ACS is an open source project providing the technical infrastructure for the software of the
Atacama Large Millimeter Array and several other projects.
ACS provides a framework for the development of distributed systems based on the Component/Container paradigm and a set of basic services like:

- Transparent remote object invocation,
- Publisher/subscriber paradigm
- System deployment/administration and object location
- Distributed error and alarm handling,
- Distributed logging,
- Configuration database,

In this presentation I will give an overview of ACS: basic concepts, history, status of collaboration and adoption, future perspectives, lessons learned.

Contents

- What is ACS?
- Why adopting a Technical Infrastruture Framework?
- Platforms
- ACS main characteristics
- The ACS Community
- Conclusion and lessons learned
- Questions and discussion

2

In this presentation I will first introduce briefly the concept if technical infrastructure framework to provide a context for the ACS project.

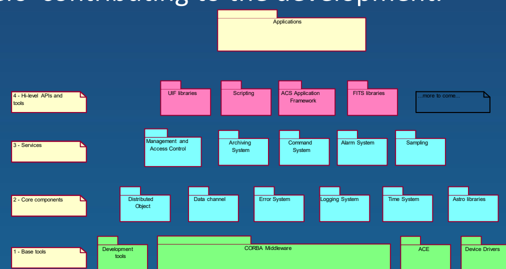
Then I will introduce ACS and its main features and characteristics.

I will then spend a few words about the community of users of ACS, a part form ALMA, and go for conclusions.

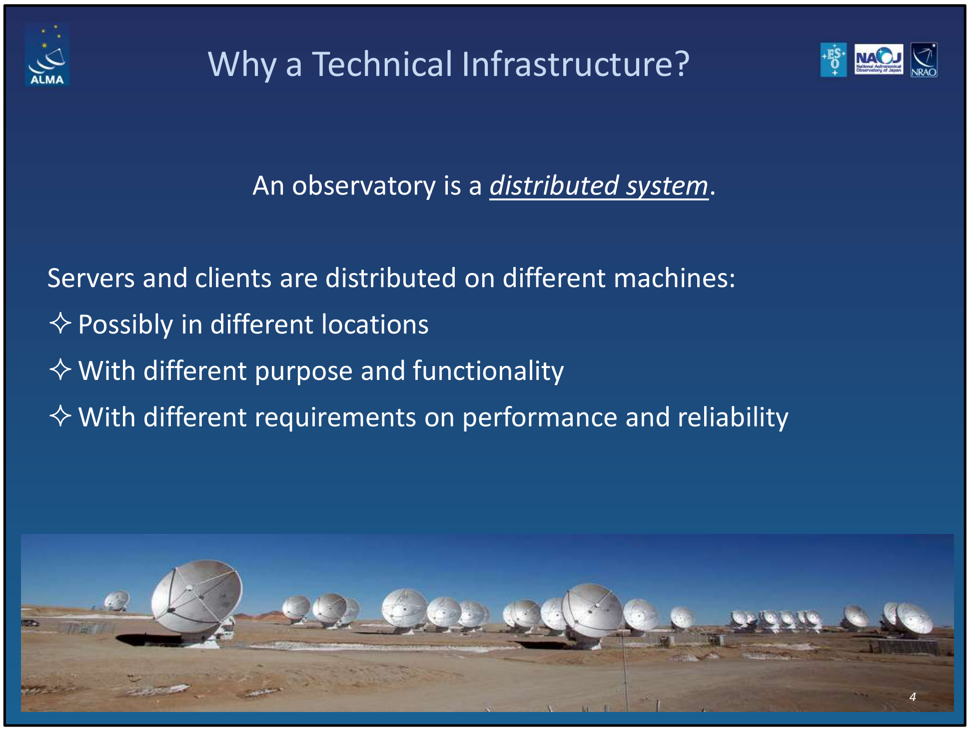
I hope we will remain with a few minutes for questions and a short discussion.

What is ACS?

- ACS is a SW Technical Infrastructure for Control Systems.
- ACS provides the basic services for OO distributed computing.
- ACS is based on a Component/Container model
- Development started in 1999 for ALMA, as an open source project.
- ACS is still actively developed.
- ACS is used in a number of projects outside ALMA
- There is a community of users contributing to the development.



3



The architecture of an observatory is very distributed.
Servers and clients need to be distributed in different locations inside and outside the physical observatory where the telescope resides.
The different parts of the system have different purpose and functionality and therefore have different requirements on performance and reliability.

If we take into account that parts of the system are dedicated to real time control of hardware, coordination, database management, data analysis up to the GUIs on the astronomer's desktop, we see that this distribution involves something more than a plain Distributed System.

Why a Technical Infrastructure? (2)

An observatory is a ***heterogeneous*** distributed system.

Servers and clients may use different:

- Hardware
- System software
- Programming languages

Even development is distributed

5

What we really have is a *Heterogeneous* Distributed Systems, since the distribution involves different:

•Hardware platforms and architectures. From field control devices like PLCs to real time computers, to PCs of any kind on the desktops, we can have very different hardware architectures (CPU, word size, alignment, memory available…)

•System software. Any of these machines can have a real time operating system, Linux or other variants of Unix, Microsoft operating systems, PLCs.

•Programming Languages. Different programming languages are more suitable for different application domains. For example, C and C++ are most suitable for real time and CPU intensive applications, while Java fits well in coordination, high level or GUI developments. Astronomers will want to write their observation scripts and reduction procedures in high level scripting languages like Python.

Why a Technical Infrastructure? (3)

Transparent heterogeneous distribution is desirable:

- Application developers should be unaware of the underlying server architecture & vice-versa
- It should be possible to change the architecture of a server transparently to the client
- Application developers should not even need to know whether a server is local or remote.

6

In order to achieve the "separation of concerns" objective, applications developers have to be unaware of the architecture (hardware, software, programming language, location) of the servers they interact with.
Having to deal explicitly with network communication protocols, byte order of message data, connection reliability and similar problem would be a major burden on the shoulders of the application developer.
The technical framework has to take up this responsibility and hide all these problems to the functional developers.

It shall even be possible to fully replace the server with a different one without the client noticing.

We could (and this has been often the case in past projects) keep the heterogeneous domains separate. For example data analysis and control system could be implemented using different and independent software infrastructures, but this approach will lead to many problems in the interfaces. In the past, interfaces were limited and this was not an important issue. But the level of integration needed nowadays makes such a choice highly problematic.

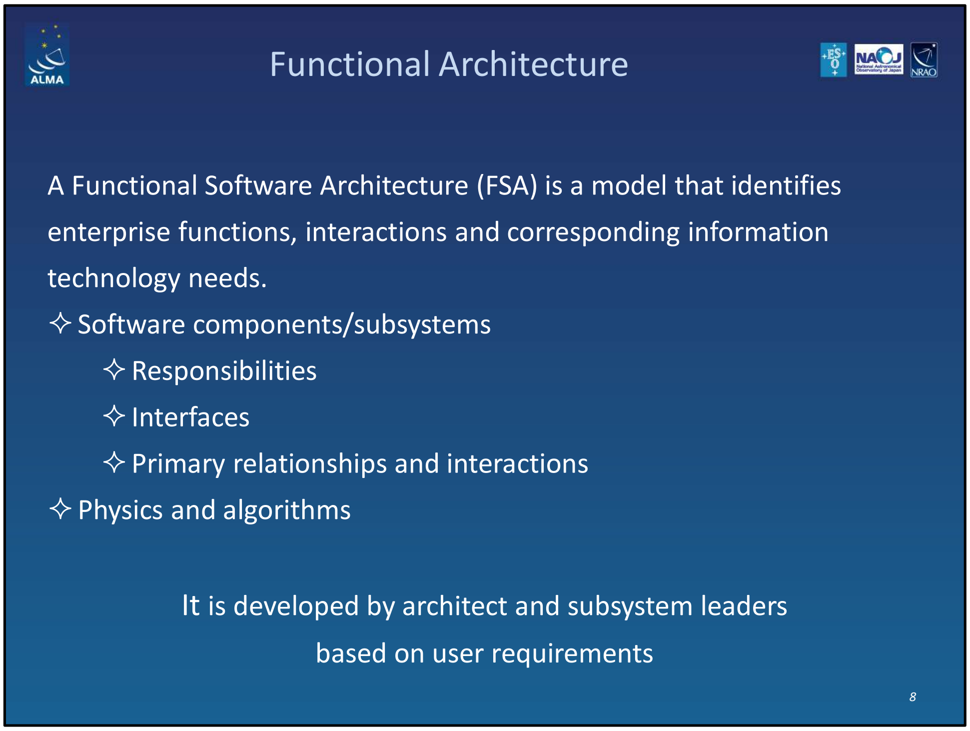
The infrastructure Framework has to take care of these aspects of the system.

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 all the subfields of computer science associated with distributed object architecture, such as 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.

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

The technical architecture provides developers of these subsystems 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.

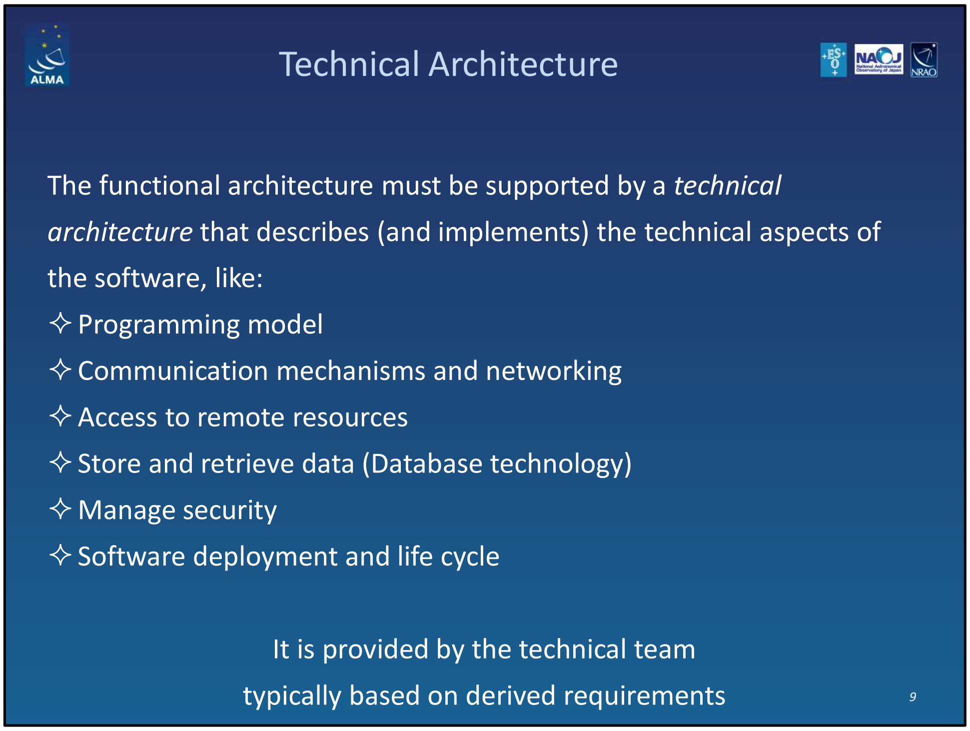


The functional architecture is built based on the user requirements.
The functionality that needs to be implemented is assigned to components/subsystems and the architecture describes the responsibilities of each subsystem and the interfaces that are exposed to the other subsystems or to the external world.

Then the relationships between the subsystems (i.e. how these interfaces are used when asking reciprocally services) are described.

The functionality must be implemented according to the physics of the system and must implement specific algorithms that must be described in this architecture. For example scheduling algorithms, control algorithms, data reduction strategies are all part of the functional architecture.

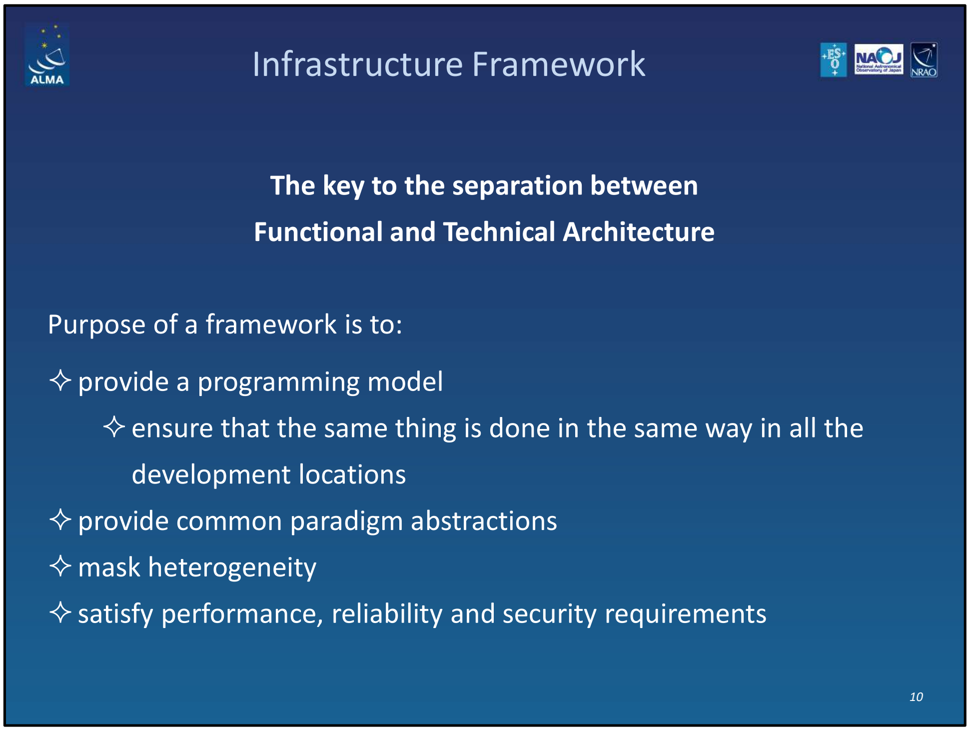
Another essential driving factor is the actual deployment and distribution of the hardware that must be controlled by the software. For example, the physical deployment of motors and sensors and the physical connection of the electronics to the control computers affects the functional architecture of the system. Or the location of the data archives and of the CPU factories for data reduction.



The "functional architecture" must be supported by a "technical architecture" that describes (and implements) the technical aspects of the software, like the communication protocols used, the threading model, the software deployment (process handling, distribution, activation and deactivation).

The requirements for the technical architecture are mostly derived requirements.

While the user requirements are the basis for the development of the functional architecture, we derive most of the technical requirements from the functional architecture itself: the technical architecture shall enable us to implement the functional architecture.



The key to reach this objective is to adopt a Software Framework that provides a consistent infrastructure for the whole observatory. On one side the framework has to satisfy all the requirements of performance, reliability and security derived from the functional architecture. On the other side it must hide as much as possible its own internal complexity to the subsystem developers and provide them with a clear and streamlined programming model.

What can be a definition of software framework?

The current definition from the Wikipedia (http://en.wikipedia.org/wiki/Software_framework) is:

A **software framework** is "the skeleton of an application that can be customized by an application developer". Like software libraries, it aids the software developer by containing source code that solves problems for a given domain and provides a simple API. However, while a code library acts like a servant to other programs, software frameworks reverse the master-servant relationship. This reversal, called "inversion of control", is the essence of software frameworks.

Frameworks are designed with the intent of facilitating software development, by allowing designers and programmers to spend more time on meeting software requirements rather than dealing with the more tedious low level details of providing a working system. However, there are common complaints that using frameworks adds to "code bloat", and that a result of competing and complementary frameworks is that one trades time spent on programming and design for time spent on learning frameworks. Having a good framework in place allows the developers to spend more time concentrating on the business-specific problem at hand rather than on the plumbing code behind it. Also a framework will limit the choices during development, so it increases productivity, specifically in big and complex systems.

However you can find many definitions pushing more of less on certain aspects of the concept of framework and even the definition in the Wikipedia has been quite volatile. The E-ELT project has written a Technical Requirements document for the TCS Software Framework. This document is used for the evaluation of the different alternatives. This document states that:

The role of the **Software Framework product** is to allow the control software applications to communicate in this distributed environment and to enforce a coherent integrated system. The Framework hides the operating systems from the application, provides common services and provides an API. The Framework may or may not include dedicated tools to generate applications, e.g. code generators, so called Application Framework. It is emphasized that the priority in this document is on the support structure.

The justification of using a Framework is to make application development easier, by providing common programming abstractions, by masking heterogeneity and the distribution of the underlying hardware and operating systems, and by hiding low-level programming details. The advantages of using a Framework come with potential caveats. These shall be taken into account when selecting and/or developing a Framework.

Which framework to chose?

All big projects have adopted an infrastructure framework

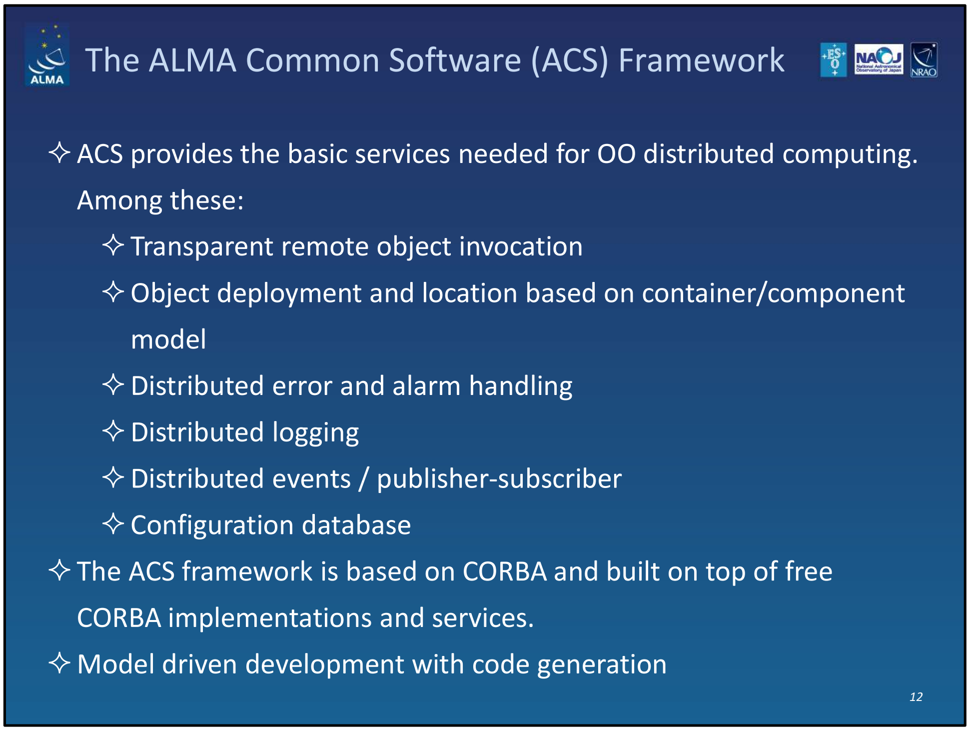
ACS in just one among several options, like

- ACS
- EPICS
- TANGO
- ESO VLT CCS
- ESO ELT CII
- ....

They are all rooted on the same basic principles described above.

They make specific technical choices and have an own history and a rationale for adopting any of them in a project, or to create a new one.

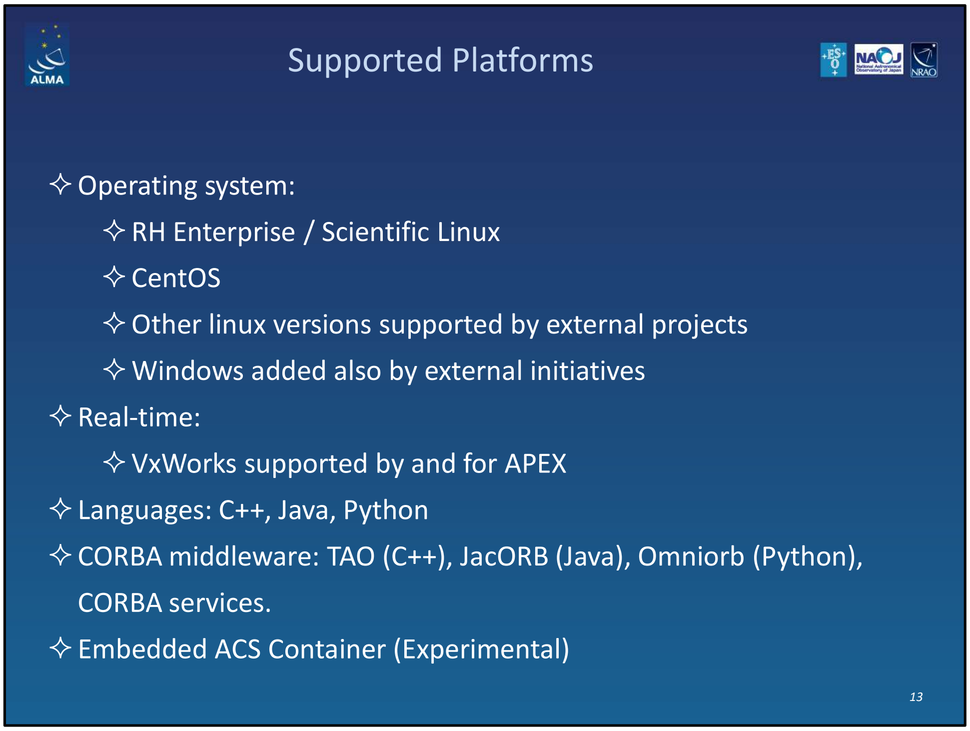
11



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

Transparent remote object invocation
Object deployment and location based on a container/component model
Distributed error and alarm handling
Distributed logging
Distributed events

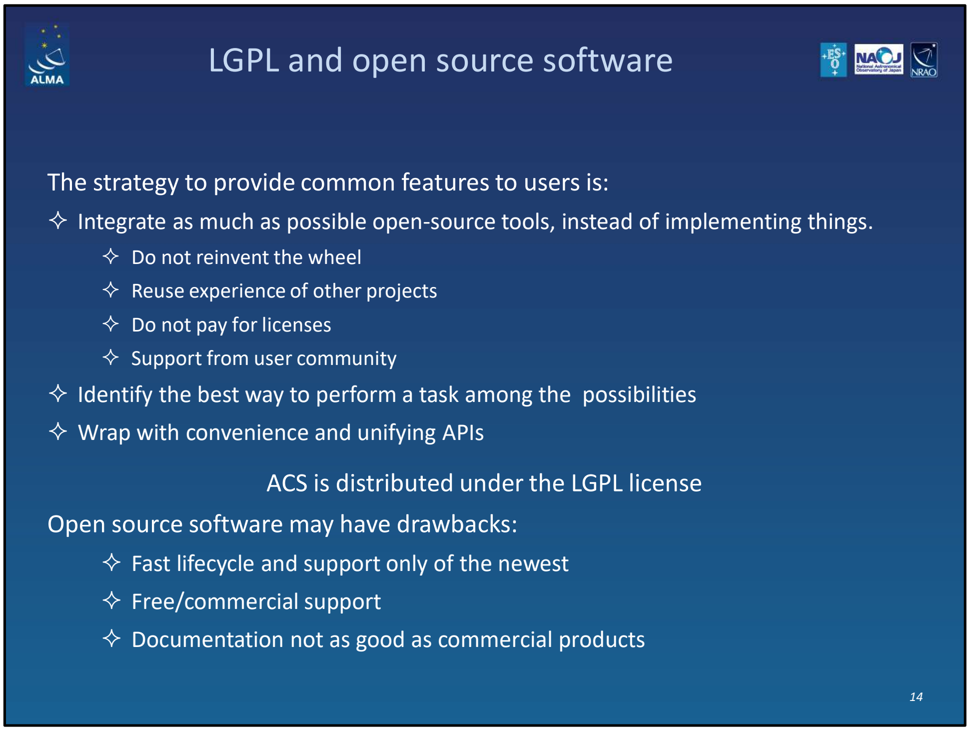
The ACS framework is based on CORBA and built on top of free CORBA implementations.
Free software is extensively used wherever possible, to avoid "re-inventing the wheel".
ACS itself is publicly available under the Lesser GNU Public License (LGPL) license
ACS's primary platform is Red-Hat Enterprise Linux, but it works and is used also on other Linux variants.
Real time development is supported on Real Time Linux (for ALMA) and VxWorks (for other projects).
Development is supported 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.



The platforms and development environments supported by ACS are decided for each release, based on the requests coming from the user's base.
Our main development platform is Linux, while real time systems run under RTAI and VxWorks.

- Linux (RH Enterprise and Scientific Linux 4) development and run time
- RTAI (Linux Kernel version 2.6.10, RTAI 3.2)
- Cross development for VxWorks (Tornado 2.5) from Linux, upload on VxWorks run time and debugging

Support for small-footprint run time only installations is foreseen.
Other platforms are supported by external groups.



One of the first key decisions for the development of the ALMA software has been the one of embracing the free-software philosophy.
The problems we face in the design of our system are similar to the problems encountered by other projects,
The adoption of commercial packages ties one to specific vendors, often with license costs that would be prohibitive for the budget of our project and with the high risk of being affected by changes in the commercial strategy of the vendor.
Therefore we decided to build our software infrastructure by taking advantage of the experience of other projects, using as much as possible freely available and at the same time widely used software.
A wide open software community promises also good and fast support through the usage of newsgroups and discussion forums. Open community forums are very active and replies come very often within a few hours.
The lifecycle of open software is very fast and there is no or little support for older versions.
When a bug is identified, the fix usually arrives very quickly, but it is almost always tied to the latest, "bleeding-edge" version of the software. Patches to previous versions are rare.
Accepting the fix thus often means accepting new features, backward incompatibilities and, perhaps, new bugs. The alternative is patching the old code ourselves (this is possible since the source code is freely available).
When an open software product really becomes mainstream the resources that the

authors would have to put in support become quite substantial. We have seen that very often at this point a company is founded to provide support and consultancy as a way to pay the costs. This corresponds normally to a sharp decline in the contribution of the core authors to the newsgroups, in order to convince the users to purchase support from the company.

Documentation for free software is very different from the documentation you normally expect from commercial products. First of all it is very different from package to package. Detailed and comprehensive user and reference manuals are typically absent. Very often we end up having to look inside the source code.

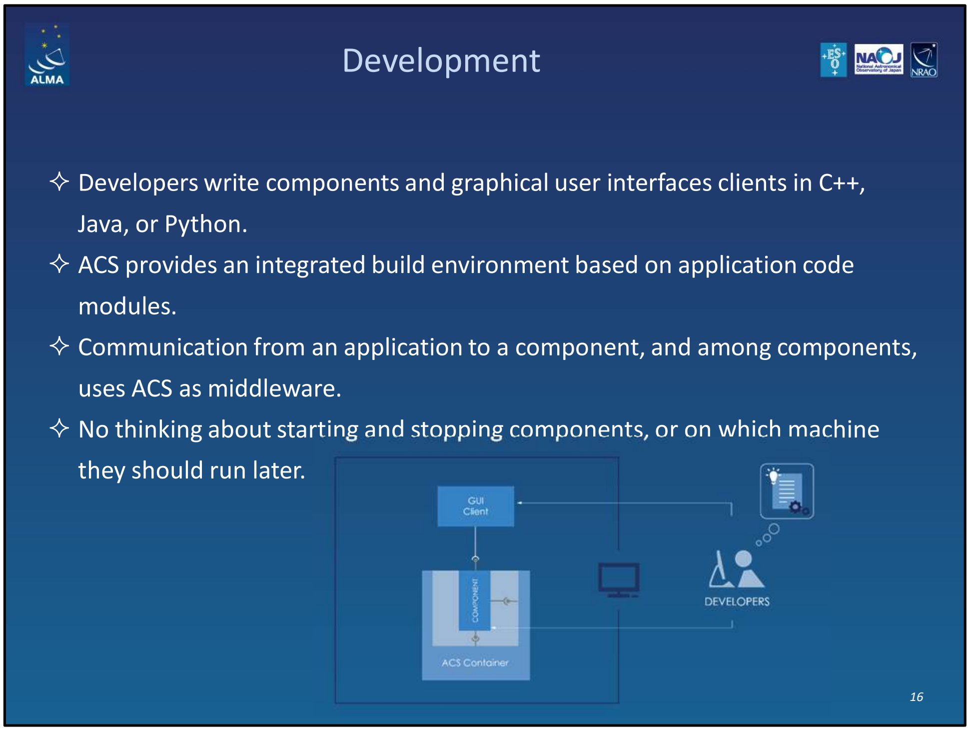
We have seen for example, that the costs during the past year for keeping pace with real-time Linux releases and having a stable system have been much higher than originally foreseen. We have been often forced to take versions of the real-time Linux development software that is "hot off the press," just to get basic features running, although reported problems have been fixed quite promptly.

Separation of roles

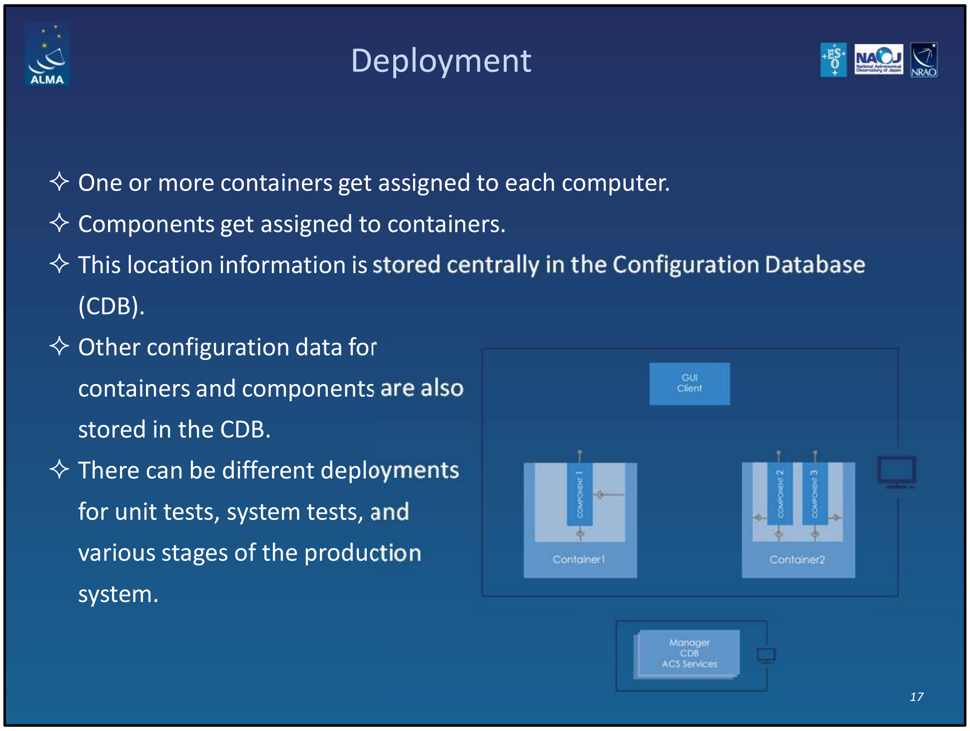
ACS keeps separate 3 roles/phases:

- ✧ Development by software developers
- ✧ Deployment by operations engineers
- ✧ Runtime by system operators (clients)

15



•Most ALMA software is written as Components, which have no GUI.
•The concept of Container / Component will be explained in separate presentations.
•Mainly the scientists (writing Observation Projects or researching the archive), as well as the ALMA operators, will see GUI clients. These are written by the ALMA subsystems ObsPrep, Exec, Pipeline/QuickLook, and Archive. ACS provides an optional GUI framework called “Abeans” which is particularly aimed at writing control applications GUIs. We will not touch on GUIs in the ACS course though.
•ACS allows to easily write distributed applications. The application developer has to write software that conforms to the standards. The reward is an application that can later run on one or many machines, without coding overhead for remote communication or starting and stopping the system.



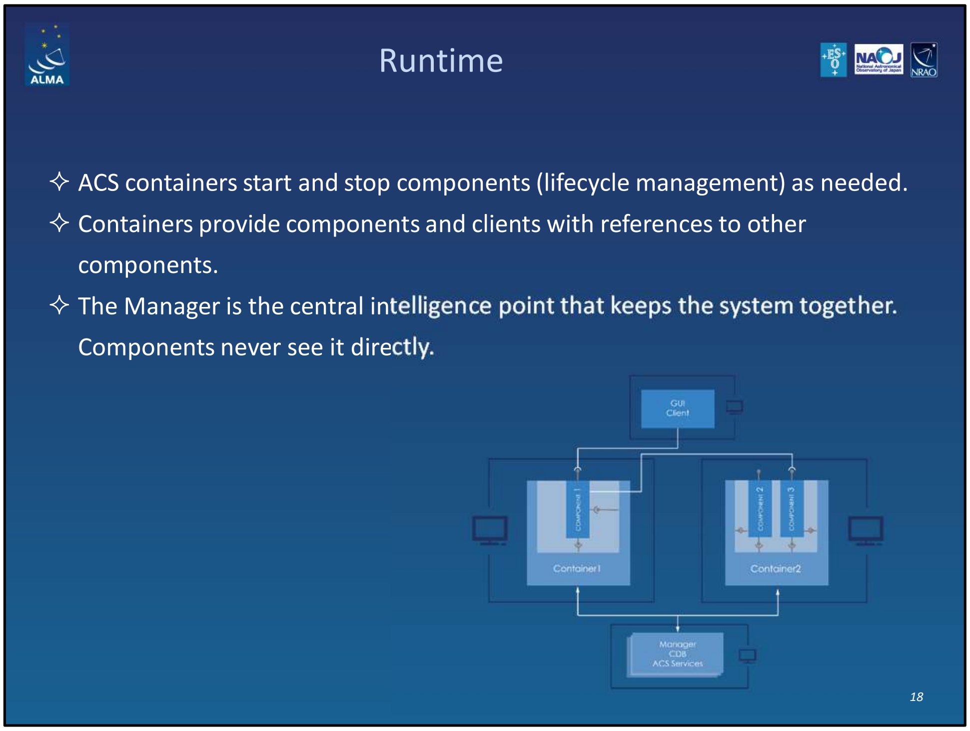
Details on container location information and container startup:

• for the system to work, it is good enough to start containers by hand on any machine. They dynamically add themselves. This is only done for tests though.

• In the real ALMA, the central starter application “Executive” starts containers on various machines

  •either based on a custom configuration file that assigns containers to machines,

  •or based on container-machine location data from the CDB from which the manager can start containers.



Details on container location information and container startup:

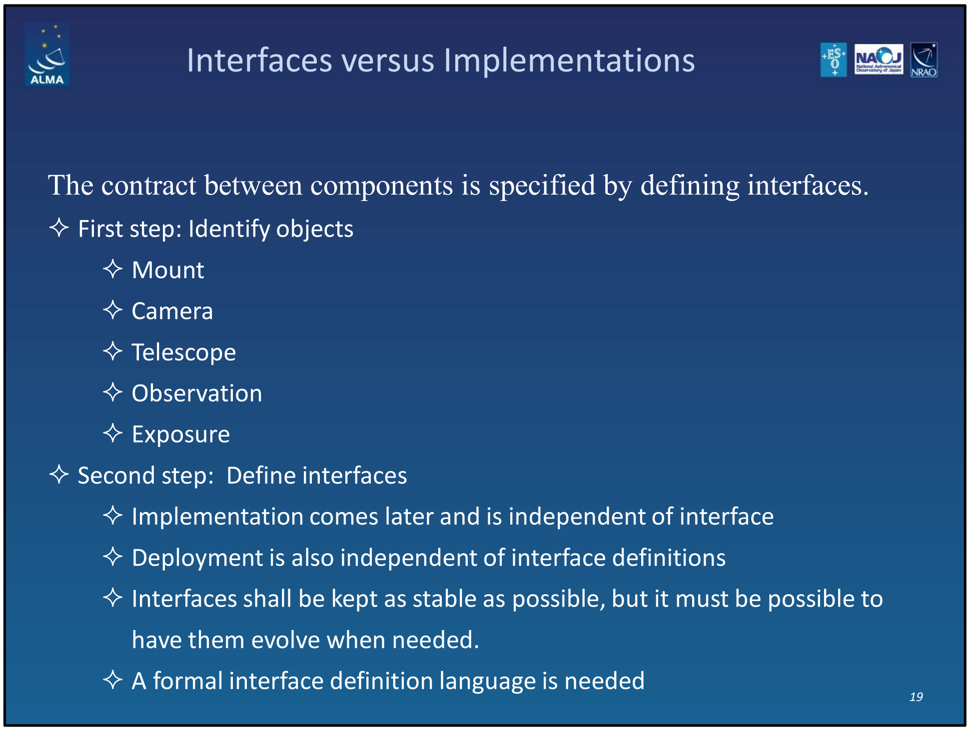
• for the system to work, it is good enough to start containers by hand on any machine. They dynamically add themselves. This is only done for tests though.

• In the real ALMA, the central starter application "Executive" starts containers on various machines, before any application software gets run. Exec maintains a configuration file that assigns containers to machines.

• Soon the CDB will optionally include container-machine location data.

•Services:

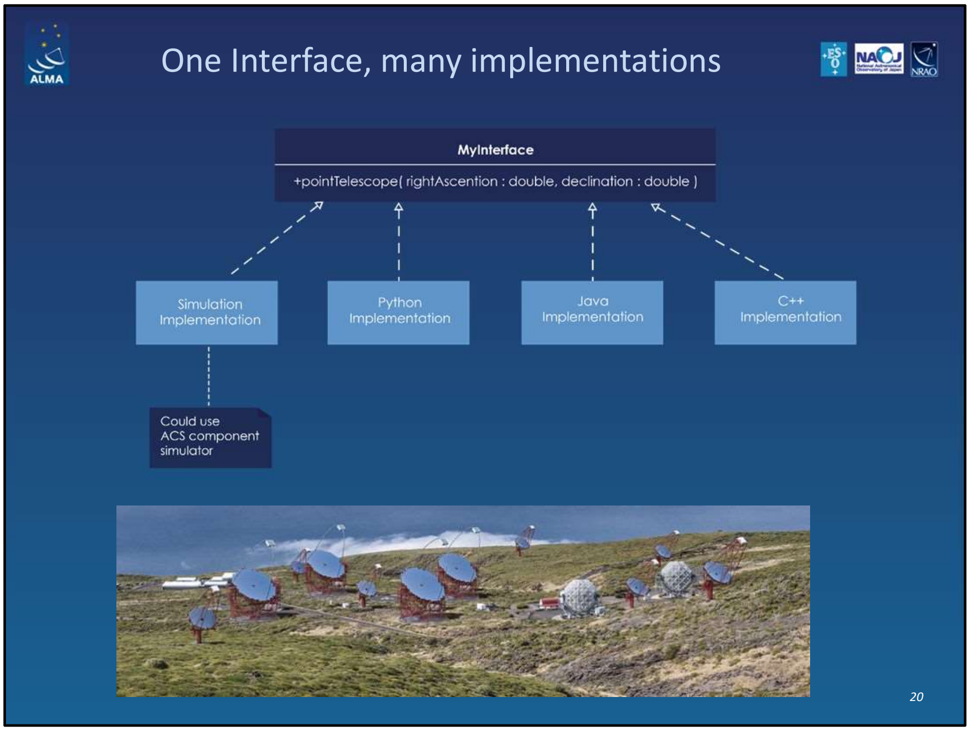
- •Error propagation across processes
- •Logging
- •Alarm
- •Event-based notification
- •Bulk data transfer
- •Other services can be plugged into the container framework if necessary (e.g. security)



The contract between the components is specified by clearly defining interfaces.

As a first step in the analysis and design of the system we have to identify the objects that will interact together.
Typically this will be done in layers.
Per each subsystem we will identify the outer layer of objects that will be used in the interactions between subsystems.
Going deeper in the analysis we will identify recursively internal layers of objects.
Once the objects have been identified, we will have to define their interfaces.
At this point we should not care about implementation and deployment.
It shall be possible to implement the interfaces later on using different programming languages and different architectures, as well as deploying the implementation in different ways.
The absolute separation between interface and implementation is essential to interoperability and scalability.
The best way to define interfaces if by using an implementation neutral but formal interface definition language that will be mapped in the implementation languages later on. Using a formal language is very important to avoid surprises and inconsistencies when integrating subsystems developed by different teams. Using just a textual Interface Control Document (ICD) can very easily lead to problems.
The clients of an object know and see only its interface and the interface shields completely the implementation underneath.
This makes it possible first of all to implement a servant in any language.
But it also means that it is possible:

•To have different implementations for the same interface, if needed in multiple languages.
For example one could provide a mock up implementation in Python for testing and an high performance servant in C++ for the final real time system.
•To have one implementation serve multiple interfaces.
For example, access to a legacy system could be done defining the interfaces for each subsystem but implementing only one generic servant (for example a sort of protocol converter) able to implement all of them. Another example is a CORBA interface to access an object (or also relational) database. It is not necessary to provide the implementation for each object type (or table) in the database. One single implementation is able to “incarnate” dynamically all interfaces.
•To have one physical instance of a Servant to represent multiple logical instances. Or the other way around. Or any intermediate situation, based on scheduling and load balancing algorithms.

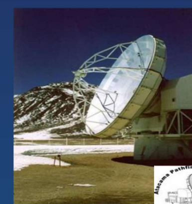


Details on container location information and container startup:
• for the system to work, it is good enough to start containers by hand on any machine. They dynamically add themselves. This is only done for tests though.
• In the real ALMA, the central starter application "Executive" starts containers on various machines, before any application software gets run. Exec maintains a configuration file that assigns containers to machines.
• Soon the CDB will optionally include container-machine location data.
•Services:
- •Error propagation across processes
- •Logging
- •Alarm
- •Event-based notification
- •Bulk data transfer
- •Other services can be plugged into the container framework if necessary (e.g. security)

The ACS Community

- ALMA
- APEX
- CTA / ASTRI
- SRT / DISCOS
- LLAMA (Argentina)
- Yebes Observatory RT40m (Spain)
- HESS
- …. Some other smaller or perspective projects ….

- Strong expertise in Italy


APEX (Chile)

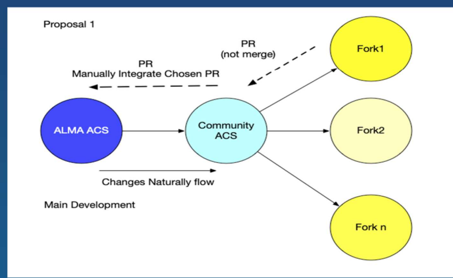

Sardinian Radio Telescope (Italy)


ASTRI (Italy)


Yebes RT40m (Spain)

How does the community work today?

- ALMA is leading ACS Maintenance (2 FTEs) and Development (Best Effort)
  - Focused on ALMA's priorities
- Preparing releases and making them available to the community
- Receiving questions, requests and suggestions from community
- Receiving patches and integrating them in ACS
- Creating tickets, following up and resolving them
- Organization of community meetings and workshops
  - Last workshop: July 2020 About 80 participants



- Web Confluence page: https://confluence.alma.cl/display/ICTACS/ACS+Community

ACS Community Objectives

- Increase Community Collaboration
  - Identify Current Community
  - Releases planning
  - Issue Tracking
  - Building / Packaging / Distributing
- Increase Community Engagement
  - More frequent community meetings
  - Better means of communication (Slack, Issue Tracking, etc.)
- Improve ACS Visibility
  - Website + Confluence
  - Improve Documentation
  - DockerHub Official Docker Image + Dev Images
  - ACS Community Slack Page
- Modernize the Framework
  - Replacement of technologies
  - New developments
  - Improve performance

Conclusions and lessons learned

- By now an “old” product
  - >20 years since inception
  - Is CORBA obsolete?
- Very stable and reliable: many years of continuous operation
- Actively supported by ALMA
- It is very difficult to engage the community in contributing
- Adoption pays off in relatively big projects
- What brakes adoption?
  - Steep initial learning curve.
    Higher level tools and more code generation would help.
  - Good documentation is critical
  - Not modular. Splitting in multiple independent packages would help but where to get resources with a relatively small community?
- ACS is getting new energy with projects like CTA and ASTRI
- There is wide expertise in Italy: it might be useful for new projects
- How to choose between the available alternative options?

Questions?

Acknowledgements
Most slides for this presentation are taken from the presentations given at the ACS Workshop 2020. ACS presentations were originally developed by the ACS team, used in many training courses since 2004. Main contributors are (listed in alphabetical order): Jorge Avarias, Alessandro Caproni, Gianluca Chiozzi, Jorge Ibsen, Bogdan Jeram, Thomas Jürgens, Matias Mora, Joseph Schwarz, Heiko Sommer plus many others

The Atacama Large Millimeter/submillimeter Array (ALMA), an international astronomy facility, is a partnership of Europe, North America and East Asia in cooperation with the Republic of Chile. ALMA is funded in Europe by the European Organization for Astronomical Research in the Southern Hemisphere (ESO), in North America by the U.S. National Science Foundation (NSF) in cooperation with the National Research Council of Canada (NRC) and the National Science Council of Taiwan (NSC) and in East Asia by the National Institutes of Natural Sciences (NINS) of Japan in cooperation with the Academia Sinica (AS) in Taiwan. ALMA construction and operations are led on behalf of Europe by ESO, on behalf of North America by the National Radio Astronomy Observatory (NRAO), which is managed by Associated Universities, Inc. (AUI) and on behalf of East Asia by the National Astronomical Observatory of Japan (NAOJ). The Joint ALMA Observatory (JAO) provides the unified leadership and management of the construction, commissioning and operation of ALMA.