able to detect the failure and to re-establish the connection with the newly born notification channels: this could lead to inconsistencies requiring a full system restart. Enabling the topology persistence features provided by the ACE/TAO implementation did not help, because it triggered crashes on the subscriber side or the hanging of the service while trying to reconnect to non-existing subscribers.

ACS has been improved by allowing load balancing using multiple notify services and providing support in case of restarting of a notify service. To reduce the load on notify services, the ACS deployment configuration allows to run as many notify services as needed and to instantiate each notification channel in a specific notify service, by providing a channel mapping in the configuration database; the ACS API hides deployment details to subscribers and publishers, so that adding notify services or moving one channel from one service to another is as simple as changing the setup in the configuration database.

ACS daemons are in charge of the startup of ACS services and containers and, in addition, they monitor the status of the services. ACS daemons monitor notification services at run time to be able to identify and restart a misbehaving service. The API of publishers and subscribers has been recently extended to recognize when the daemons restarted a notify service: the publishers cache the events in a local queue that is flushed when the notify service becomes available again. In a similar way, a dedicated thread on the subscriber can now check if the notify service has been restarted and reconnects to the newly born service. The auto-reconnection of publishers and subscribers reduces significantly the downtime due to notification channel problems and the need for a full system restart.

The auto-reconnection has also been implemented in the logging notification channel, the channel where all the logs produced by the online software are centrally published by the loggers. As described for the other notify services, if the logging notify service disappears the logs are temporary cached by the loggers and will be flushed as soon as the notification channel will be restored. Clients like the loggingClient automatically reconnects without any intervention from the operator.

The ACS eventGUI is a powerful eclipse Rich Client Platform (RCP) application that allows debugging notification channel problems at run-time allowing to browse the available notification channels at a given point in time and, after subscribing to the channels, to observe in real time the number of connected subscribers and publishers and to receive the events published in the notification channels. The eventGUI is very useful for debugging, but unfortunately it cannot be practically used in operation because it requires a running connection to the notification channels and a software engineer looking at the events in real-time. To get information of the running system, ACS provides statistics on the usage of notification channels at run time as we will see in 5.1.



```
DefaultNotifyService
  CMW.ALARM_SYSTEM.ADMIN_CACHE_LOADEI
  CMW.ALARM_SYSTEM.ALARMS.SOURCES.ALAI
    Consumers: 1
      AlarmService-133397278
    Suppliers: 1
      AlarmSenderPanel-1835834860
    QueueSize: 0
  CMW.ALARM_SYSTEM.CATEGORIES.CATEGOR
    Consumers: 1 (+1)
      AlarmPanel-1829683170
    Suppliers: 1
      AlarmService-877237725
    QueueSize: 0
  CMW.ALARM_SYSTEM.CATEGORIES.ROOT
  CMW.ALARM_SYSTEM.CLIENTS.25290707
  CMW.ALARM_SYSTEM.HEARTBEAT
Logging
  LoggingChannel
    Consumers: 0
    Suppliers: 1
    QueueSize: 0
```

Figure 1: A snapshot of the eventGUI

A prototype implementation of the notification channels over DDS was set up time ago[2] because DDS offers many advantages with respect to the CORBA notification channels especially in terms of performance and scalability. DDS does not require a dedicated server and is largely configurable with QoS (Quality of Service) policies. An implementation of notification channel over a messaging system like ZeroMQ will also be evaluated. There has been no further development in this area to give priority to stability and robustness of the current system while transitioning to full operation with all the antennas: the number of problem reports on publishers and subscribers has greatly decreased in the past and we believe that the risk of instability introducing a new technology plus the effort required for the porting the existing code is not worth the case. Still we would like to replace the CORBA notification service in the long term.

4. BULK DATA SYSTEM

The ACS Bulk Data (BD) service allows to transfer big amount of astronomical data at the highest possible data rate from many-to-one and one-to-many clients. The first version of the bulk data system was developed on top of the CORBA Audio/Video Streaming service using the TAO implementation[3]. This implementation was not regularly maintained and we saw it was inadequate for our purposes especially for the lack of proper error handling. Therefore we decided to replace it with a new when entering in early science operation. This new version of the bulk data (BDNT), currently in operation, has been implemented on top of DDS[5]. A prototype had been built to evaluate the performances and to select the vendor. At the end of the process, RTI had been selected. The ACS API wraps RTI/DDS to hide to the user low level DDS details and the interfaces has been kept very similar to those of CORBA A/V, to minimize the porting effort on the side of the clients.

Apart from the improved error handling, the DDS implementation of the BDNT offers many advantages over the CORBA one: there is no central BD server, the usage of multicast simplifies the deployment, it is highly configurable with a lot of QoS settings, and connection order is not relevant. To get the highest possible transmission data rate BDNT uses the UDP multicast protocol.

The initially requested data rate for the tool was about 65MBytes/s, being the baseline correlator the most demanding bulk data sender. BDNT has been delivered and tested with only 44 available antennas running in a single array, it is now used with 66 and up to 6 subarrays. The amount of data transmitted nowadays is much different from the initial requirements and a tuning of the BDNT was required. BDNT senders fire UDP packets at the highest possible rate saturating the network; in this situation datagrams are lost and the multicast protocol requires the resending of the lost packets generating even more traffic. We have measured that our network is very stable up to about 95MBytes/s; there is a 35% of loss packets when BDNT senders fire UDP packets at a higher rate. To overcome this problem ACS implemented throttling in the senders: each sender is assigned a maximum data rate for sending data through BDNT, statically configurable in configuration database or dynamically through the API. A proper configuration of the throttling of all the BDNT senders ensures to send datagrams at the highest possible data rate without flooding the network.

After completing the development phase, we moved to the free-of-charge RTI Open Community Source license version of RTI/DDS, made recently available by the vendor. This version does not include support and provides only a limited number of tools to investigate failures and asses performances, making the investigation of DDS asynchronous problems very complicate, but we considered it sufficient for our purpose. As a lesson learned, we now think that ALMA should have switched to a free version of the tool only after having all the antennas and subarray in operation so that debugging and fixing would have been much faster. As a partial justification, the success of the initial version and the known scalability of DDS protocol made us confident enough to get rid of the licensed version of the tool.

5. STATISTICS

Logs, alarms and events are all propagated by means of notification channels instantiated in the notification services. The number of notification services, their deployment as well as the association of channels to services is part of the deployment and the configuration database.

Even if the daemons are able to identify and restart a misbehaving notification service, it is important to identify and fix the cause of the failure. At run time a great number of notification channels is created and the number of subscribers and publishers changes a lot during the different phase of the observations, the number of arrays and available antennas. Debugging problems in such circumstances is very complicate and time consuming therefore we had to introduce several tools to help us troubleshooting problems, one among them is to provide statistics of different, critical parts of the ACS like events, logging and alarms.

5.1 Event channel statistics

ACE/TAO implementation of notification channels provides extensions that can be used to get information on the usage of notification channels at run time. ACS provides tools that use such extensions to measure statistics on the usage of notification channels at run time and that could help identifying anomalies. Statistics are taken at customizable frequency and published at given time intervals and provide, among others, the number of notification channels instantiated in a notify service; for each event channel, the number and names of subscribers and publishers, the size of the queue the names of the consumers that times out, the timestamp of the oldest event in the queue.

We have noticed that if a consumer is slow getting the events, the size of the queue of the notification channel increases and could lead to an out of memory of the service. The statistics help finding which was the problematic consumer and which channel, of the many instantiated in the service, was the guilty one.

In the present implementation, statistics are published in form of logs published at the user defined time interval, in future we plan to save statistics on text files to reduce the load on the logging and increase readability. As future extension, the tool, actually started by the operator, will be part of the deployment.

5.2 Logging statistics

Logs are both saved locally and centrally propagated by means of notification channel in XML format. Tools like the loggingClient get the logs from the logging channel and present them to operators and engineers by means of a java swing table. The loggingClient is used mostly for online debugging and after-the facts investigation.

In operation, the average number of logs published in the logging notification channel is about 4000/second (~8Mbytes/minute) with a peak of about 20000/second (~40Mbytes/minute). The high amount of logs can easily flood the logging service and the network. ACS allows to configure the number of local and centralized logs produced by each logger by setting the log level: logs having a lower level will be discarded. The log level of a logger can be statically configured in the configuration database and changed at run time. Ideally the level should be set high enough to get high level information and lowered only to get more logs to investigate malfunctions.

In some cases the amount of logs produced during observations has been so high to flood the logging service and the network. To investigate such cases ACS loggers implementation provide statistics on their usage by counting, among others, the number of logs published in the user defined time interval, the trend, the number of errors.

Loggers publish statistics issuing an INFO log every user defined time interval. The feature is disabled by default and an IDL method of the logger allows to enable. Enabling and disabling the generation of statistics is done with a command line tool, in future it will be possible through the logLevelPanel, the same GUI that allows to dynamically change the central and local log levels of the loggers.

5.3 Alarm statistics

During the startup of the services, ACS runs one of the 2 available implementations of the alarm system: the ACS implementation that produces a log for each alarm submitted by the sources or the CERN implementation[1]. In the latter, all the alarms generated by the sources are received by the alarm server that, using its knowledge base, is able to identify from all the active alarms the root cause of a problem to be presented to the operators. The binding to the running type of alarm system is done at run time and the APIs of the 2 implementations are the same therefore it is possible to switch from one alarm system implementation to another by tuning a field of the configuration database.

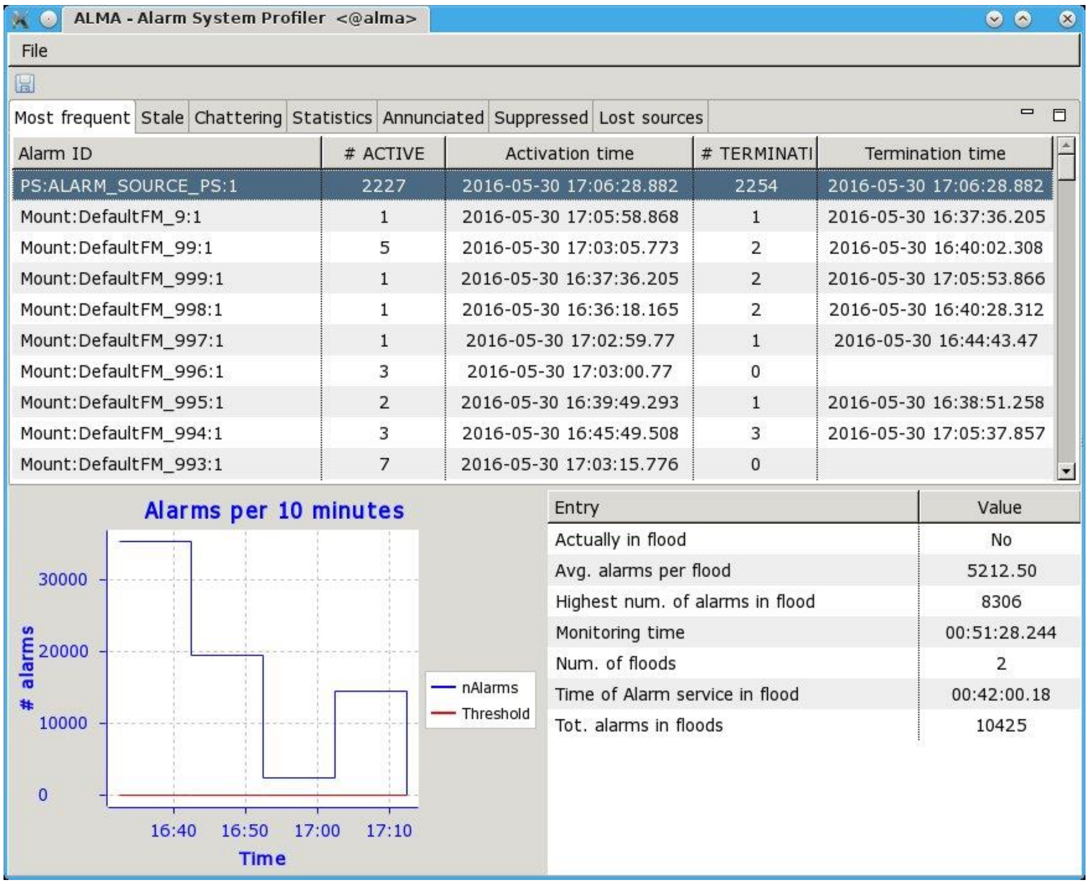
In the CERN implementation, the alarm sources publish the alarms in a notification channel being the alarm server one of the consumers. The alarm server elaborates the alarms from the sources and sends to the alarm clients (for example the alarm panels) an enriched version of the alarms by means of a set of notification channels.

In operation, in average the sources publish 50 to 70 alarms per second but it happened that misbehaving alarm sources were rapidly activating and deactivating alarms, flooding the alarm system notification channels and increasing in an uncontrolled way the workload on the alarm server: in such cases we have measured data rates of more the 2000 alarms per second. The ACS API of the sources has been improved by substantially reducing the actual submission of alarms with internal checks, for example by avoiding to publish an alarm if its actual state had not changes changed and limiting the notification frequency. The alarm server has also been made more robust by moving the alarms out of the notification channel into an internal queue from an independent thread, decoupling the processing from the collection and therefore preventing problems at the level of the notification channel.

Normally the number of alarms produced by the sources should be very low, but in case of a malfunction a burst of alarms can be generated. The alarm server provides statistics, by means of a log message published at user defined time intervals, with the number of alarm activations and deactivations produced by the sources allowing to identify critical execution patterns. The server writes on files more detailed statistics like for example the sources with the greatest number of activations and deactivations in a given time interval.

Usually, if an alarm source is changing its state very frequently, this it is due to a programing or configuration error. Monitoring the system during operation, we have seen that typically few sources are those who generate most of the alarm system traffic, flooding the system. The statistics provided by the alarm server allow to easily identify and fix the most problematic alarm sources.

The alarmSystemProfiler eclipse RCP, allows to investigate the health of the alarm system. It connects to the all the alarms notification channels to get the alarms produced by the sources plus the enriched alarms forwarded by the alarm server to the clients and produces detailed statistics and graphs. At the moment, the alarmSystemProfiler only connects to the running system and therefore is not very practical to monitor the system for a long time span. It will be extended in future to analyze the alarms from files and to produce detailed statistics offline.



Alarm ID	# ACTIVE	Activation time	# TERMINATI	Termination time
PS:ALARM_SOURCE_PS:1	2227	2016-05-30 17:06:28.882	2254	2016-05-30 17:06:28.882
Mount:DefaultFM_9:1	1	2016-05-30 17:05:58.868	1	2016-05-30 16:37:36.205
Mount:DefaultFM_99:1	5	2016-05-30 17:03:05.773	2	2016-05-30 16:40:02.308
Mount:DefaultFM_999:1	1	2016-05-30 16:37:36.205	2	2016-05-30 17:05:53.866
Mount:DefaultFM_998:1	1	2016-05-30 16:36:18.165	2	2016-05-30 16:40:28.312
Mount:DefaultFM_997:1	1	2016-05-30 17:02:59.77	1	2016-05-30 16:44:43.47
Mount:DefaultFM_996:1	3	2016-05-30 17:03:00.77	0	
Mount:DefaultFM_995:1	2	2016-05-30 16:39:49.293	1	2016-05-30 16:38:51.258
Mount:DefaultFM_994:1	3	2016-05-30 16:45:49.508	3	2016-05-30 17:05:37.857
Mount:DefaultFM_993:1	7	2016-05-30 17:03:15.776	0	

Entry	Value
Actually in flood	No
Avg. alarms per flood	5212.50
Highest num. of alarms in flood	8306
Monitoring time	00:51:28.244
Num. of floods	2
Time of Alarm service in flood	00:42:00.18
Tot. alarms in floods	10425

Figure 2: The alarm system profiler

6. CONCLUSIONS

Transition of ALMA telescope from development to operations marked an important change in the life cycle of the software, development focusing mostly on scalability, reliability and robustness of the existing software, rather then in delivering new features. Investigation and fixing of the failures got a great momentum and tools to investigate the health of ACS services and to recover from malfunctioning without having to restart the entire software suite were introduced.

The different life cycle of the Online and Offline software was emphasized to quickly reply to operational needs. ACS philosophy to be a centralized distributor of tools and libraries on behalf of all the ALMA subsystem has been relaxed and offline tools have been made independent of ACS whenever possible. The release cycle was also changed in the same perspective: the number of Offline releases is higher than that of Online and we are still discussing the optimal balance in the tradeoff between frequent updates/fixes and a stable system. Testing at the site has been increased to isolate and fix problems during validation and acceptance of the software. In a short future the telescope will be mostly dedicated to science, reducing significantly the amount of time available to testing and engineering activities. A major effort will be therefore put in providing a simulated platform for testing.

Changes in the number of antennas and subarrays going through the different observation cycles introduced scalability challenges, especially in the notification channels but also in the bulk data transfer, the logging and the alarm systems. To increase the robustness, ACS provides new tools to quickly identify problems and to restart the services minimizing the impact during observation and with almost no impact in the user API.

In these transition years, ACS has been stable enough and resources have been mainly spent investigating and fixing problems reported in operation rather than implementing new features that could have introduced instabilities.

At this point in the life of the project, with transition to operation almost complete and with most of the usage of ACS taking place at the observatory during operation activities, ACS enters a pure maintenance phase and it is clear that problems appear and can be investigated much more efficiently at the observatory rather than at the partner's development sites. Therefore we have now decided, in agreement, with the Joint ALMA Office (JAO), to move ACS maintenance from the ESO headquarters in Garching to Chile.

REFERENCES

1. A.Caproni, K.Sigerud, K.Zagar, "Integrating the CERN Alarm System with the Alma Common Software", Proc. SPIE Vol.6274, May 24, 2006.
2. J.Avarias, H.Sommer, G.Chiozzi, "Data Distribution Service as an alternative to CORBA Notify Service for the ALMA Common Software", ICALEPCS, Kobe, Japan, 2009.
3. R.Cirami, P.Di Marcantonio, G.Chiozzi, B.Jeram, "Bulk data transfer distributer: a high performance multicast model in ALMA ACS", SPIE Vol.6274, May 2006.
4. P.Di Marcantonio, R.Cirami, B.Jeram, G.Chiozzi, "Transmitting huge amounts of data: design, implementation and performanceof the bulk data transfer mechanism in ALMA ACS", ICALEPCS, Geneva, Switzerland, 2005.
5. B.Jeram, G.Chiozzi, R.J.Tobar, M.Watanabe, R.Amestica, "Reimplementing the Bulk Data System with DDS in ALMA ACS", ICALEPCS, TUCOCB08, San Francisco, USA, 2013.
6. A.Caproni, K.Sigerud, K.Zagar, "Integrating the CERN LASER Alarm System with the ALMA Common Software", SPIE Vol.6274, May 2006.
7. R.Soto, T.Shen, N.Saez, J.Ibsen, "ALMA release management: a practical approach", ICALEPCS, Melbourne, Australia, 2015.
8. T.Shen, R.Soto, N.Saez, G.Velez, S.Fuica, T.Staig, N.Ovando, J.Ibsen, "The eveolution of the simulation environment in ALMA", ICALEPCS, Melbourne, Australia, 2015.
9. D.Fugate, "A CORBA event system for the ALMA Common Software", SPIE, 5496-68, Glasgow, Scotland, UK, 2004.