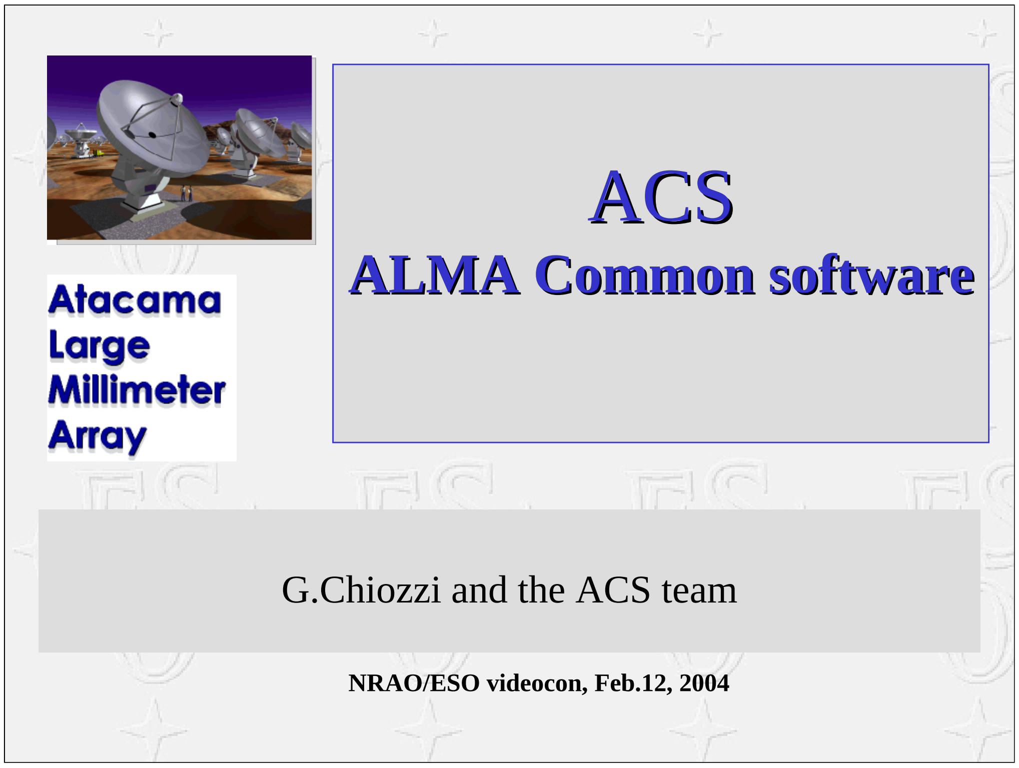
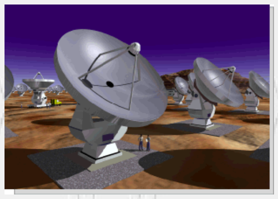




Atacama
Large
Millimeter
Array

ACS
ALMA Common software

G.Chiozzi and the ACS team

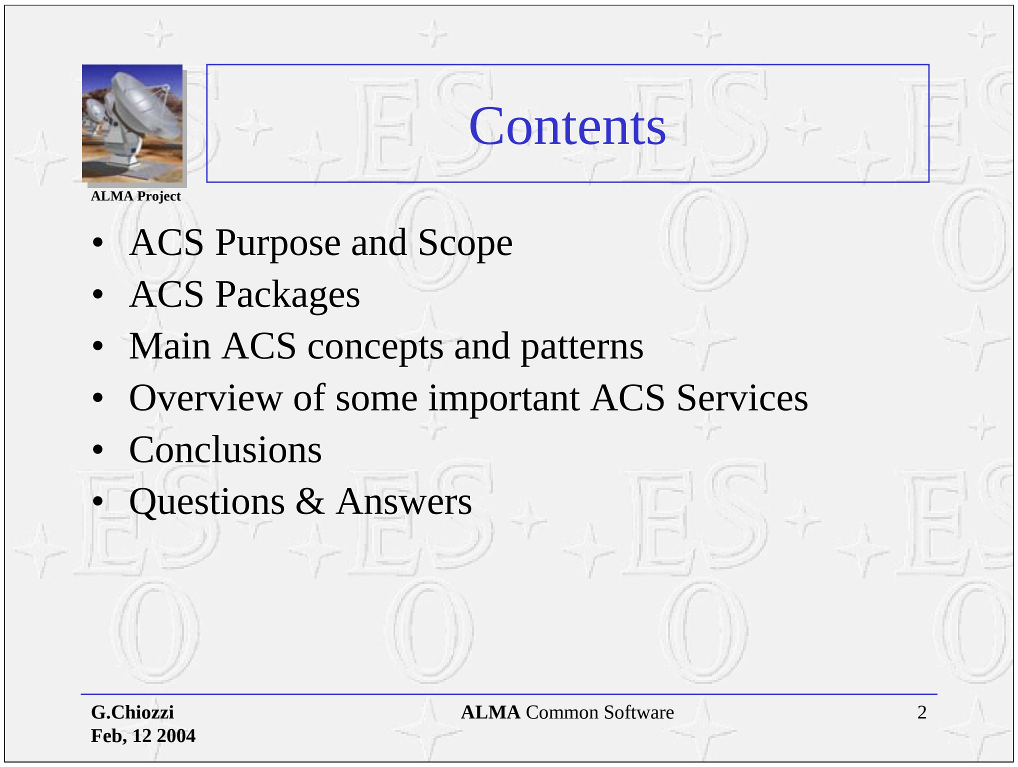
NRAO/ESO videocon, Feb.12, 2004

The Atacama Large Millimeter Array (ALMA) is a joint project between 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, with baselines up to 14 km. It will be located at an altitude above 5000m in the Chilean Atacama desert.

The ALMA Common Software (ACS) is a set of application frameworks built on top of CORBA to provide a common software infrastructure to all partners in the ALMA collaboration. The main purpose of ACS is to simplify the development of distributed applications by hiding the complexity of the CORBA Middleware and guiding the developers to use a documented collection of proven design patterns.

ACS is used in ALMA to cover from Control System development up to high level coordination and data flow applications. C++ is the the language of choice for high performance and real time applications in the Control System domain, while Java is considered the most suitable general purpose development language for higher level and coordination applications, also in the Control System domain. Python is used a scripting language and glue.

Using standard CORBA services, ACS implements a Component/Container model that is language and platform independent. Containers written in C++, Java and Python manage the lifecycle of components implemented in these languages and provide them a very simple way to access common centralized services like logging, alarming, error handling, configuration database, archive, object location and, at the same time, hiding most of CORBA. Clients written in any CORBA-aware language can access these Containers and Components while the implementation of the servant side in any other of these languages would be easy.



ALMA Project

Contents

- ACS Purpose and Scope
- ACS Packages
- Main ACS concepts and patterns
- Overview of some important ACS Services
- Conclusions
- Questions & Answers

G.Chiozzi
Feb, 12 2004

ALMA Common Software

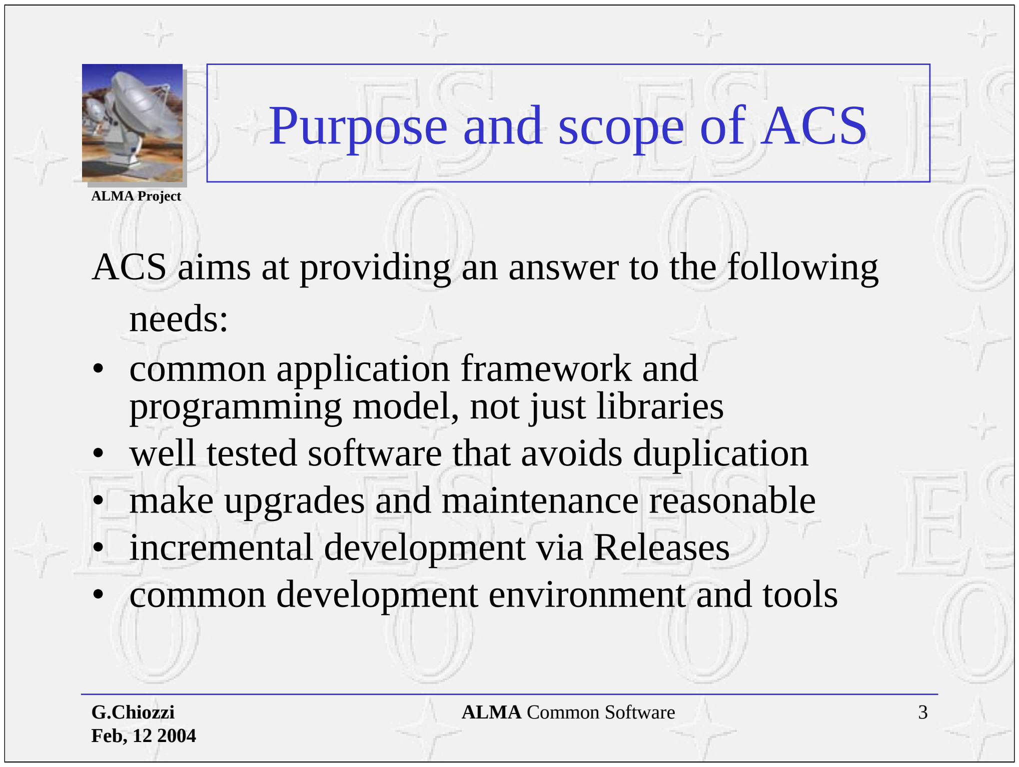
2

The heart of ACS is an Component/Container model, Components implemented as CORBA objects. For example, the teams responsible for the control system development use Components as the basis for devices, like and antenna mount control. A code generator creates a Java Bean for each Component. Programmers can write Java client applications by connecting those Beans with data-manipulation and visualization Beans.

ACS is based on the experience accumulated in the astronomical and particle accelerator contexts, reusing and extending concepts and components. Although designed for ALMA, ACS has the potential for being used in other new control systems, since it implements proven design patterns using state of the art, stable and reliable technology.

The ACS package provides Java, C++ and Python support on Linux and few other selected platforms. The complete ALMA SW development and in particular for the Control System of the ALMA Test Interferometer, currently used for the evaluation of the three ALMA prototype antennas, are based on ACS. Also other projects are collaborating with ACS, already using or evaluating it, since ACS is publicly available under the LGPL license. In particular the ANKA Synchrotron in Karlsruhe is in scientific production, the APEX radiotelescope in Chile is under commissioning and the 1.5m Hexapod Telescope in Chile is in an advanced implementation stage.

This presentation describes the architecture of ACS and its status, detailing the object model and some services.



ALMA Project

Purpose and scope of ACS

ACS aims at providing an answer to the following needs:

- common application framework and programming model, not just libraries
- well tested software that avoids duplication
- make upgrades and maintenance reasonable
- incremental development via Releases
- common development environment and tools

G.Chiozzi
Feb, 12 2004

ALMA Common Software 3

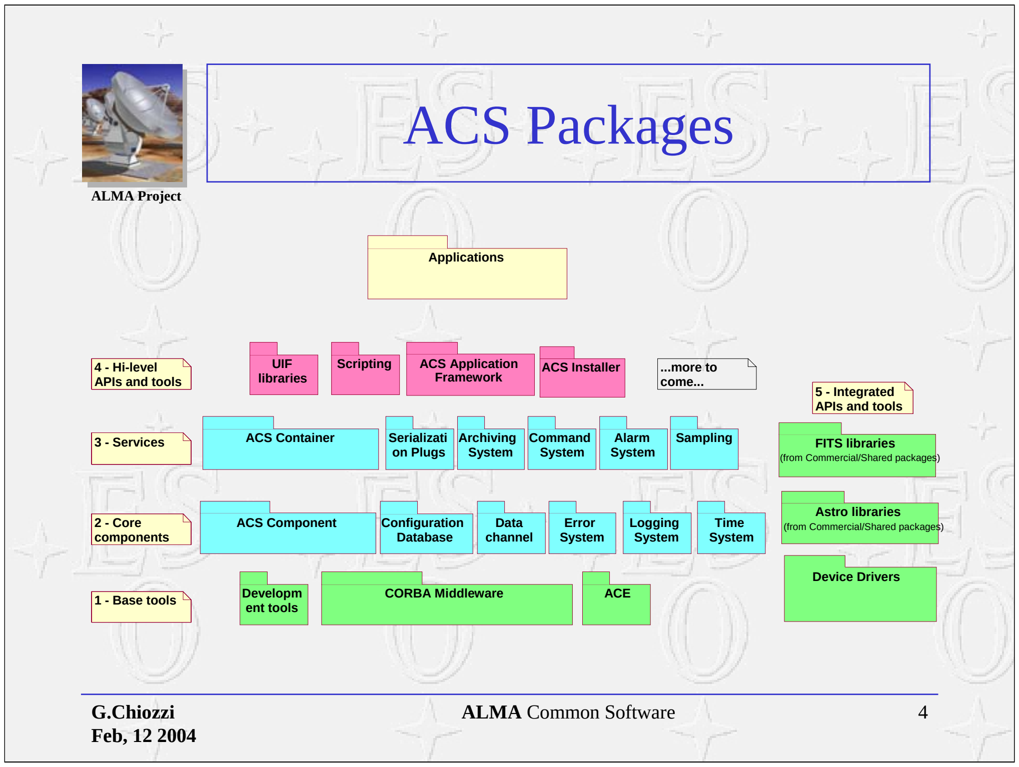
The ALMA project is characterized by the fact of being an highly distributed effort, with many sites and many "development cultures" involved. The Computing Group is scattered across 3 continents and many more institutes.

Early in the ALMA project it was decided that an ALMA Common Software (ACS) would be developed as a way to provide to all partners involved in the development a common software platform. The original assumption was that some key middleware like communication via CORBA and the use of XML and JAVA would be part of the project. It was intended from the beginning to develop this software in an incremental way based on releases, so that it would then evolve into an essential embedded part of all ALMA software applications. In this way we would build a basic unity and coherence into a system that will have been developed in a distributed fashion.

The ALMA Common Software (ACS) is a comprehensive framework on top of the operating system, offering a complete environment and structures at the base of application software developments. ACS is ment to be a general system, based on available middleware (CORBA) and the easy embedding of languages like Java and Python.

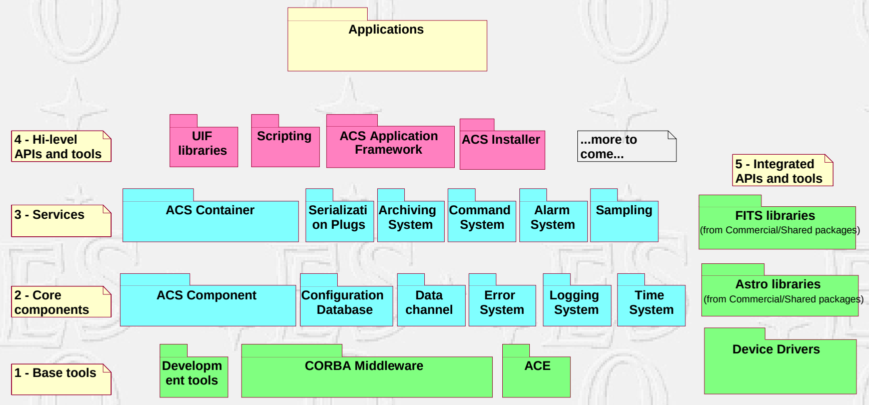
We believe that the use of a common software layer in a very geographically distributed development situation will be the best way to enforce the use of common constructs. ACS shall provide a well tested platform that embeds standard design patterns (much better than a set of written rules) and avoids duplication of effort. At the same time this will provide a platform where upgrades can be incorporated and brought to all developers. It will also standardize the underlying architecture of software modules, making maintenance affordable.

ACS is also the mean used to distribute support tools and make uniform the development environment among the various sites.



ALMA Project

ACS Packages



G.Chiozzi
Feb, 12 2004

ALMA Common Software

4

This package diagram summarizes the status of ACS with the respect to the foreseen final architecture.

It is a simplified version of the complete ACS Package Diagram from the Architecture document

Here I would discuss mainly the layers saying what characterizes each layer.

An important aspect is that the “base tools” layer is made up of “off the shelf” publicly available tools and software packages.

This includes or defines as pre-requisite for ACS installation:

- A standardized set of development tools (like compilers, Makefile extensions, installation procedures and tools, JUnit and other test support tools, emacs configuration and so on)
- CORBA implementation and services for the different languages
- ACE and other public domain libraries used by ACS and available for application developers.

I will describe the purpose of the layers one by one from bottom up and quickly list the most important packages.

Most of the packages in this diagram will be described with more details in the presentation and we will give also information about the development status.

In this layered architecture, each package is allowed to use only packages in the lower layers or in the same layer. This allows us to keep under control the dependencies between packages.

The ACS distribution will be used also to package and distribute other APIs and tools that are not part of ACS and that are not used by ACS, but are used by a number of ALMA subsystems and that is therefore convenient to distribute as one single entity together with ACS. This can include for example FITS libraries, Astronomical Calculation libraries or device drivers.

Most of the packages in this diagram are in a stable state and mainly need extensions that are foreseen in the ACS development plan according to the need of the various ALMA subsystems.

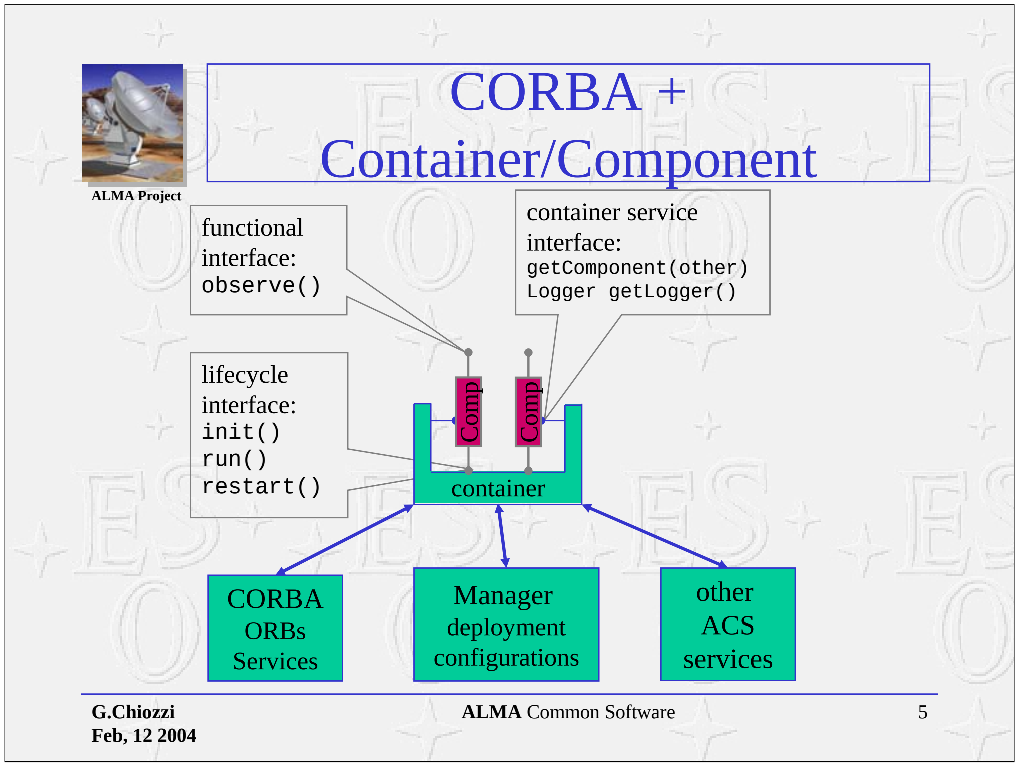
The next ACS release, ACS 3.1, will concentrate on providing as new developments:

•A first usable prototype of the ALMA Alarm System (a prototype is available already now, but is very limited and does not implement the desired architecture).

•A first implementation of the bulk data transfer

•ACS support for HTTP and email communication for higher level subsystems

•IDL Generic Simulator



CORBA + Container/Component

ALMA Project
functional interface: observe()
container service interface: getComponent(other) Logger getLogger()
lifecycle interface: init() run() restart()
Comp
Comp
container
CORBA ORBs Services
Manager deployment configurations
other ACS services

G.Chiozzi
Feb, 12 2004

ALMA Common Software

5

The ACS Architecture is founded on the Component-Container model.

A Component-Container based architecture emphasizes Separation of Concerns.

The Container hides as much as possible CORBA and the underlying architecture to the developers of Components, that can concentrate on the functional aspects of their specific Component.

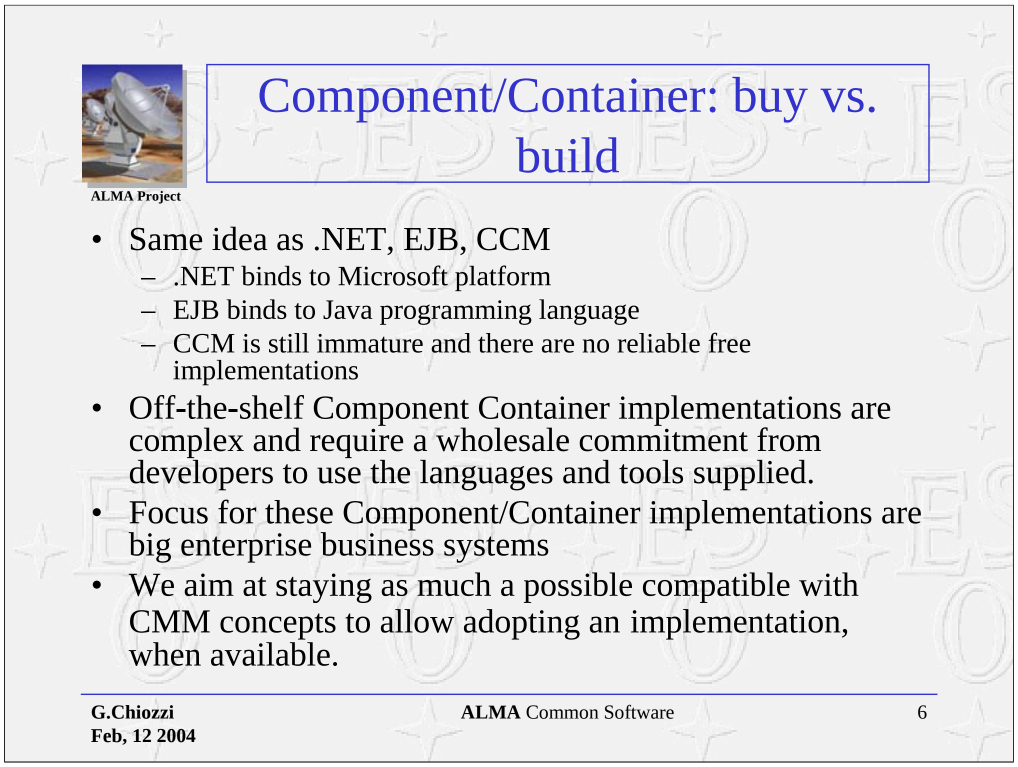
The Container manages Component's

- •Lifecycle interface (init, start, stop, update)
- •Functions interface (what component offers)
- •Optional: security, persistency, transactions…

A client (which would typically be itself be a Component residing in another Container) accesses the services of Components via their published service interfaces, defined in IDL.

Containers provide an environment for Components to run in, with support for basic services like logging system, configuration database, persistency and security. Developers of Components can focus their work on the domain-specific "functional" concerns without having to worry about the "technical" concerns that arise from the computing environment in which their components run.

The division of responsibilities between components and containers enables decisions about where and when individual components are deployed to be deferred until runtime, at which point configuration information is read by the container. If the container manages component security as well, authorization policies can be configured at run time in the same way.



Component/Container: buy vs. build

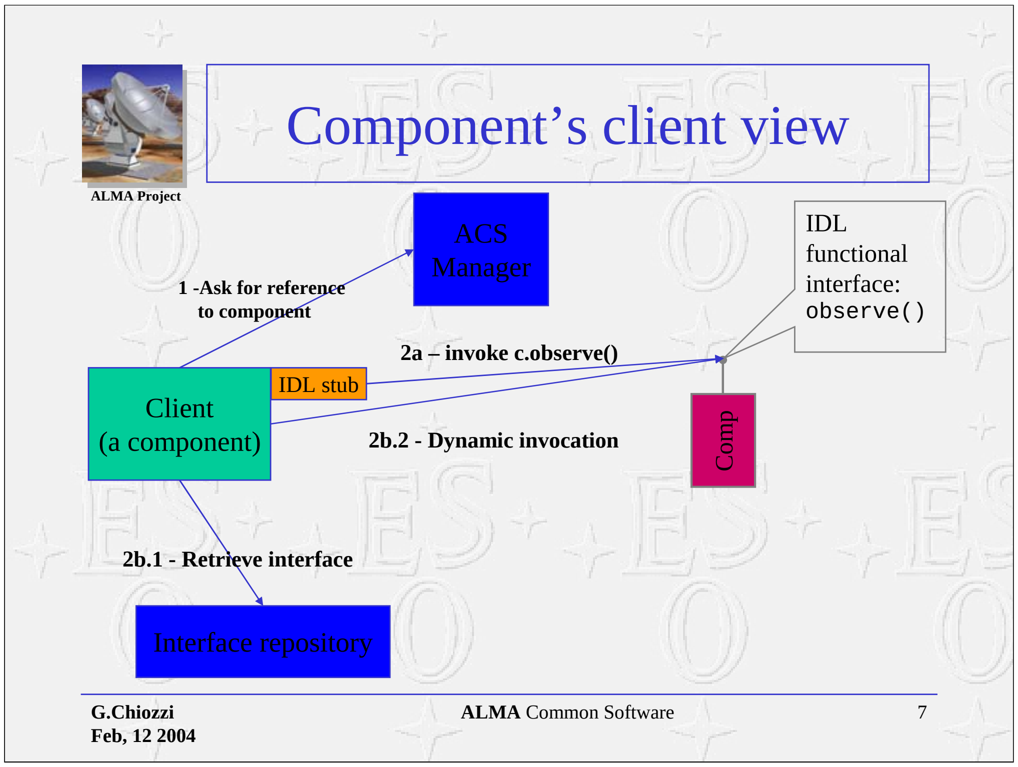
- Same idea as .NET, EJB, CCM
  - .NET binds to Microsoft platform
  - EJB binds to Java programming language
  - CCM is still immature and there are no reliable free implementations
- Off-the-shelf Component Container implementations are complex and require a wholesale commitment from developers to use the languages and tools supplied.
- Focus for these Component/Container implementations are big enterprise business systems
- We aim at staying as much a possible compatible with CMM concepts to allow adopting an implementation, when available.

Commercial implementations of the Component-Container model are quite popular (EJB, .NET).

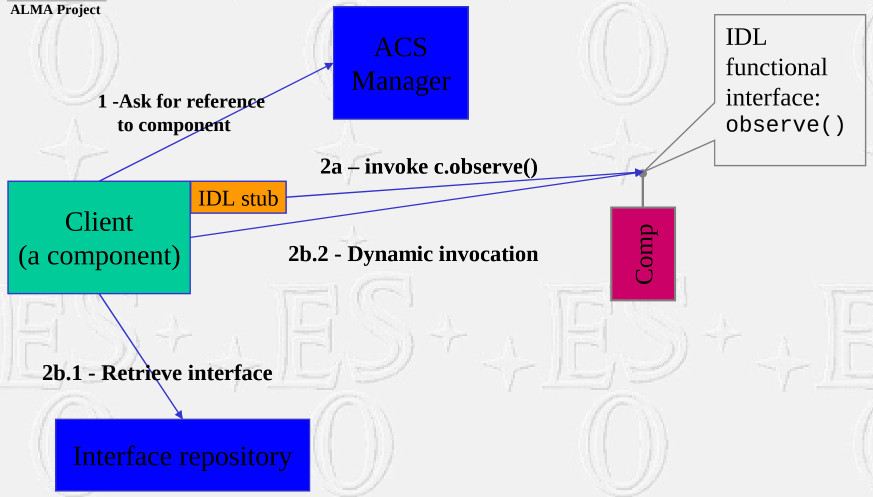
A vendor-independent specification, the CORBA Component Model (CCM), is under development, but it is not complete, and production implementations do not yet exist.

These are rather comprehensive systems, and require a wholesale commitment from developers to use the languages and tools supplied.

The learning curve for the ACS Component/Container model should be much smoother that for any of these systems, that are thought for complex enterprise distributed commercial systems



Component's client view


ACS Manager
IDL functional interface: observe()
1 -Ask for reference to component
2a – invoke c.observe()
IDL stub
Client (a component)
2b.2 - Dynamic invocation
Comp
2b.1 - Retrieve interface
Interface repository

G.Chiozzi
Feb, 12 2004

ALMA Common Software

7

A component exposes its IDL interface to clients.

A client (possibly itself a component) that wants to access the component, needs to ask the Manager for a reference.

The client is completely unaware of any deployment and lifecycle issues for the component it wants to talk to.

Once the client has a reference, can call directly the interface via the IDL stubs.

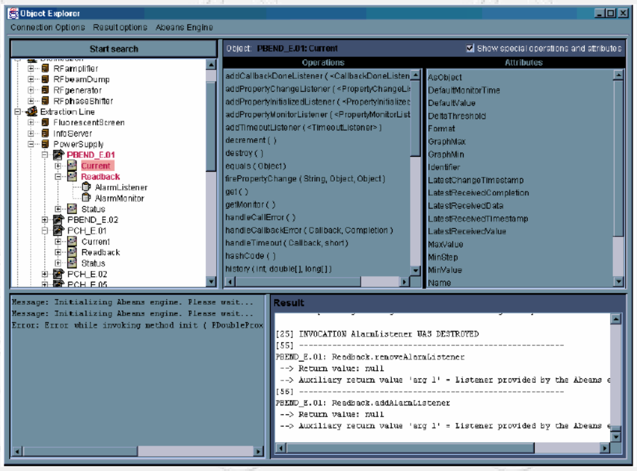
Alternatively a generic client can retrieve the interface from the Interface Repository and use Dynamic Invocation (CORBA Introspection).

The Object Explorer is such a generic client.

The Executive could also probably be also such a generic client.

ALMA Project

Object Explorer



```
Message: Initializing Abeans engine. Please wait...
Message: Initializing Abeans engine. Please wait...
Error: Error while invoking method init ( PDoubleProx
```

```
[25] INVOCATION AlarmListener WAS DESTROYED
[55] -------------------------------------------------------
PBEND_E.01: Readback.removeAlarmListener
 --> Return value: null
 --> Auxiliary return value 'arg 1' = Listener provided by the Abeans e
[56] -------------------------------------------------------
PBEND_E.01: Readback.addAlarmListener
 --> Return value: null
 --> Auxiliary return value 'arg 1' = Listener provided by the Abeans e
```

G.Chiozzi
Feb, 12 2004

ALMA Common Software

8

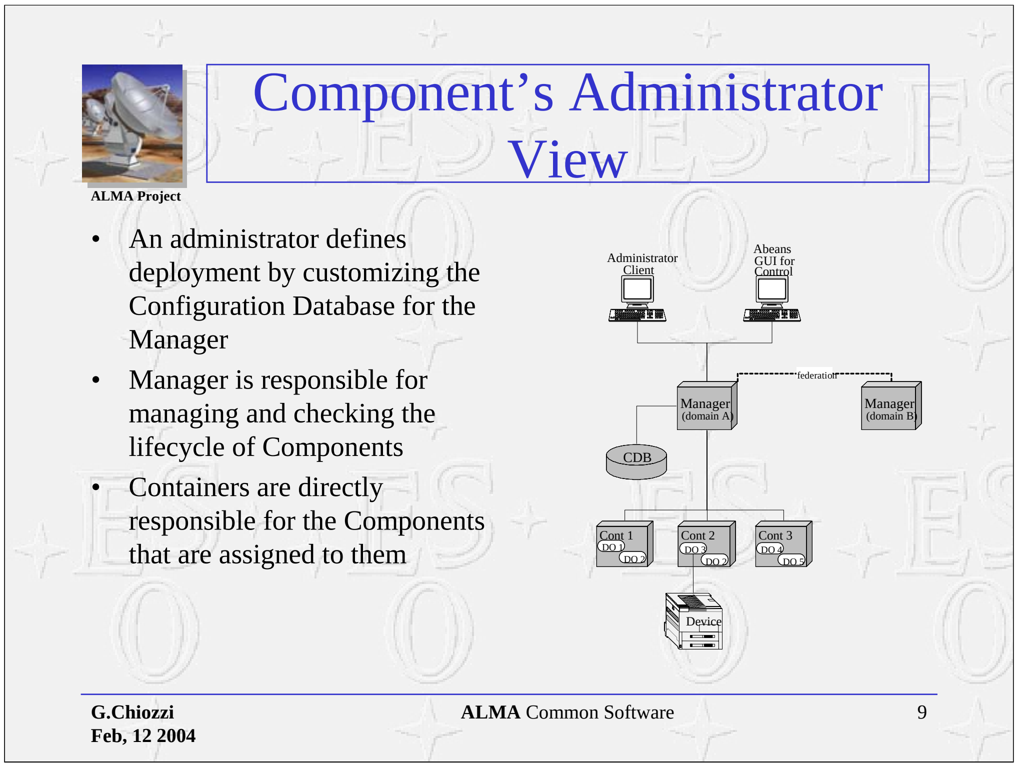
The **Object Explorer (OE):**

• Is a generic tool used for low-level inspection of objects in ACS. It can be used as a debugging or testing tool by the developers and maintainers of a system.

• Allows to interact with any CORBA Object known to the system (I.e. whose reference can be retrieved from the Manager - see following slides - and whose IDL interface can be retrieved from the Interface Repository).

• Is aware of the design patterns implemented by the Component-Property-Characteristic paradigm - see following slides - and can for example monitor the values of properties and draw trend plots, as will be shown in the demo.

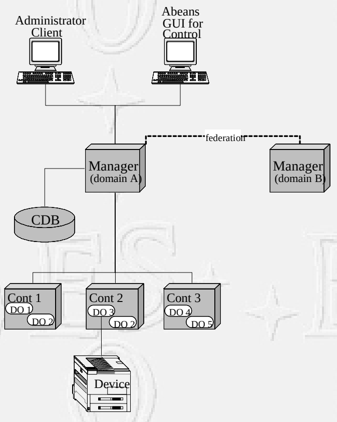
• Is a standalone Java program designed using a “plug-in” concept and has been adapted to different systems than ACS.



Component's Administrator View

ALMA Project

- An administrator defines deployment by customizing the Configuration Database for the Manager
- Manager is responsible for managing and checking the lifecycle of Components
- Containers are directly responsible for the Components that are assigned to them



G.Chiozzi
Feb, 12 2004

ALMA Common Software

9

Deployment and lifecycle of Components is implemented as a separate and independent logical layer based on two major elements:

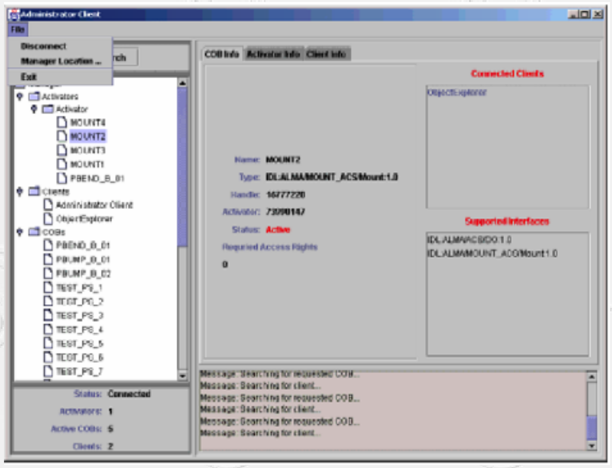
• **Containers**. They are deployed locally on all hosts involved in the system, ranging from real-time local control units to high-performance workstations. Their primary task is preparing the local environment in which Components are created, giving them all the resources they need to perform their tasks, such as CORBA connectivity, connection establishment with other Components and Configuration Database access. Containers provide the process space where Components are running.

• **Manager**, which is set up at one central location that is widely known across the entire system (Manager federation will be implemented at a second stage). The Manager is acquainted with all the Components and Containers in the system, as well as other resources, such as configuration database and CORBA services. In particular, the Manager closely cooperates with the CORBA Naming Service, in which it publishes all of its acquaintances, making them accessible to non-MACI-aware CORBA software.

The contract between Manager and Containers is defined in the Management and Access Control IDL Interface (MACI), that defines how they interact. Containers implemented in any programming language have to implement this interfaces to be administered by the Manager.

ALMA Project

Administrator Client



Activators
Activator
MOUNT4
MOUNT2
MOUNT3
MOUNT1
PBEND_B_01
Clients
Administrator Client
ObjectExplorer
COBs
PBEND_B_01
PBUMP_B_01
PBUMP_B_02
TEST_PS_1
TEST_PS_2
TEST_PS_3
TEST_PS_4
TEST_PS_5
TEST_PS_6
TEST_PS_7

Name: MOUNT2
Type: IDL:ALMA/MOUNT_ACS/Mount:1.0
Handle: 16777220
Activator: 73090147
Status: Active
Required Access Rights
0

ObjectExplorer

IDL:ALMA/ACS/DO:1.0
IDL:ALMA/MOUNT_ACS/Mount:1.0

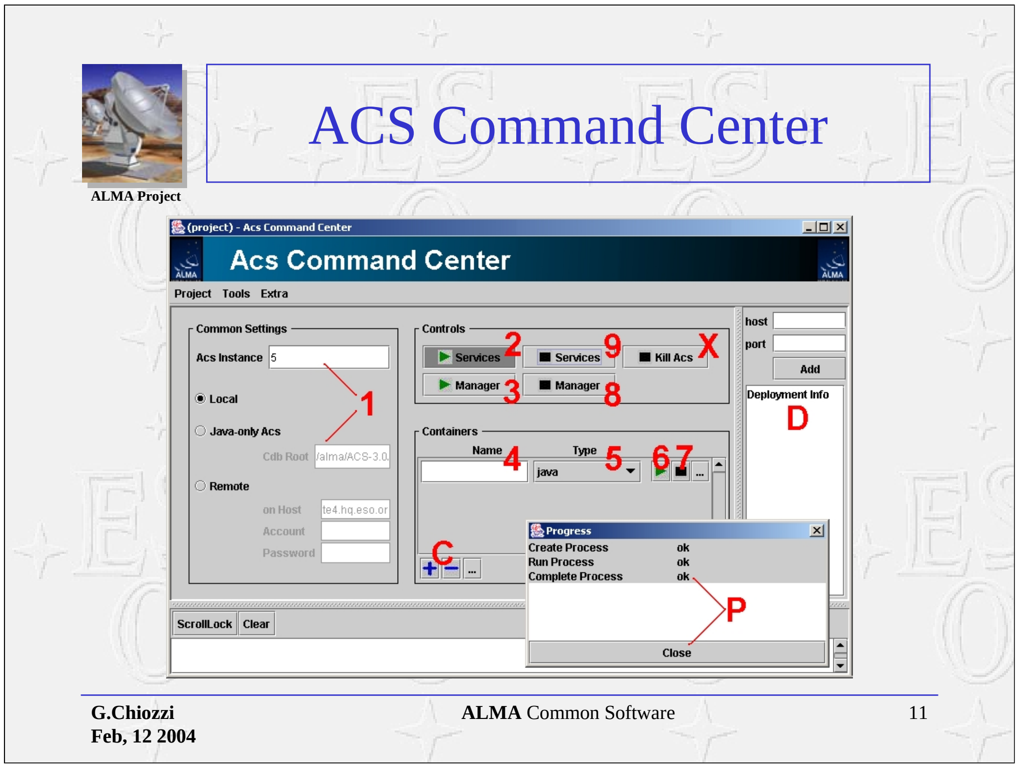
Status: Connected
Activators: 1
Active COBs: 5
Clients: 2

```
Message: Searching for requested COB...
Message: Searching for client...
Message: Searching for requested COB...
Message: Searching for client...
Message: Searching for requested COB...
Message: Searching for client...
```

Administrator Client (AC) uses a specific Administrator interface part of MACI to allow ACS System Administrators to inspect and configure the deployment view of a running ACS system.

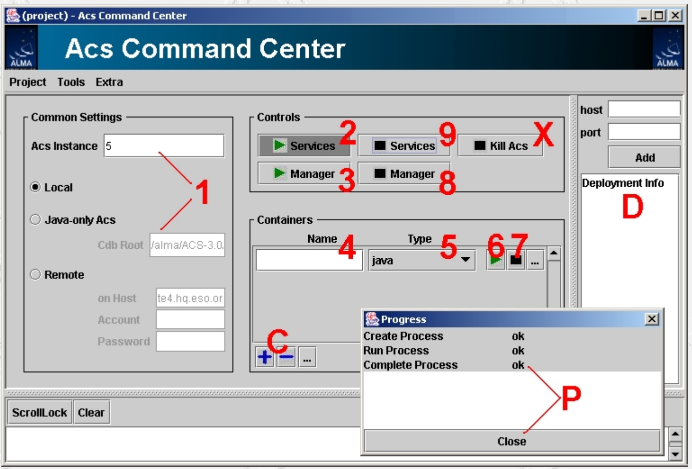
The Administrator Client:

- Displays the information about CORBA objects on the local network. This includes MACI defined objects, such as Containers and Managers as well as Components (or, more in general, Controlled Objects (COBs), since the Manager is able to provide information and partially manage not only Components but also other CORBA servants that do not implement the full MACI interface , like the standard CORBA Services). Both currently active COBs and potentially active COBs (i.e. COBs that the Manager is able to bring online on request) are displayed.
- Interact with the MACI Manager through IDL Administrator interface, by receiving notifications about other clients and activators in the system from the Manager.
- Is a standalone Java program.



ACS Command Center

ALMA Project



Acs Instance 5

Cdb Root /alma/ACS-3.0

on Host te4.hq.eso.or

Type java

```
Create Process      ok
Run Process         ok
Complete Process    ok
```

G.Chiozzi
Feb, 12 2004

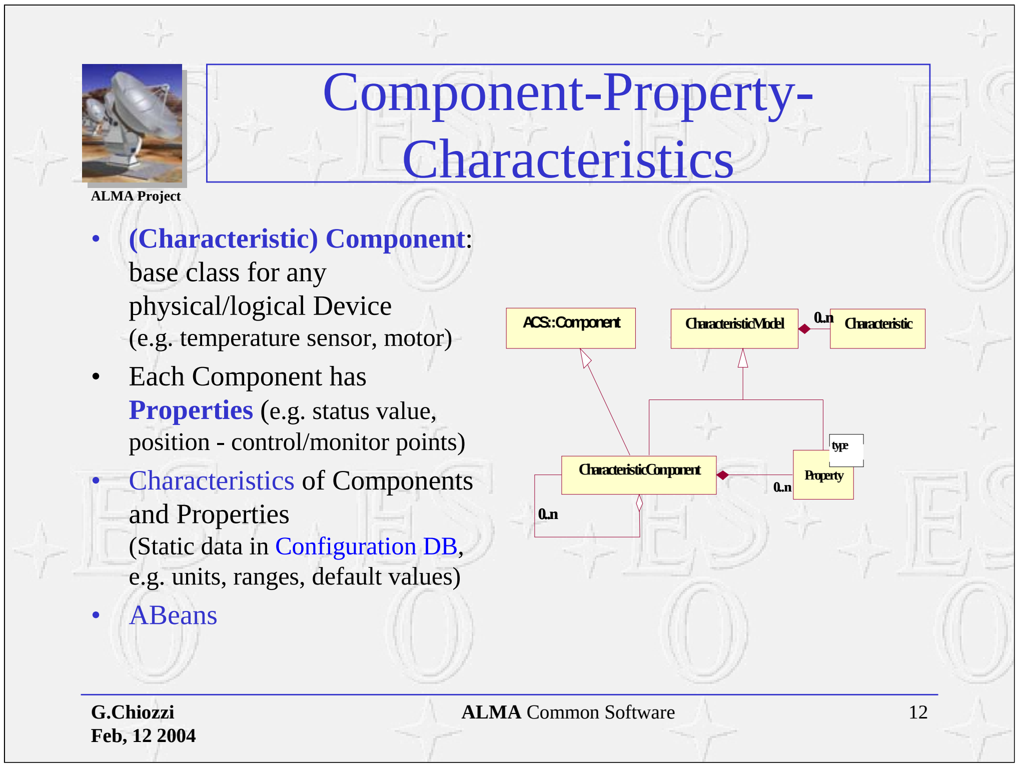
ALMA Common Software

11

The ACS Command Center is a new administrative application used to start and stop ACS services, managers and container.

I allows to manage the system distributed on several hosts, start tools and inspect the deployment of the system.

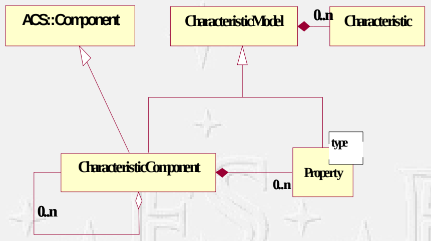
Its functionalities partially overlap the Administrator Client and the two applications are being merged.



ALMA Project

Component-Property-Characteristics

- **(Characteristic) Component**: base class for any physical/logical Device (e.g. temperature sensor, motor)
- Each Component has **Properties** (e.g. status value, position - control/monitor points)
- Characteristics of Components and Properties (Static data in Configuration DB, e.g. units, ranges, default values)
- ABeans



G.Chiozzi
Feb, 12 2004

ALMA Common Software

12

The heart of BACI is a distributed object model.

All common telescope components such as antenna mount, antenna control unit, correlator, etc. are Components defined by means of *Characteristic Components*. Characteristic Components are implemented as CORBA objects that are remotely accessible from any computer through the client-server paradigm.

Each Characteristic Component is further composed of *Properties.* A Characteristic Component can also contain references to other Characteristic Component *s* to build hierarchical structures of components.

Both Characteristic Component*s* and *Properties* have specific *Characteristics*, e.g. a Property has a minimum, a maximum, units…. The common behavior of Characteristic Component and *Property* has been factorized in the *Characteristic Model* common base class. Notice that programmatically characteristics are read-only. For example, all constants related to a *Property* such as min/max, and description are obtained by clients directly from the *Property* by means of remote methods - no direct access to the configuration database is necessary.

Values of the *Properties* are updated asynchronously by means of monitor objects. While there are in principle an infinite number of Characteristic Component types, for example one for each physical controlled device, there are very few different *Property* types

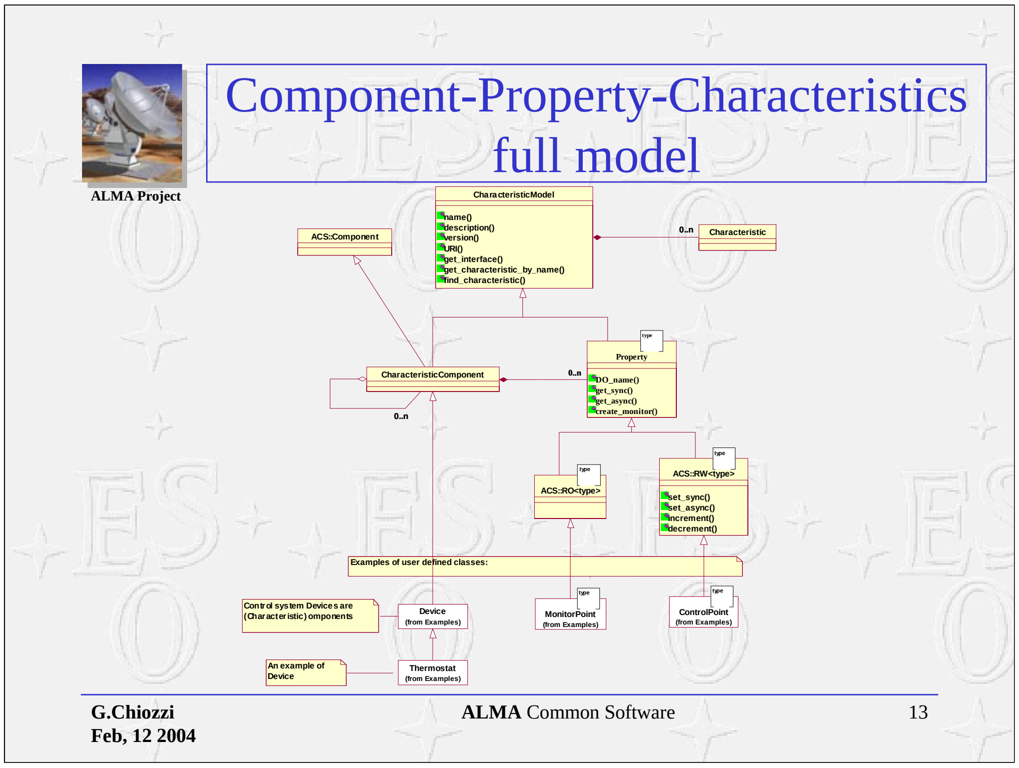
ACS provides a basic Component implementation (called BACI, standing for Basic Access Control Interface) based on the

Component - Property - Characteristic

paradigm, well established in the world of accelerator control systems.

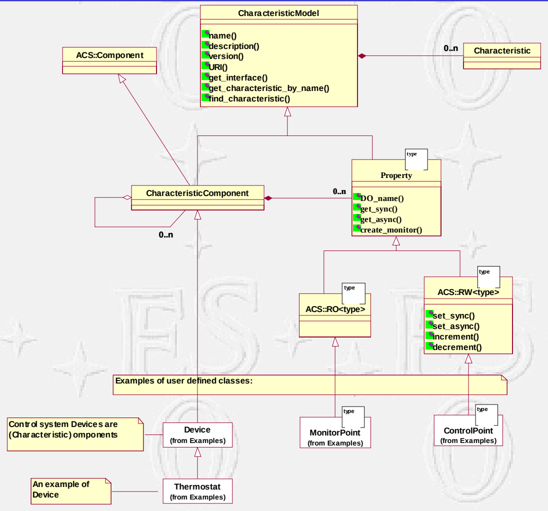
All common telescope components such as antenna mount, antenna control unit, correlator, etc. are CharacteristicComponents.

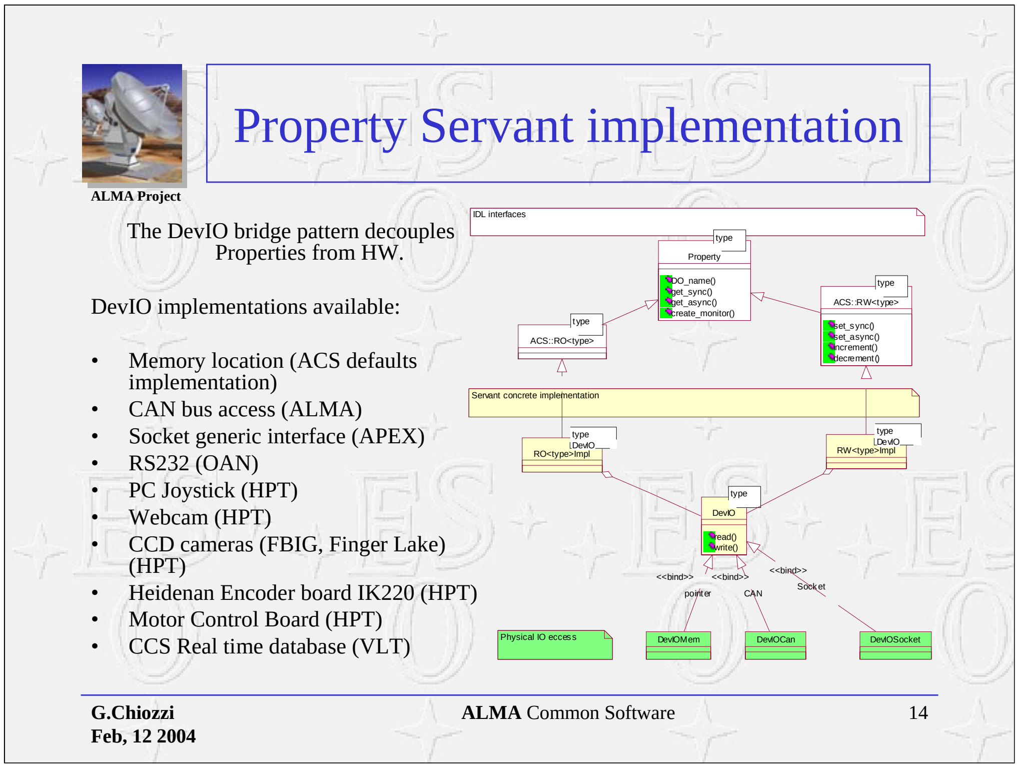
Design patterns for synchronous and asynchronous value retrieval/setting, monitoring and archiving or alarms are part of the Property definition



ALMA Project

Component-Property-Characteristics full model





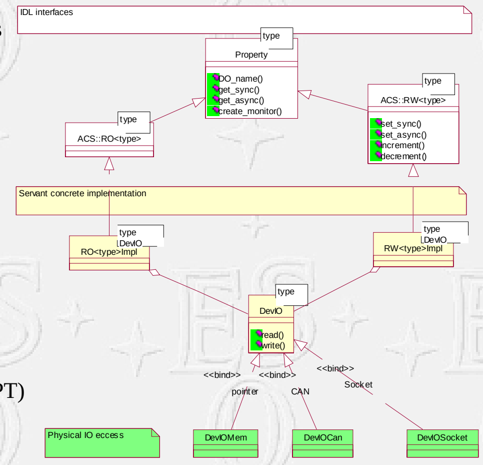
Property Servant implementation

ALMA Project

The DevIO bridge pattern decouples Properties from HW.

DevIO implementations available:

- Memory location (ACS defaults implementation)
- CAN bus access (ALMA)
- Socket generic interface (APEX)
- RS232 (OAN)
- PC Joystick (HPT)
- Webcam (HPT)
- CCD cameras (FBIG, Finger Lake) (HPT)
- Heidenan Encoder board IK220 (HPT)
- Motor Control Board (HPT)
- CCS Real time database (VLT)



G.Chiozzi
Feb, 12 2004

ALMA Common Software

14

This slide show the decoupling between the high level concept of Property and the access to the actual hardware.

While the implementation of Properties is completely general, access to hardware is delegated to a simple DevIO class according to the Bridge design pattern.

The DevIO class needs to implement read and write functions to access the hardware.

When a property is instantiated, it receives a proper DevIO implentation that enable it to retrieve and store (if writable) values in the hardware.

There are already many DevIO implementation available, some developed for ALMA and some developed from other projects:

•Memory location (ACS defaults implementation)

•CAN bus access (ALMA)

•Socket generic interface (APEX)

•RS232 (OAN)

•PC Joystick (HPT)

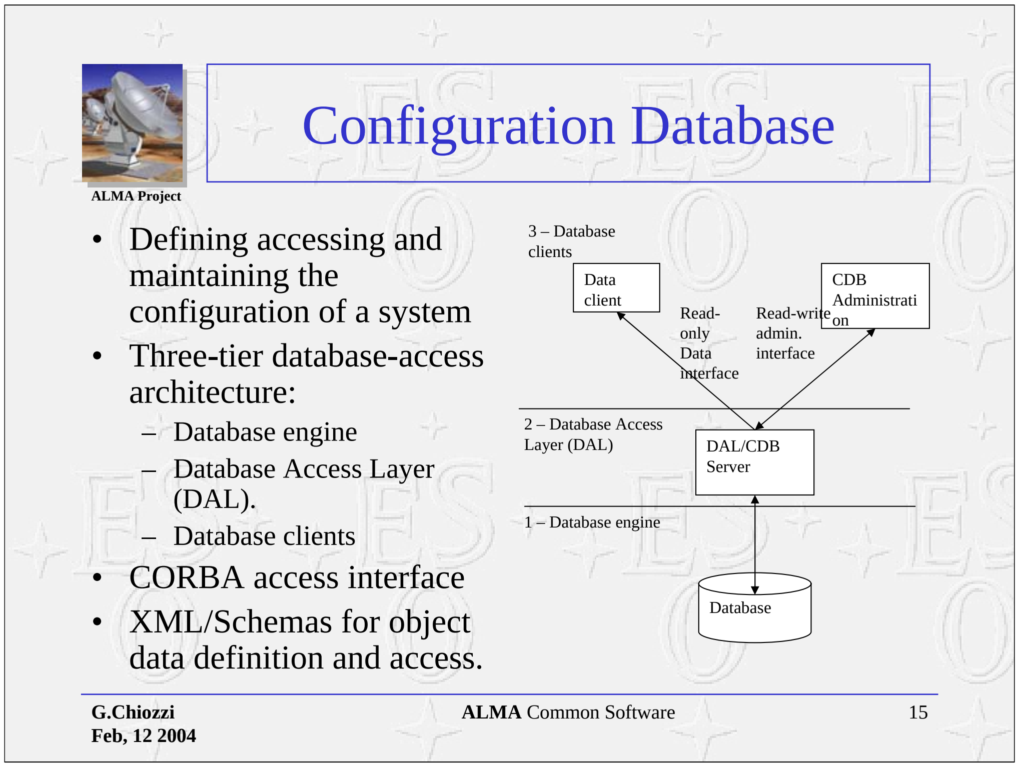
•Webcam (HPT)

•CCD cameras (FBIG, Finger Lake) (HPT)

•Heidenan Encoder board IK220 (HPT)

•Motor Control Board (HPT)

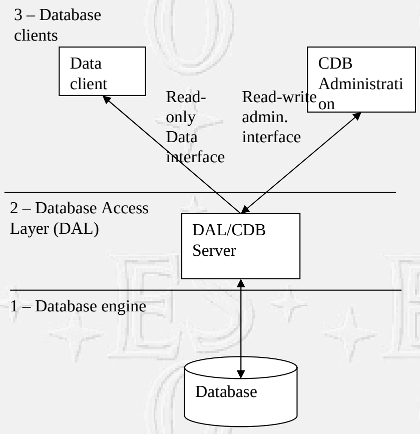
•CCS Real time database (VLT)



Configuration Database

ALMA Project

- Defining accessing and maintaining the configuration of a system
- Three-tier database-access architecture:
  - Database engine
  - Database Access Layer (DAL).
  - Database clients
- CORBA access interface
- XML/Schemas for object data definition and access.



G.Chiozzi
Feb, 12 2004

ALMA Common Software

15

The ACS Configuration Database addresses the problems related to defining, accessing and maintaining the configuration of a system based on ACS.

There are 4 different issues related to this problem:

1. input of data by the user
System configurators define the structure of the system and enter the configuration data.

2. storage of the data
The configuration data is kept in a database.

3. maintenance and management of the data (e.g. versioning)
Configuration data changes because the system structure and/or the implementation of the system’s components changes with time and has to be maintained under configuration control.

4. loading data into the ACS Components
At run-time, the data has to be retrieved and used to initialize and configure the Characteristic Components.

The architecture of the ACS configuration database is based on three layers:

1. The Database Itself
2. The Database Access Layer (DAL)
3. The Database Clients access data from the database using only the interfaces provided by the DAL.

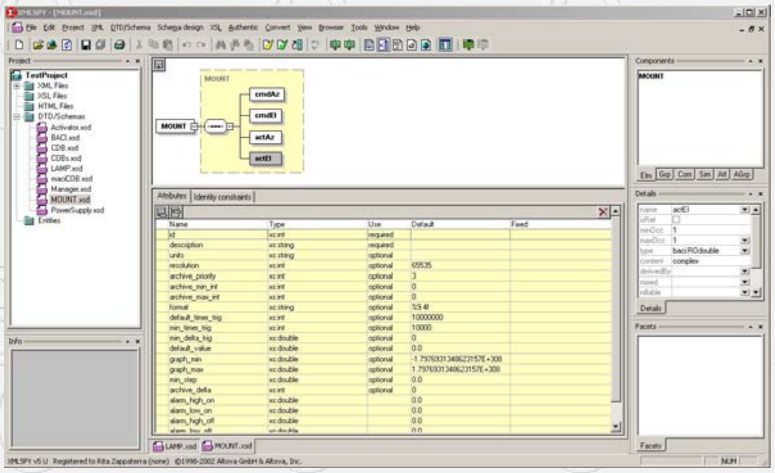
There are two separate interfaces to the CDB:

•Read-Only Data interface, used by all clients that need to retrieve their own configuration. Is very simple and can be very easily implemented using minimal resources.

•Read/Write administrator interface, used to modify and administer the Configuration Database. Few high level applications need to access this interface.

Configuration Database: DO Schemas

ALMA Project



Name	Type	Use	Default	Fixed
id	xs:int	required		
description	xs:string	required		
units	xs:string	optional		
resolution	xs:int	optional	65535	
archive_priority	xs:int	optional	3	
archive_min_int	xs:int	optional	0	
archive_max_int	xs:int	optional	0	
format	xs:string	optional	%9.4f	
default_timer_trig	xs:int	optional	10000000	
min_timer_trig	xs:int	optional	10000	
min_delta_trig	xs:double	optional	0	
default_value	xs:double	optional	0.0	
graph_min	xs:double	optional	-1.7976931348623157E+308	
graph_max	xs:double	optional	1.7976931348623157E+308	
min_step	xs:double	optional	0.0	
archive_delta	xs:int	optional	0	
alarm_high_on	xs:double		0.0	
alarm_low_on	xs:double		0.0	
alarm_high_off	xs:double		0.0	
alarm_low_off	xs:double		0.0	

name	actEl
isRef	☐
minOcc	1
maxOcc	1
type	baci:RODouble
content	complex
derivedBy	
mixed	
nillable	

The choice of XML for the definition of the Configuration Database allows to use off-the-shelf tools to administer the Configuration Database itself.

Per each Component type in the system, an XML Schema defines its configuration structure.

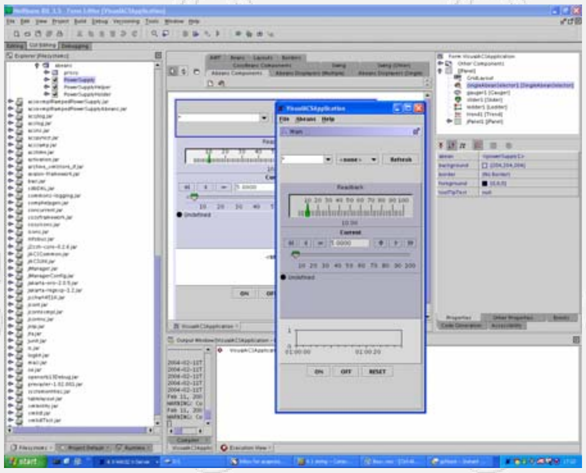
Per each Component instance in the system, an XML file defines the actual configuration, based on the XML Schema.

This allows to define default configuration values in the Schema, “inheriting” from other schemas and override them in the instance XML files.

In this slide we use XMlSpy to edit the XML Schema

ALMA Project

Abeans and visual editing



G.Chiozzi
Feb, 12 2004

ALMA Common Software

17

Abeans are Java Beans that are aware of the MACI and BACI paradigms.

In this way it is possible to use any Java Visual Builder (like Sun Java Netbeans, or upcoming Eclipse extensions) to visually build use interfaces for ACS Components.

A set of graphical Java Beans implements the most useful widgets for the development of Control System applications, aware of the concepts of Compoents, Properties and Characteristics.

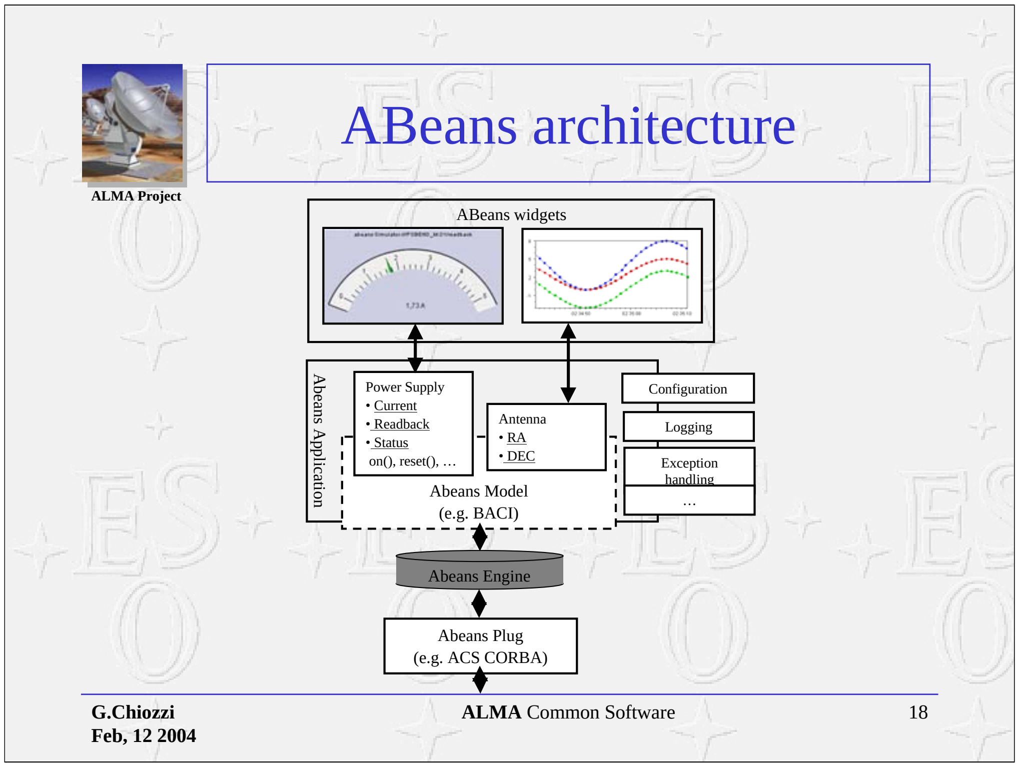
At the same time a code generator produces Java Beans based on the IDL interface of ACS Components. These Beans are therefore automatically integrated in any Visual Builder.

For example, a Gauge widget can be associated to an ACS Property to display the value, draw trend plots and configure automatically itself based on the Characteristics stored in the Configuration Database.

The suggested Java IDE is Eclipse, because it provides extremely good functionality and performance, but unfortunately there is not yet a usable GUI builder available.

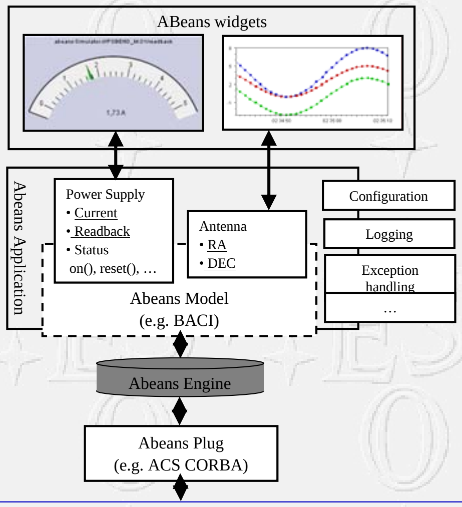
Therefore we currently suggest to use Sun Java Netbeans for GUI layout development. But although the features provided by Netbeans are also extremely good, performance is rather poor.

There are no problems in using at the same time Eclipse for code development and Netbeans ofr GUI layout.



ALMA Project

ABeans architecture



G.Chiozzi
Feb, 12 2004

ALMA Common Software

18

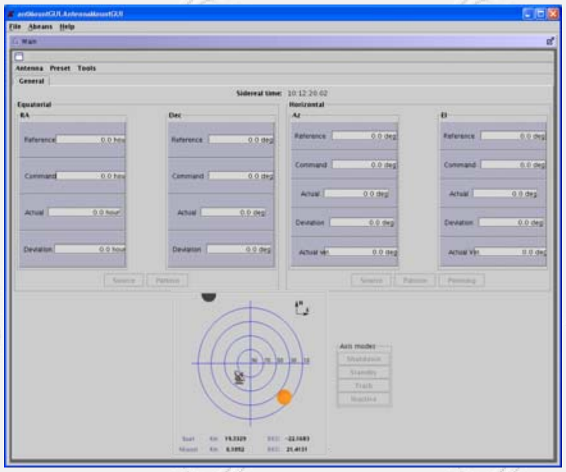
This picture shows the architecture of Abeans.

ABeans “plug-in” architecture allows to use them on top of different Control System architectures (ACS, EPICS, TINE, JADE) and they are therefore used in many different projects independently from ACS.

ABeans have been developed by JSI/Cosylab independently from ACS and are used on different projects.

ALMA Project

ACS sample Mount Control Panel



This picture shows a Mount Control Panel that we are currently developing as a test bed application for TICS.

It is more than 90% built simply by drag and drop from Java NetBeans.

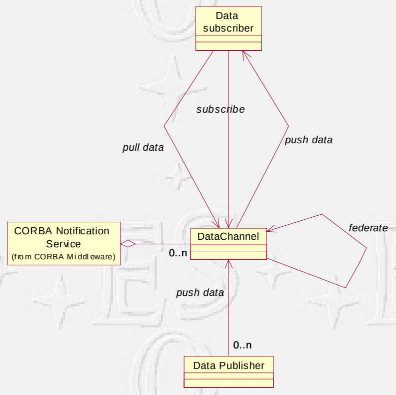
Only a very little amount of Java code needs actually to be “written by hand”.

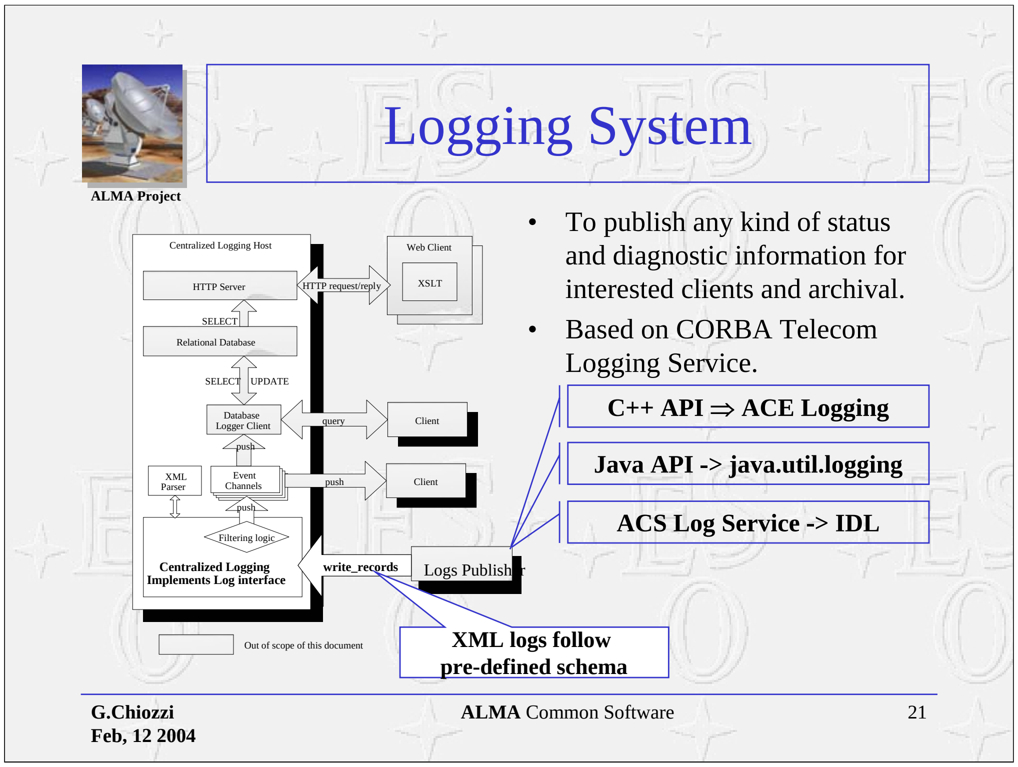
The Java Beans used for the composition work based on the Component-Property-Characteristic pattern and therefore they configure themselves automatically.

ALMA Project

Data Channel

- Implementation of Observer Design Pattern
- Asynchronously pass information between data suppliers and data consumers in a many-to-many relationship
- Based on CORBA Notification Channel
- An ACS API provides simplified client and server API





ALMA Project

Logging System

- To publish any kind of status and diagnostic information for interested clients and archival.
- Based on CORBA Telecom Logging Service.

C++ API ⇒ ACE Logging

Java API -> java.util.logging

ACS Log Service -> IDL

XML logs follow pre-defined schema

G.Chiozzi
Feb, 12 2004

ALMA Common Software

21

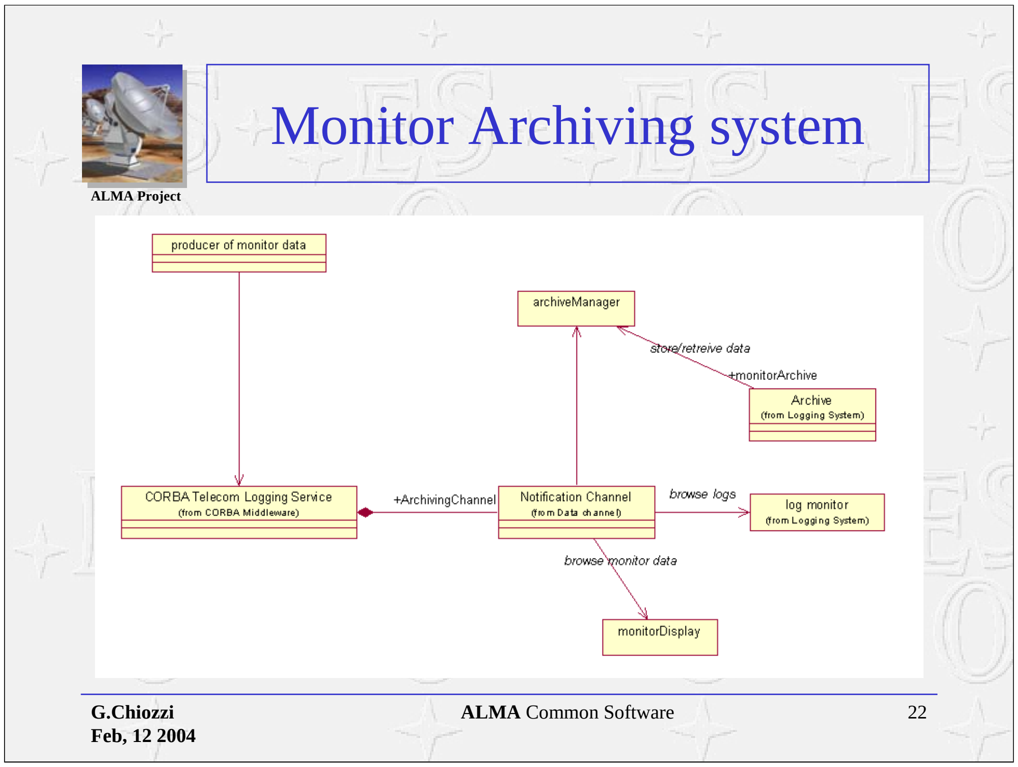
To publish any kind of status and diagnostic information for interested clients and archival.

Based on CORBA Telecom Logging Service.

Logs are published as XML messages.

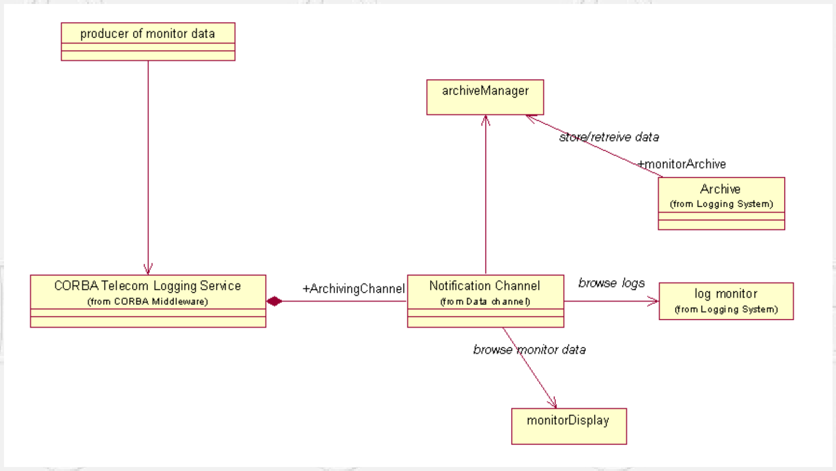
Logging for C++, Java and Python is based on standard native APIs.

An ACS Logging IDL interface provides convenient logging capabilities from any CORBA aware application.



ALMA Project

Monitor Archiving system



G.Chiozzi
Feb, 12 2004

ALMA Common Software

22

The monitor archiving system provides archiving of monitor point values. ACS provides only the data collection mechanism, but not storing of data (this task is left at the Archive Subsystem). Data are made available from the archiving event channel to clients.

The value of each property in the control system can be archived. Archiving is enabled/disabled and configured on per-property basis. ACS Properties publish their value on a specific ArchivingChannel notification channel as structured events, by using the ACS Logging System.

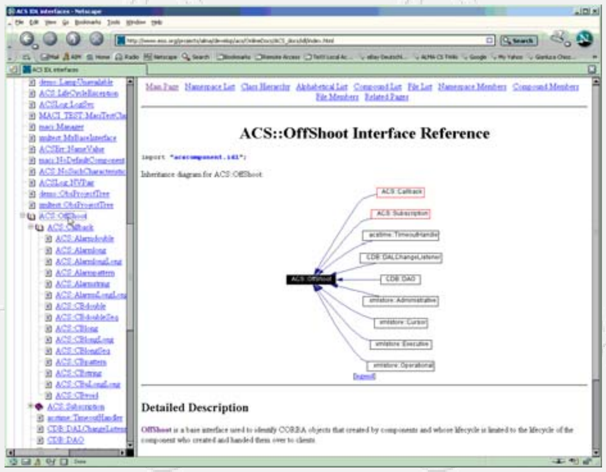
The parameters for data publishing are defined in the following Property's Characteristics:

- **`archive_priority`**
- **`archive_max_int`**
- **`archive_min_int`**
- **`archive_delta`**

Whenever the Characteristics of a Property are such that archive values have to be submitted, a BACI Monitor is created to periodically submit the values to the Logging System.

ACS Documentation:
http://www.eso.org/projects/alma/develop/acs

ALMA Project


Main Page Namespace List Class Hierarchy Alphabetical List Compound List File List Namespace Members Compound Members File Members Related Pages

ACS::OffShoot Interface Reference

`import "acscomponent.idl";`

Inheritance diagram for ACS::OffShoot:

Detailed Description

OffShoot is a base interface used to identify CORBA objects that created by components and whose lifecycle is limited to the lifecycle of the component who created and handed them over to clients

G.Chiozzi
Feb, 12 2004

ALMA Common Software

23

A major effort has been spent in providing good an comprehensive documentation for ACS.

ACS Overview, Installation and Release Notes

These are the first documents to be read when receiving the package.

- The **Overview** provides a general overview of the distribution and in particular of the processes and executables available.
- The **Installation Manuals** guides step by step in the installation and verification of ACS.
- The **Release Notes** describe the current status of ACS and the backward incompatibilities with. It is an essential reading before porting applications from a previous version of ACS.

• **Specification Documents**

The main document in this section is the ACS Architecture, that describes the final architecture ACS aims to. The other documents detail the most important packages defined/implemented with ACS

• **User Manuals and Tutorials**

The **BACI Device Server Tutorial** is an introduction to writing C++ code based on ACS and is a must to read. Then there are a few User Manuals for ACS Logging Service and the tools GUIs (Object Explorer, Jlog, Administrator Client).

• **Abeans (Java ACS Beans) Documents**

Tutorials and manuals on writing user interface applications with Abeans. **Visual Bean Composition Tutorial** is the most useful.

• **ACS IDL Online Documentation and UML Model**

HTML Online reference for the ACS IDL interfaces

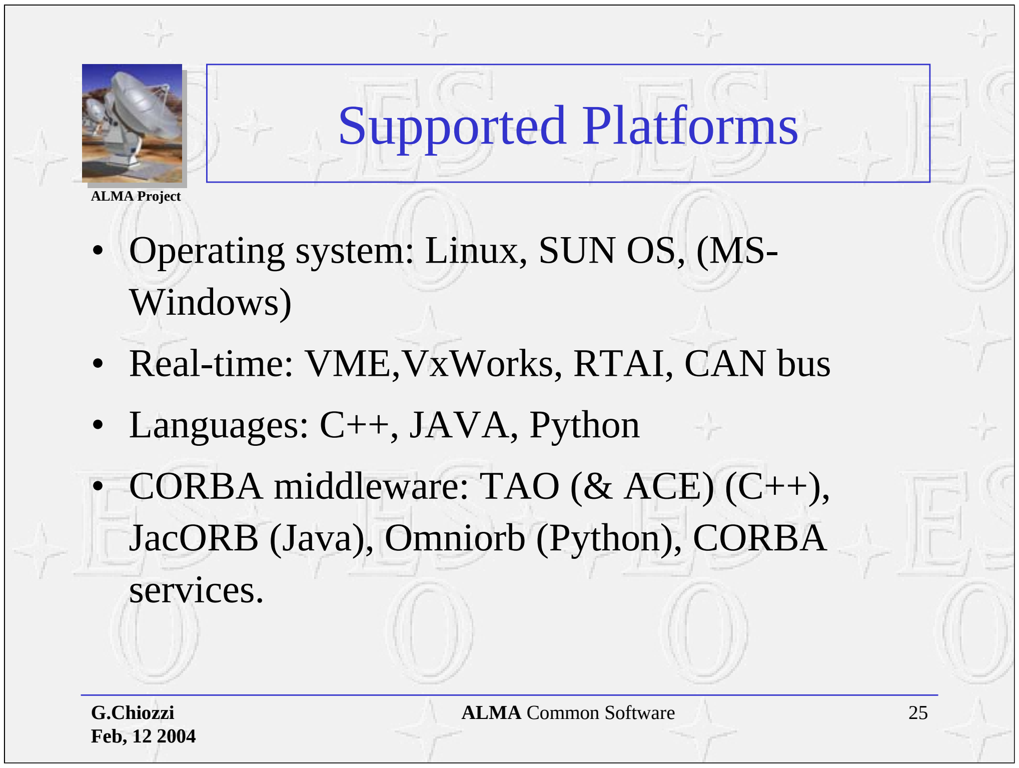
ALMA Project

ACS Documentation:
http://www.eso.org/projects/alma/develop/acs

ALMA Common Software
ACS 3.0 documentation

ACS::OffShoot Interface Reference

Detailed Description



ALMA Project

Supported Platforms

- Operating system: Linux, SUN OS, (MS-Windows)
- Real-time: VME,VxWorks, RTAI, CAN bus
- Languages: C++, JAVA, Python
- CORBA middleware: TAO (& ACE) (C++), JacORB (Java), Omniorb (Python), CORBA services.

G.Chiozzi
Feb, 12 2004

ALMA Common Software

25

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.

For example ACS 3.1 will provide complete support for:

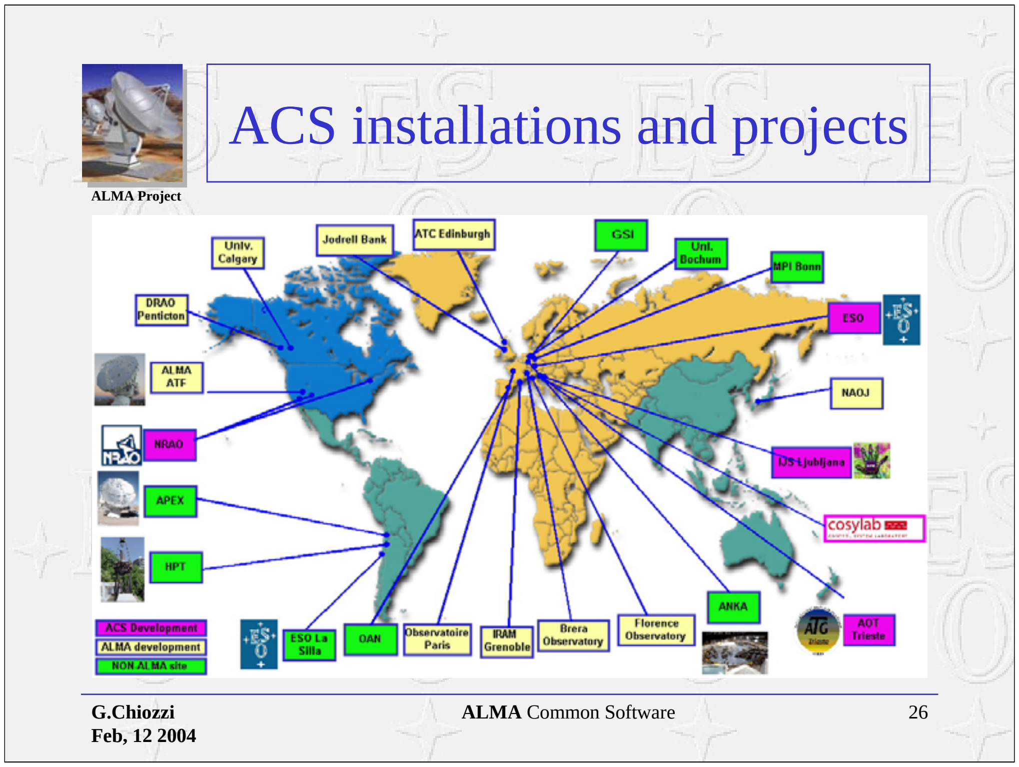
• Linux (RH 9) development and run time

• RTAI support inside RH 9

• Cross development for VxWorks (Tornado 2.5) from SUN workstations (Solaris 2.8), upload on VxWorks run time and debugging

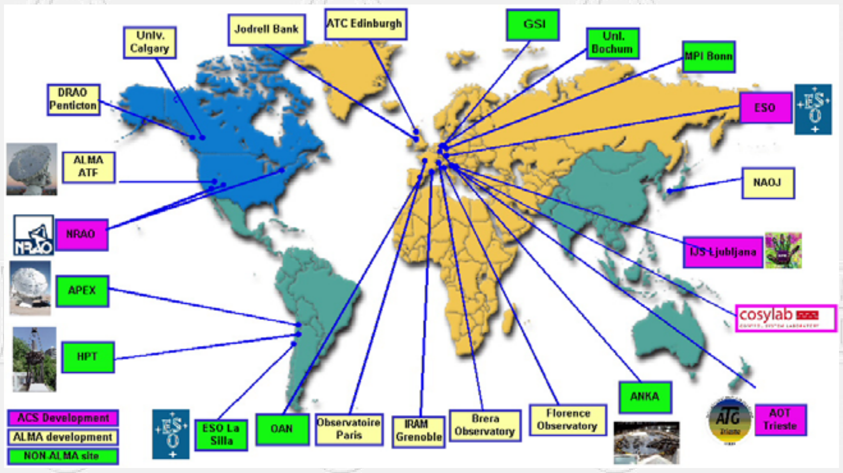
Future releases of ACS will also support small-footprint run time only installations.

An MS-Windows version of most ACS core components is used at JSI for development and testing.



ACS installations and projects

ALMA Project



G.Chiozzi
Feb, 12 2004

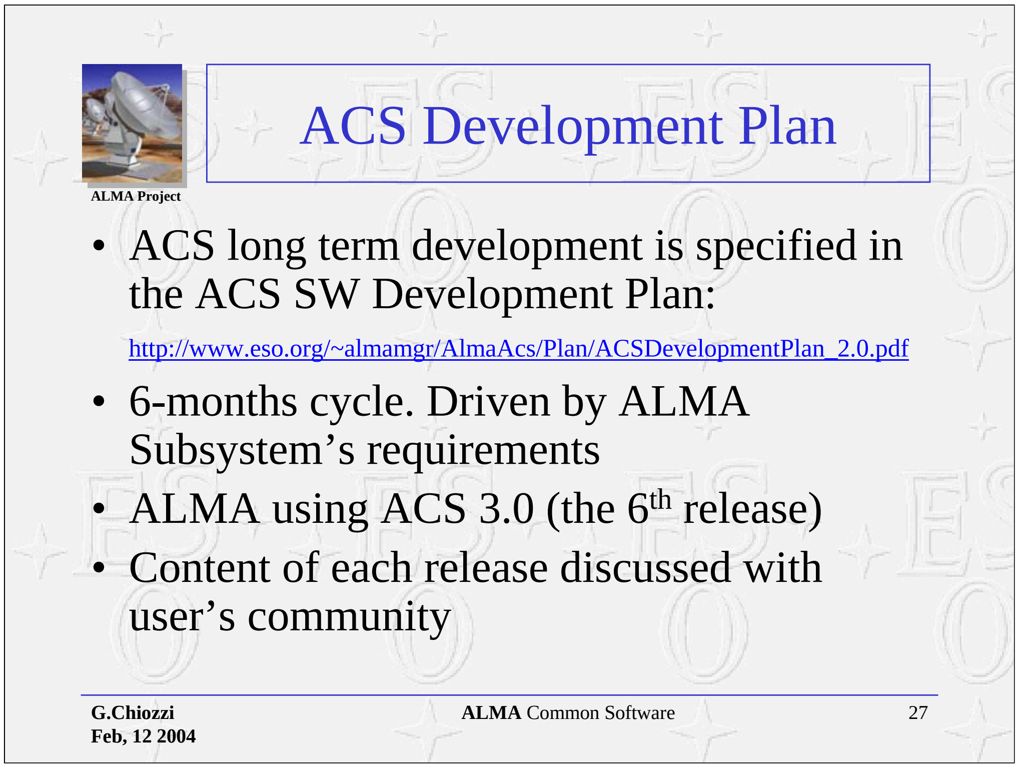
ALMA Common Software

26

ACS is installed in all ALMA development sites, but it is also used or under evaluation by a number of other projects.

On March 8th and 9th there will be a workshop in at ESO in Garching with all current and potentially interested users of ACS.

ABeans have been developed by JSI/Cosylab independently from ACS and are used on different projects, in particular on a number of EPICS projects.



ALMA Project

ACS Development Plan

- ACS long term development is specified in the ACS SW Development Plan:

  http://www.eso.org/~almamgr/AlmaAcs/Plan/ACSDevelopmentPlan_2.0.pdf

- 6-months cycle. Driven by ALMA Subsystem's requirements
- ALMA using ACS 3.0 (the 6th release)
- Content of each release discussed with user's community

G.Chiozzi
Feb, 12 2004

ALMA Common Software

27

The ACS development is based on the ACS SW Development Plan document, that is updated at each ALMA SW Design Review. This document gives the long term direction of development.

The ACS release plan is further based on a 6-months release cycle, synchronized with ALMA SW releases to allow subsystems to develop the code for their releases based on a stable ACS release. We alternate:

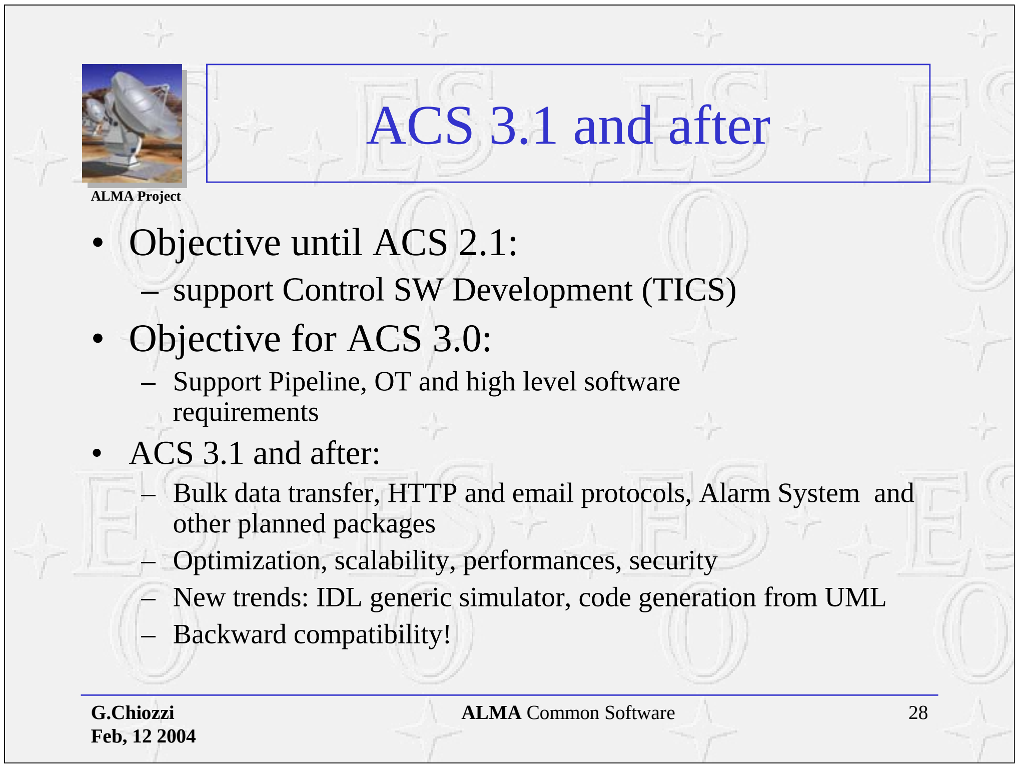
•Major releases, with the main objective of introducing new features

•Minor releases, with the main objective of bug fixing

Between release we have "third number" patch releases, if and when necessary.

The detailed content of each release is discussed with the user's community beforehand, based on a proposal of the ACS team that takes into account both the long term development plan and the requests from the ALMA subsystems (and eventually other external users).

This scheme allows ACS development to follow a clearly defined roadmap, being at the same time very responsive to user's needs.



ALMA Project

ACS 3.1 and after

- Objective until ACS 2.1:
  - support Control SW Development (TICS)
- Objective for ACS 3.0:
  - Support Pipeline, OT and high level software requirements
- ACS 3.1 and after:
  - Bulk data transfer, HTTP and email protocols, Alarm System and other planned packages
  - Optimization, scalability, performances, security
  - New trends: IDL generic simulator, code generation from UML
  - Backward compatibility!

G.Chiozzi
Feb, 12 2004

ALMA Common Software

28

The first releases of ACS concentrated on supporting Control SW development, because of the priority of TICS for the project.

With ACS 3.0 we have moved the focus of the development to satisfying the requirements from higher level subsystems and in particular the Pipeline and the OT. For the first “customer” we have therefore implemented much stronger support for Python scripting and dynamic deployment that are needed for the integration of AIPS++ in ACS. For the OT we have implemented pure Java installation features based on Java Web Start technology, allowing the automatic installation and update of applications on the astronomer’s desk.

The upcoming ACS 3.1 release will provide updates on the operating systems supported, introduce support for RTAI and solve bugs and requests for changes triggered by the usage of 3.0.

On top of that we will introduce prototypes for new developments that will be released as production code in ACS 4.0:

•A new ALMA Alarm System (a prototype is available already now, but is very limited and does not implement the desired architecture).

•A first implementation of the bulk data transfer

•ACS support for HTTP and email communication for higher level subsystems

•IDL Generic Simulator to emulate the interfaces of any Component known to the system

The ACS interfaces have stabilized in the course of the previous releases and from ACS 3.1 we want to put as an objective backward compatibility, to minimize the changes in code required for the porting to a new release.

We are also introducing new packages and developments on top of the existing framework based on emerging technologies that we confider sufficiently mature to be applied successfully. The ALMA HLA team is for example working, in collaboration with the ACS team, with code generation tools to produce code for the LAMA Data Model from a UML model. We consider this approach extremely effective. Using these same technology applied to Component design, we will be able to simplify to a high degree ACS development and system deployment (the APEX and HPT team are already using successfully a less advanced form of Component code generation).

From ACS 4.0 we will also concentrate on some optimization, scalability and performance issues (like Manager federation) that are not a concern now but will be important for the final deployment of ALMA. CORBA provides anyway strong support for such issue (as well as for security, if needed) and therefore it is more a matter of selecting what to do and how among the various available solutions, rather than really investigating if the problems can be solved.

ALMA Project

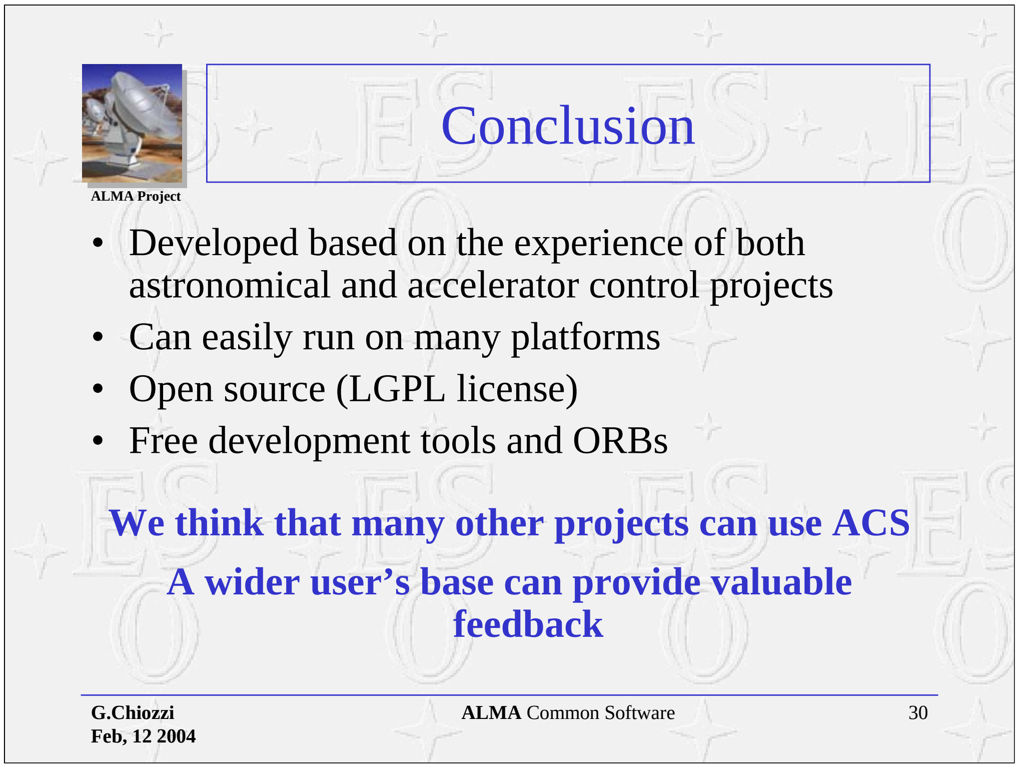
ALMA Sites
Chajnantor

www.eso.org/projects/alma

www.alma.nrao.edu/development/computing

http://www.eso.org/projects/alma/develop/acs

http://kgb.ijs.si/KGB/



ALMA Project

Conclusion

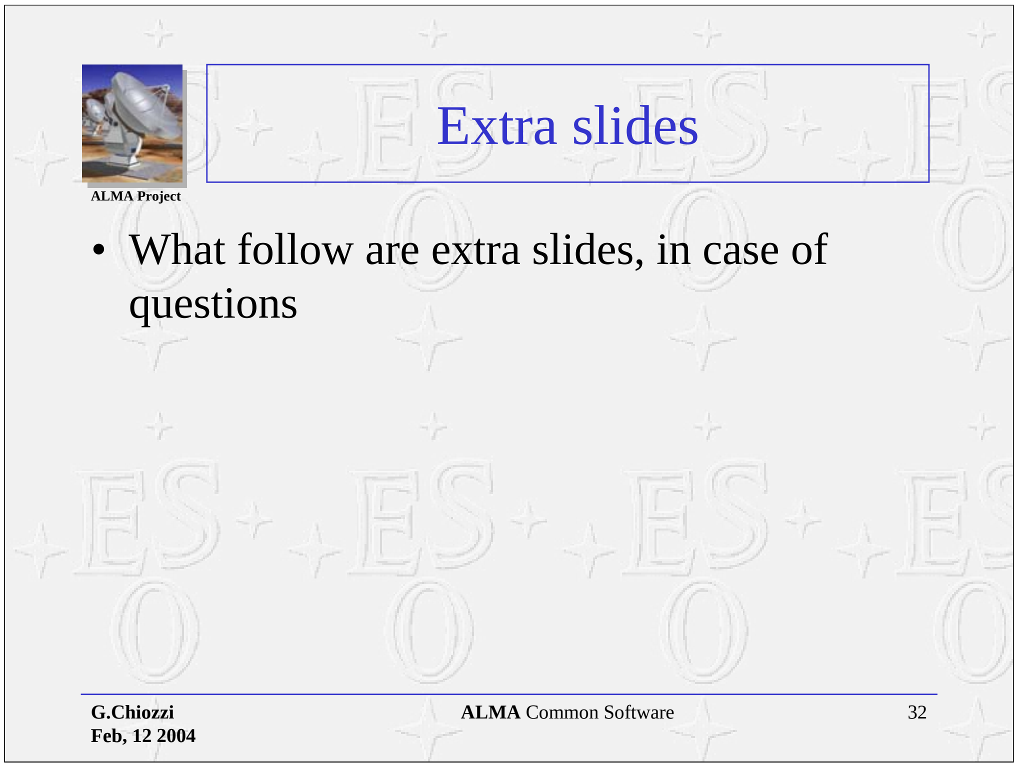
- Developed based on the experience of both astronomical and accelerator control projects
- Can easily run on many platforms
- Open source (LGPL license)
- Free development tools and ORBs

We think that many other projects can use ACS

A wider user's base can provide valuable feedback

ALMA Project

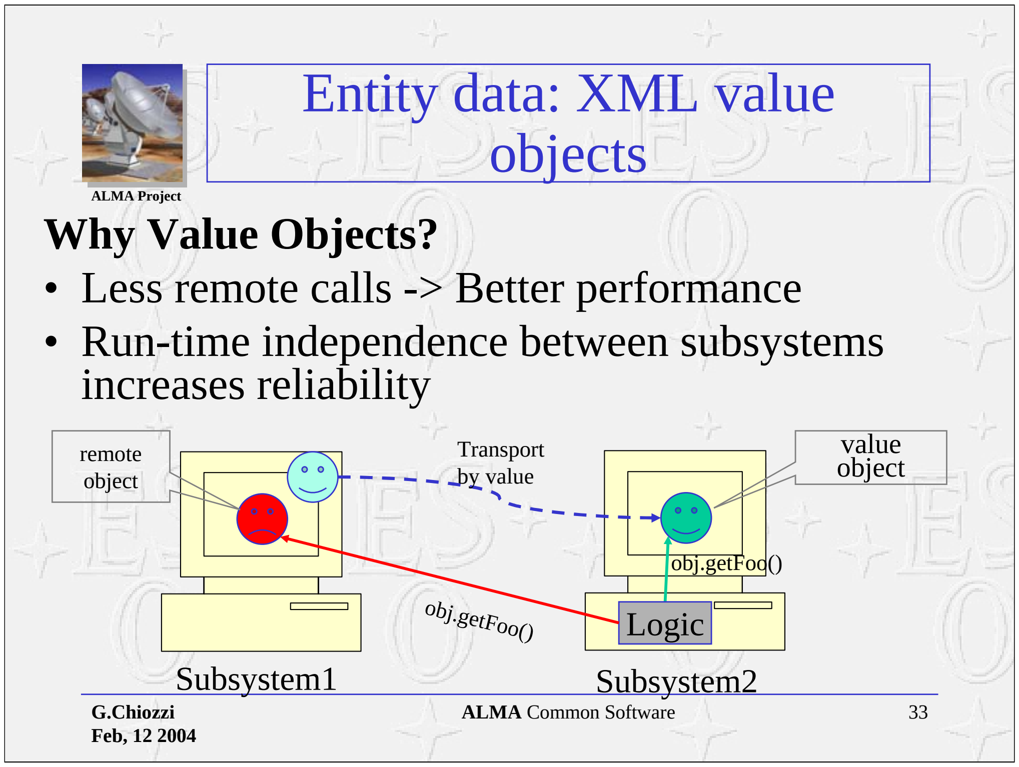
Questions (& Answers)



ALMA Project

Extra slides

- What follow are extra slides, in case of questions

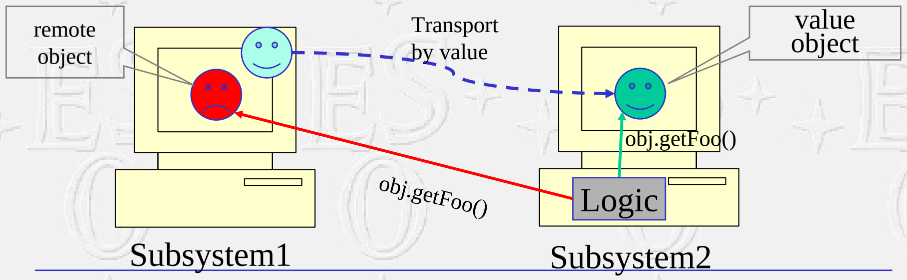


ALMA Project

Entity data: XML value objects

Why Value Objects?

- Less remote calls -> Better performance
- Run-time independence between subsystems increases reliability



G.Chiozzi **ALMA** Common Software 33
Feb, 12 2004

The Acs Architecture requires the abilty to send Entity Data as Value Objects from one subsystem to another or to retrieve Entity Data from the Archive Subsystem and use it locally, until it is time to commit the changes in the archive.

Thie applies, for example, to Persistent Objects, like “User”, “ObservingProject”, “CorrelatorConfig”

XML as the Format for Value Objects

We have chosen to use XML as the format to be used for the serialization of Value Objects.

Using CORBA and different programming languages, the only alternative would have been **CORBA valuetype**.

XML serialization has the following advantages over CORBA valuetype:

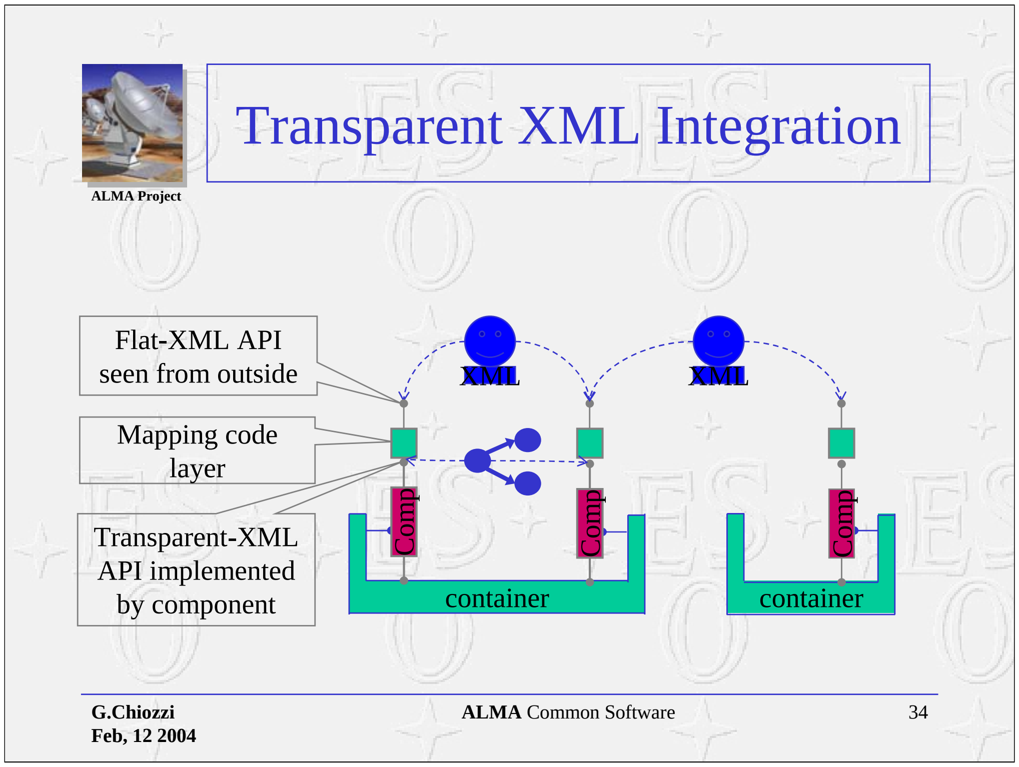
• XML is suitable also for Data Persistence

• XML is usable also on transport protocols different from CORBA, like http or email.

• XML Schema allows stronger typed declarations with respect to IDL and allows to use versatile automatic validation tools

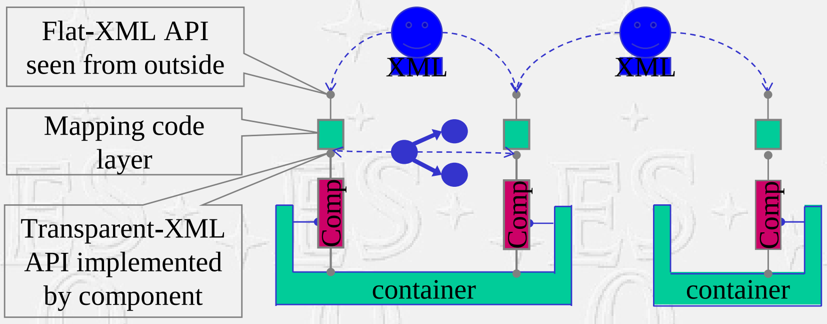
• CORBA valuetype is not supported by many ORBs

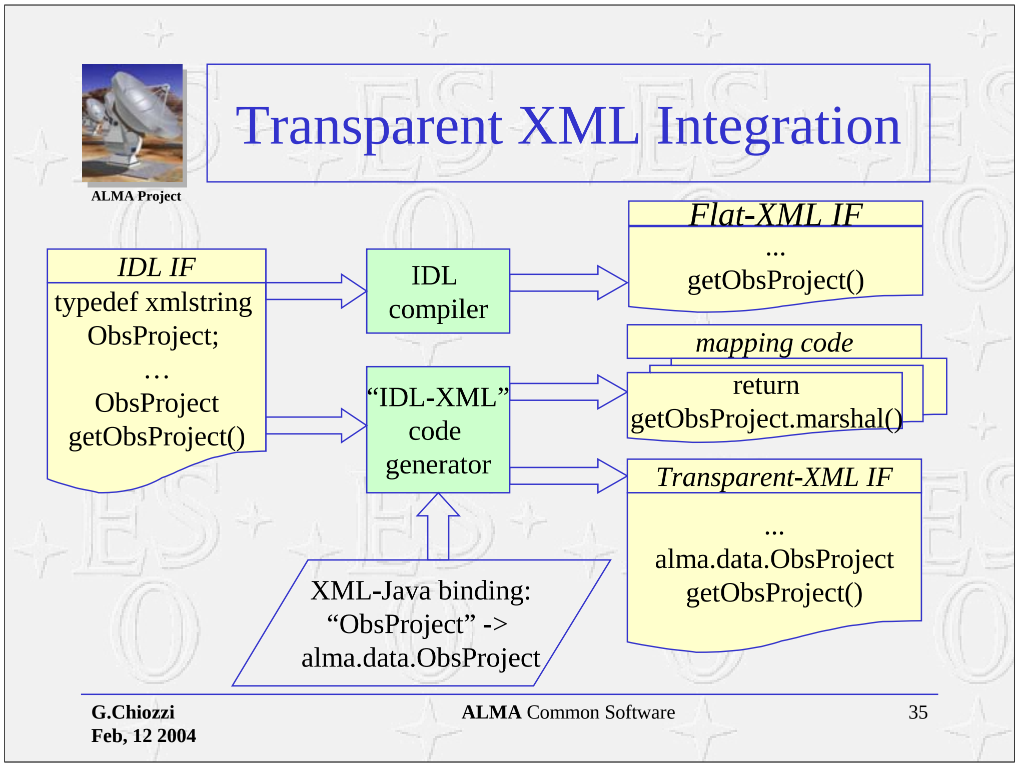
• XML can be easily manipulated “by hand”or using many publicly available tools. This is particularly important for a step-by-step development of the software, where advanced manipulation tools will be developed in later phases of the project.



ALMA Project

Transparent XML Integration


Flat-XML API seen from outside
Mapping code layer
Transparent-XML API implemented by component
XML
XML
Comp
Comp
Comp
container
container



Transparent XML Integration

ALMA Project
IDL IF
typedef xmlstring ObsProject;
…
ObsProject getObsProject()
IDL compiler
"IDL-XML" code generator
Flat-XML IF
…
getObsProject()
mapping code
return getObsProject.marshal()
Transparent-XML IF
…
alma.data.ObsProject getObsProject()
XML-Java binding: "ObsProject" -> alma.data.ObsProject

G.Chiozzi
Feb, 12 2004

ALMA Common Software

35

CORBA IDL signature of a method includes XML Data Entity parameters as a simple structure containing a string with the XML information.

A code generator is used to map this signatures into a new API that contains the corresponding XML mapping classes:

• Native binding classes instead of XML strings

• New structures, interfaces etc. if they contain XML Data Entities

• Automatic (de)serialization