ALMA COMMON SOFTWARE (ACS) STATUS AND DEVELOPMENT

G. Chiozzi*, B. Jeram, A. Caproni, H. Sommer, J. Schwarz, ESO, Garching, Germany
M. Sekoranja, Cosylab, Ljubljana, Slovenia
R. Cirami, INAF-OAT, Trieste, Italy
H. Yatagai, NAOJ, Tokyo, Japan
J. A. Avarias, NRAO, Socorro, New Mexico, U.S.A.
A. Hoffstadt, J. Lopez, UTFSM, Valparaíso, Chile
N. Troncoso, ALMA, Santiago, Chile
A. Grimstrup, University of Calgary, Calgary, Alberta, Canada

Abstract

ACS provides the infrastructure for the software of the Atacama Large Millimeter Array and other projects[1][2]. Using CORBA middleware, ACS supports the development of component-based software, from high-level user interfaces down to the hardware device level. It hides the complexity of CORBA beneath an API that allows the application developer to focus on domain-specific programming. Although ACS, now at release 8, has been used operationally by the APEX radio telescope and at the ALMA Test Facility, the commissioning of ALMA in Chile brings major challenges: new hardware, remote operation and, most important, up-scaling from 2 to 60+ antennas[3][6]. Work now turns to scalability and improving the tools to simplify remote debugging. To further identify potential problems, the University of Eindhoven is formally analysing ACS. Meanwhile, new developments are under way, both to respond to newly identified needs of ALMA, and those of other projects planning to use ACS. Examples include the refactoring of the interface to the CORBA Notify Service, integration with the Data Distribution Service, generation of state machine code from abstract models and of Python binding classes from XML schema.

ACS... IN BRIEF

ACS is a software infrastructure and framework for the development of distributed systems based on the Component/Container paradigm[4], built on top of free CORBA implementations. ACS partially wraps CORBA to hide its complexity and to make it easy to implement applications following standardized architecture and design patterns.

Free software is extensively used, to avoid "re-inventing the wheel". Commonly used libraries and tools are integrated in ACS to ensure that a coherent and homogeneous package is available to all developers

ACS provides the basic services needed for object oriented distributed computing. Among these are:

- Transparent remote object invocation,
- Publisher/subscriber paradigm,
- System deployment/administration and object location based on a container/component model,
- Distributed error and alarm handling,
- Distributed logging,
- Configuration database,
- Thread management,
- XML binding classes and transparent serialization,
- Simulation facilities,
- Standardized testing infrastructure

ACS's primary platforms are Red-Hat Enterprise and Scientific Linux, but it is used also on other Linux variants. A partial implementation on Windows is in use by other projects and work is underway to provide complete support. Real time development is supported on Real Time Linux (for ALMA) and VxWorks (for the APEX project).

Development APIs are available in C++, Java and Python. Any other language with a CORBA mapping can be used, if needed. Coherent support of multiple programming languages is one of the key motivations for the implementation of ACS.

PUBLISHER/SUBSCRIBER WITH CORBA

ACS provides support for the publisher/subscriber paradigm through the ACS Notification Channel[7].

The current implementation is based on the CORBA Notify Service and uses the TAO Notify Service implementation, hiding as much as possible of the CORBA complexity from the developers. The APIs are available for all programming languages supported by ACS: C++, Java and Python.

Although the present system fully satisfies all requirements of the ALMA project, the Notify Service has some limitations, being resource intensive and not scaling well with the number of subscribers.

Investigating problems with the CORBA Notify Service is often very hard and requires a lot of time. Currently ACS is able to restart the Notify Service processes, but this is unable to recreate the connection topology between publishers and subscribers after the service is restarted.

To solve this last issue, ACS has recently implemented an extension of TAO allowing the Notify Service to store its state persistently each time that a change in the

* gchiozzi@eso.org

topology occurs. This allows restoring its last state (objects and connections) upon restart. We have also implemented a circular queue to store temporally the events that are not sent through the Notification Channel and a Handler Callback to let a user of the NC deciding what to do with events not correctly delivered to the subscribers.

All these new features make a more reliable Notification Channel, decreasing the downtime and eliminating the need to restart all the others subsystems after a potential NC crash.

PUBLISHER/SUBSCRIBER WITH DDS

Since the CORBA Notify is not an ideal publisher/subscriber implementation, we have also decided to investigate new technologies able to provide a better solution.

The Data Distribution Service (DDS, http://www.omgwiki.org/dds) is an OMG open international middleware standard (as is CORBA) directly addressing publish-subscribe communications for real-time and embedded systems. With respect to CORBA Notify, it offers better performance and features decentralized message processing, scalable peer-to-peer communication, and a wide set of QoS policies.

For ACS we have implemented a prototype of the same ACS Notification Channel APIs with OpenDDS and used it for evaluation in the E-ELT Demonstrator . We have also been evaluating RTI DDS. It is possible to switch transparently between the two implementations, so that a comparison of performance and reliability is possible.

This implementation seems very well suited for intensive data-centric applications where the publisher/subscriber model allows keeping much better de-coupling between the various components of an application with respect to a more traditional client/server architecture.

More details on DDS for ACS are presented in another paper in this conference[5].

MODEL DRIVEN DEVELOPMENT

Modern control systems involve multiple subsystems and devices. During the lifetime of the project, the design changes and keeping models and implementation aligned is time consuming and often neglected.

Moreover, many of the steps needed to move from design to implementation involve a lot of repetitive code copying activities, in particular when using a comprehensive application framework like ACS.

A better approach is to use model-driven development techniques, where the code is generated automatically by a generator, based on a model. This directly reduces the number of errors per LOC and enforces a single coding style for a large portion of the code base.

Such an approach allows accommodating model changes more easily and it improves productivity, since developers can focus on business logic instead of on implementation details. ACS is very well suited for this approach.

The MDD system currently under development allows:

1) Code generation starting from a UML model:
   a) Generate the IDL (Interface Definition) files. This implies creating a full implementation of the UML model as an IDL file so that it may be compiled by IDL compilers.
   b) Generate (now only in Java) fully functional base class implementation of the applications' components starting from a class diagram.
   c) Generate the corresponding Configuration Database and deployment information
   d) Generate a basic test suite
2) Integrate design patterns into the code during the generation process.

An advanced prototype is available and has been used to generate simple systems.

FINITE STATE MACHINES

Control applications are in principle very naturally mapped into state machines.

Clearly the direct control of physical devices needs to be modeled using finite states machines, but also the high level coordination between the subsystems of a telescope or the sequencing of observations would be very conveniently described using state machines.

Finally, in the last few years some generic ways of specifying and implementing finite state machines have become available.

In recent ACS developments we have adopted the approach of modeling state machines with a UML tool and generating from that the skeleton of a complete application where only the specific code for actions and transitions need to be implemented.

The actual state machine is generated from UML into a general purpose state machine engine, using the openArchitectureWare(http://www.openarchitectureware.org/) Framework, so that this stage of generation can be used in different application frameworks (like ACS and VLT CCS).

An additional generation step produces all application code needed for the specific application framework adopted by the system.

This strategy allows generating the application for different frameworks from the same model, allowing better reuse of the model.

ACS DAEMONS

Until ACS 7.0, all ACS services and Containers running on the different nodes of a distributed control system were started by the main startup node by means of remote secure shell sessions. The intelligence on the deployment of the system was therefore centralized on this main node.

As of ACS 7.0 we introduced a new strategy for startup, deployment and system management based on daemons deployed on the distributed nodes with the

responsibility for monitoring and deploying services and containers.

The central ACS manager delegates to the daemons the task of starting and stopping containers.

The daemons can monitor resources on the nodes, check the life status of all entities under their control, collect statistics on CPU, disk, memory and other resources much better than what the manager and the other ACS administration tool were previously able to do using plain SSH sessions.

This approach has significantly improved the reliability of the system, its resilience to problems and the possibility of monitoring status and diagnosing problems.

New features are being added to the daemons in the upcoming releases, based on the feedback from the commissioning team at the ALMA operation site.

PYTHON XML BINDING

The ALMA Science Data Model (ASDM) is a collection of XML schema that describe format and characteristics of the data used to define an observation. The ALMA Observing Tool generates a set of XML files that contain the necessary information to manage the operations of the array and achieve the observer's science goals.

The transformation of the information from Java's internal representation to text-base XML is performed by a set of bindings generated using the Castor libraries. Using generated bindings can improves the accuracy of the resulting XML document while reducing the programming effort required in its generation. The generated code will always be consistent with the schema and it is easier to migrate old observation models to newer versions of the schema.

While the observers benefit greatly from such standardization, those working on telescope acceptance and verification are not so fortunate. A large portion of that effort, both interactive and automated, is performed using Python scripts. Python excels as a language for test automation, but the tools available in ACS to manipulate ASDM files are complex to use and fail to capitalize on the main strength of XML schema: data validation. Having an equivalent set of XML bindings for Python would greatly improve the test code and the efficiency of the verification process.

The Python language has many XML parsing libraries available, but very few binding generators. We have evaluated two packages for inclusion into the ALMA Common Software suite: generateDS and PyXB. generateDS creates a SAX-based binding for a schema while PyXB is more DOM-based. Both packages are under active development, but currently PyXB is better able to handle highly interrelated data models like the ASDM. PyXB-generated bindings also have stronger constraint-checking. Both packages are under active development.

At present, PyXB better fits the needs of ALMA. We are working closely with the developer to identify and address problem areas and hope to incorporate it into the ACS tool suite soon.

CONCLUSION

ACS development is very active at this time.

On the one hand, the core ALMA/ACS team is working to satisfy the requests coming from the integration team at the ALMA Operations site. This involves mainly optimization, bug fixing, improvements in performance and reliability. ALMA does not need “innovation” now, but to bring the system reliably from 2 to 60+ antennas.

On the other hand, new projects starting using or evaluating ACS request support for new technologies and want to simplify the lifecycle of application development, trying, for example, to obtain higher productivity through Model Driven Development. As in any good open source project, the community outside the core development team takes a very active role in doing the investigations and developing prototypes or even implementing production quality solutions in these areas. A major role is being played at this time by the collaborations with UTFSM in Valparaiso and other universities.

Nevertheless, this external work is coordinated with the core team, ensuring coherent development and avoiding dead-ends. From past experience, we can already say that several of these new developments will be integrated in one of the next ACS releases and that they will bring benefits to ALMA as well.

For more information, see also the ACS Web Page: http://www.eso.org/projects/alma/develop/acs/.

REFERENCES

[1] G. Chiozzi et al., “The ALMA common software: a developer friendly CORBA-based framework”, Proc. SPIE Vol. 5496-23, Astronomical Telescopes and Instrumentation, Glasgow, June 2004.

[2] G. Chiozzi et al., “Application development using the ALMA Common Software”, Proc. SPIE Volume 6274, Astronomical Telescopes and Instrumentation, Orlando, Florida, USA, May 2006.

[3] G. Raffi, B.E.Glendenning, “ALMA Software Project Management – Lessons Learned” ICALEPCS2009, Kobe, Japan, October 2009, these proceedings.

[4] H. Sommer, G. Chiozzi, “Container-component model and XML in ALMA ACS”, Proc. SPIE Vol. 5496-24, Astronomical Telescopes and Instrumentation, Glasgow, Scotland, June 2004.

[5] J. Avarias H. Sommer G. Chiozzi, “Data Distribution Service as an alternative to CORBA Notify Service for the ALMA Common Software, ICALEPCS2009, Kobe, Japan, October 2009, these proceedings.

[6] J. Schwarz et al., “The ALMA Common Software — Dispatch from the trenches”, Proc. SPIE Vol. 7019-32, Astronomical Telescopes and Instrumentation, Marseille, France, June 2008.

[7] D. Fugate, “ A CORBA Event System for ALMA Common Software”, Proc. SPIE Vol. 5496-23, Astronomical Telescopes and Instrumentation, Glasgow, June 2004.