ACS – OVERVIEW OF TECHNICAL FEATURES

K. Žagar, M. Plesko, M. Šekoranja, I. Verštovsek, D. Vitas (JSI and Cosylab)
G. Chiozzi, B. Jeram, H. Sommer, R. Georgieva (ESO)
D. Fugate (NRAO)
R. Cirami, P. Di Marcantonio (INAF-AOT)

Abstract

The ACS is a CORBA-based framework for the development of Control Systems and higher level data flow and coordination applications. It is used to develop the ALMA software and in particular the ALMA Control System. It currently runs in the ALMA Test Interferometer, in the APEX radiotelescope and in the accelerator ANKA in Karlsruhe. More about ACS status and developments can be found in the article by G. Chiozzi [1]. This paper provides an overview and description of ACS features. ACS uses several standard CORBA services such as notification service, naming service, interface repository and implementation repository. ACS hides all details of the underlying mechanisms, which use many complex features of CORBA, queuing, asynchronous communication, thread pooling, life-cycle management, etc. In addition, ACS provides a powerful XML-based configuration database, synchronous and asynchronous communication, configurable monitors and alarms that automatically reconnect after a server crash, run-time name/location resolution, archiving, error system and logging system. Any logical or physical device (e.g. power supply) in a control system is represented by a component. Component contains properties - entities that can be monitored and controlled, and characteristics which contain static data, such as name, units or description. Component management is handled by the ACS Component/Container model. In this simple model, Containers manage the lifecycle of Components with the help of a centralized Manager. The Component/Container model is language and platform independent and the Manager is capable of deploying, manage the lifecycle and locate Components in appropriate Containers written in C++, Java and Python. Containers provide Components with a very simple way to access common centralized services. 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. The Container also supports transparent XML serialization of complex data entities (like a complete Observing Proposal or an Observing Script) through CORBA. This capability is very important to allow a smooth data flow from high level software down to the Control System. ACS comes with all necessary generic GUI applications and tools for management, display of logs and alarms and a generic object explorer, which discovers all CORBA objects, their attributes and commands at runtime and allows the user to invoke any command. C++ is the main language for the development of core Control System DOs and for real time applications. Work is being done to port ACS to Real Time Linux. Coordination application, clients, GUIs and general higher level applications are written in Java and can run on any JVM-enabled platform. The Java-ACS subset is therefore available as a light separate package that can be installed on any platform where Java is available. In C++, ACS uses the free ORB TAO, which is based on the operating system abstraction platform ACE, ACS has been ported to Windows, Linux, Solaris and VxWorks. On the Java side, JacORB is used. Python clients are based on OmniORB. Accompanying this article, a live demo of ACS will be presented at the conference.

INTRODUCTION

From the abstract point of view, control systems have many things in common. Basically, they convey data acquired from devices to the operator and/or control logic, and commands issued by the operator/control logic back to the devices.

Because large experimental physics facilities are inherently complex, several processing units (host computers) are needed to handle all control-related tasks, resulting in a distributed system. In distributed systems, management issues arise: which part of the system is operational and which isn't, what versions of device drivers are deployed, how they are configured, etc.

With the aim to solve these issues we have developed the *Advanced Control System* (ACS, [2]) framework, whose technical features are presented in subsequent sections.

Note that abbreviation **ACS** has two interpretations, depending on the context. In a standalone context, the acronym stands for *Advanced Control System*, and is applicable to any control-system related application. In the context of the Atacama Large Millimeter Array (ALMA) project, ACS stands for *ALMA Common Software*. Here, it contains additional services, build procedures and documentation that are specific to ALMA.

ARCHITECTURE

ACS-based distributed control systems span three *tiers* (see Figure 1):

1. **The presentation tier** which is responsible for representing data to the human user.

2. **The middle tier** where data is collected, analyzed, converted and prepared for consumption by the presentation tier.

3. **The hardware/database access tier** for accessing/manipulating the hardware/database.

The Component/Container Model

At the core of the ACS is the *component/container model.* On every host, a **container** is deployed, which has the following responsibilities:

1. Contains **components**.

2. Management of components' lifecycle (starting and stopping components, loading component's executable code into memory, configuring components, ...).

3. Interception of calls to the components to perform custom marshalling/unmarshalling (e.g., serialization/deserialization of complex objects to/from XML structures).

4. Communication with the **Manager** (heartbeat, registration, deregistration, ...)

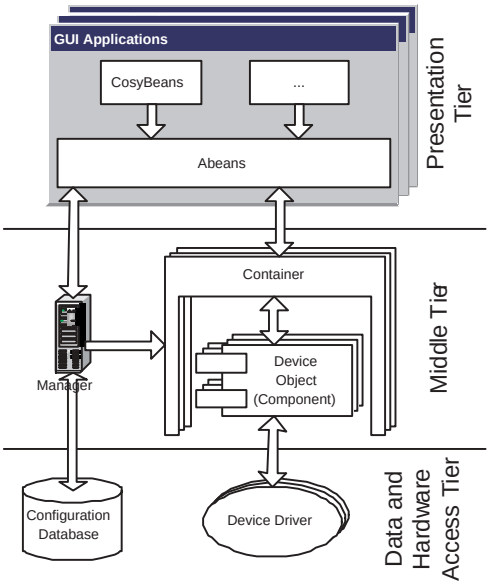

Figure 1: ACS architecture.

Control System Components

In a control system, devices would be represented as components. Such components would expose one or more interfaces through which other components and GUI applications could manipulate the devices. All devices have these things in common:

1. Their **attributes** correspond to input/output channels.

2. Their **operations** correspond to actions the devices are capable of performing.

3. Operations and attributes can be accessed in a **synchronous** or **asynchronous** manner.

Since ACS is object-oriented, these commonalities are defined, once and for all, in the base class of all devices.

The Manager

Manager acts as a broker of components, offering its services to other components, GUI applications, or other subsystems (e.g., automated schedulers, ...). Whenever access to a component is desired, the requestor contacts the Manager. To this end, the Manager maintains the register of all components present in the system.

Not all components are necessarily active at a given point in time (here, the term *active* denotes *loaded in memory, configured, and ready to respond to requests*). If Manager is requested for a component that is not active, it finds a suitable container given the constraints specified in the configuration database, and asks it to instantiate the component. Once done, the Manager returns the reference to this component to the requestor, to whom all the details related to construction have been hidden.

Currently, there can be only one Manager in the system. This is sufficient if the number of requests for components per unit time is sufficiently low, so that the Manager does not yet represent a bottleneck. There are, however, plans to allow for multiple Managers, either as backups in a *hot-standby* mode, or as having responsibility for only a part of the system, similarly to Internet's *Domain Name System* – DNS (*federation*).

Manager's knowledge of containers and their components must not be lost. To achieve this, manager uses an in-memory database of all components. The database is replicated on persistant storage (e.g., hard disk) using transactional techniques.

Manager can be tightly integrated with a CORBA Naming Service. Whenever a component is registered/deregistered with the Manager, the Manager also updates the associated Naming Service accordingly. This way, CORBA applications that are not ACS-aware can still gain access to components.

CORBA Middleware

Common Object Request Broker Architecture (CORBA, [3]) is used to hide the intricacies of network communication (socket binding, remote command invocation, distributed object references, ...). CORBA was chosen because it is a widely adopted standard, and as such offers platform independance and an abundance of existing services and solutions. ACS makes use of the following CORBA services:

- **Event and Notification Services** are used by the ACS's logging and archiving subsystem, through

which events and log entries are dispatched, filtered and consumed.

- **Naming Service** is used in cooperation with the ACS Manager to offer name-to-object resolution to clients that are unaware of ACS.
- Generic applications such as Object Explorer use the **Interface Repository** to determine which operations and attributes are exposed by a given object, so that they can construct calls to them in run-time.

CONFIGURATION DATABASE

Every component in the system must be configured. For components that represent devices, the following data are typically found in the configuration:

- The name of the device.
- List of properties (input and/or output channels).
- Characteristics of properties (name, physical units, channel's hardware address, range, ...).

To make maintenance easier, it is convenient to store the entire configuration in a central place – the **Configuration Database**.

The ACS's Configuration Database (**CDB**) uses XML for data storage and transport over the network. Configuration Database consists of a CORBA service implementing a **Data Access Layer** (**DAL**) interface, and the **Data Access Object** (**DAO**) interface, which makes typed and easy-to-use access to the data possible.

Typically, only one DAL is deployed and registered with the Manager. The Manager consults this DAL on the following occasions:

- When a component is requested that is not yet registered with the Manager, the Manager looks up the component's information in the CDB. The CDB is expected to contain the list of possible hosts (containers) for the component, and the Manager chooses the most appropriate of them for hosting the component.
- When a component is created, the component's container retrieves configuration data from CDB, wraps it in a DAO object and gives it to the component.

With the CDB infrastructure, it will soon be possible to write data to the database, not just access it for reading (the **WDAL**.and **WDAO** interfaces). This functionality is imporant for configuration tools, through which it will be possible to reconfigure the control system without the need to shut it down.

PLATFORMS AND LANGUAGES

ACS is developed and thoroughly tested on Microsoft Windows, Linux, Sun Solaris and WindRiver VxWorks platforms.

When developing with ACS, developers are not limited to a single language: when they are in need of performance or low-level programming, they would use C++ or even C/assembly language. At a higher level, where performance is not such an issue as inherent complexity, Python or Java would be the platform of choice.

ABEANS

For development of applications in Java, it is possible to use ACS services using the Abeans library [4]. Abeans are a framework that provide an object model of the control system to the rest of the application, while hiding the complexities of underlying middleware. This makes application development much easier and more manageable.

CONCLUSION

ACS is a stable and comprehensive framework that allows development of complex, yet maintainable control systems. The investment into building this framework is starting to pay off, thanks to its several deployments around the globe. The increasing number of users provides the development team with valuable feedback, such as reports about unpredictable behavior in previously untested environment, or requests for improvements, making it more bug-free and generaly applicable [1].

In the upcoming years, ACS will address additional requirements. In particular, it will become more scalable and reliable through federation and fault-tolerance capabilities of the management subsystem. Also, it will be ported to embedded platforms to alleviate the need for expensive middle-tier server computers.

REFERENCES

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

[2] M. Plesko, K. Zagar, et al., "ACS – The Advanced Control System", PCaPAC 2002, Frascati, Italy, October 2002

[3] Object Management Group, "Common Object Request Broker Architecture Specification", `http://www.omg.org`

[4] I. Verstovsek et al., "CosyFramework: Application Development Framework for Java", ICALEPCS 2003, Gyeongju, Korea, October 2003