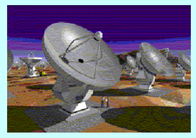


Atacama Large Millimeter Array

A CORBA event system for ALMA common software

D.Fugate (University of Calgary)

SPIE 2004

OVERVIEW

The ALMA Common Software notification channel framework provides developers with an easy to use, high-performance, event-driven system supported across multiple programming languages and operating systems. It sits on top of the CORBA notification service and hides nearly all CORBA from developers. The system is based on a push event channel model where suppliers push events onto the channel and consumers process these asynchronously. This is a many-to-many publishing model whereby multiple suppliers send events to multiple consumers on the same channel. Furthermore, these event suppliers and consumers can be coded in C++, Java, or Python on any platform supported by ACS. There are only two classes developers need to be concerned with: *SimpleSupplier* and *Consumer*. *SimpleSupplier* was designed so that ALMA events (defined as IDL structures) could be published in the simplest manner possible without exposing any CORBA to the developer. Essentially all that needs to be known is the channel's name and the IDL structure being published. The API takes care of everything else. With the *Consumer* class, the developer is responsible for providing the channel's name as well as associating event types with functions that will handle them.

Introduction

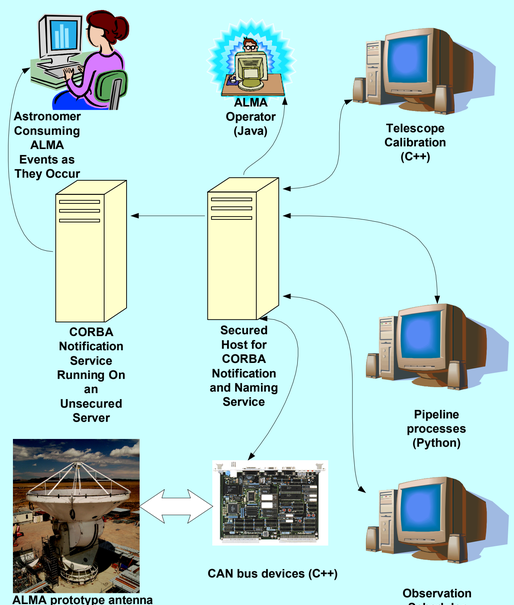
ACS is located in between the ALMA application software and other basic software packages that reside on top of the operating system. In a nutshell, ACS provides a generic interface between ALMA software and hardware, and it also provides basic software services common to all ALMA subsystems. This is accomplished in part by using OMG's CORBA technology which allows code written in one programming language to transparently communicate across networks with code written in entirely different languages.

The generic definition of a software event is an occurrence or happening of significance to a task or program, such as the completion of an asynchronous operation. A channel is defined to be a course or pathway through which information is transmitted so we can naturally assume that an event channel is a pathway where the information consists of events.

For ALMA, there are well-defined data types to be passed around as events. These data types are defined as IDL structs which in turn are nearly identical to C structures. All of these IDL structs include a minimal set of related data and are intended to notify another piece of code that some software state has been reached. The conditions upon which these events occur do not really matter from ACS's standpoint. As far as ACS is concerned, all that needs to be known is the IDL struct to be published along with some sort of identification for the channel the event is to be sent on.

Now that a working definition of an ALMA event and channel has been established, there are two more abstract terms which must be covered – suppliers and consumers. Those familiar with the observer design pattern will recognize these two terms as subjects and observers respectively. At its most basic level, an event channel supplier does exactly what its name implies: supplies events to channels. An event consumer on the other hand consumes these events. Current ALMA software requirements state that suppliers should push events synchronously to consumers which process these events asynchronously using a thread. This is called a push event channel model.

High-level Usage of ALMA Events



Integration of CORBA Notification Channels into ACS

While CORBA provides many ways of dealing with events, not all of them are appropriate for ALMA software. In as such, the following general standards have been adopted:

- Events are defined as user-defined IDL structs
- Channels are identified by constant strings defined in IDL
- The software subsystem supplying events to a channel is responsible for the channel's lifecycle
- Push suppliers are used to supply events
- Push consumers are used consume events
- CORBA structured events contain a publication date and the name of the publisher
- Details of the CORBA Notification Service are hidden from developers

At the most basic level supplier objects are responsible for creating channels if they do not already exist and supplying events to channels. Depending on the programming language a developer wants to use, at least one of three classes is available:

- *Supplier* - a base class which provides a complete implementation of the *PushStructuredSupplier* IDL interface defined by CORBA used to push structured events onto a CORBA notification channel
- *SimpleSupplier* – a subclass of *Supplier* which hides 100% of CORBA from developers and is capable of publishing ALMA events without the need for subclassing
- *RTSupplier* – a subclass of *Supplier* nearly identical to *SimpleSupplier* except that the *publishEvent* method stores the ALMA event in a local queue and returns control immediately. Later, a low-priority thread will wakeup and send the event across the network

The *Consumer* class provides a concrete implementation of the *StructuredPushConsumer* IDL interface needed to consume CORBA structured events from a channel. The general functionality provided in *Consumer* is as follows:

- Connects to only one channel implying events sent from other channels can never be received
- Only receives the type of ALMA events that have been subscribed to
- Has the capability of unsubscribing from an ALMA event type at any time
- Can filter out entire structured events based on the data contained within them
- Has the capability of stopping and resuming receiving all events
- Is notified when a supplier starts or stops publishing an event type
- Includes the implementation of the CORBA method invoked by the channel itself (i.e., *push_structured_event*) each time an event is received. This implementation automatically extracts the ALMA event and passes it to an ACS-defined *Consumer* method (i.e., *processEvent*) which should be overridden by the developer. In the case of *SimpleConsumer*, a *Consumer* subclass, *processEvent* is overridden to invoke a function or method the developer has associated with that particular ALMA event type

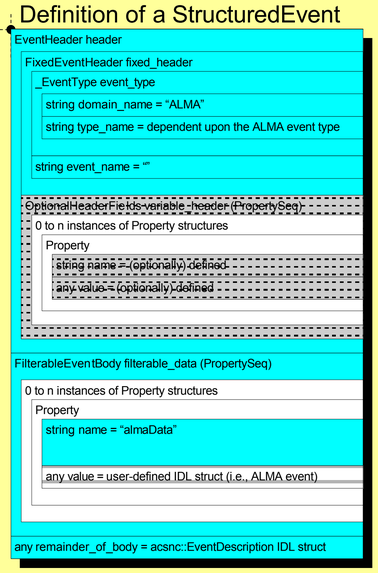
CORBA's Support of Events

Fortunately for ACS, CORBA provides a well-defined specification for the creation, destruction, etc. of event channels. These channels provide support for suppliers and consumers implemented in any programming language or OS platform providing a CORBA implementation. The communication between CORBA suppliers and consumers is completely decoupled, distributed, and is in the form of standard CORBA requests. Furthermore, CORBA also provides a specification for notification channels which are a superset of event channels. A few features these notification channels add are:

- Support for structured events
- Support for channel administrative properties
- Run-time discovery of published event types
- Filtering of structured events

Within ALMA, the primary reason notification channels were chosen over plain event channels was because notification channels support publishing structured events (see figure to right).

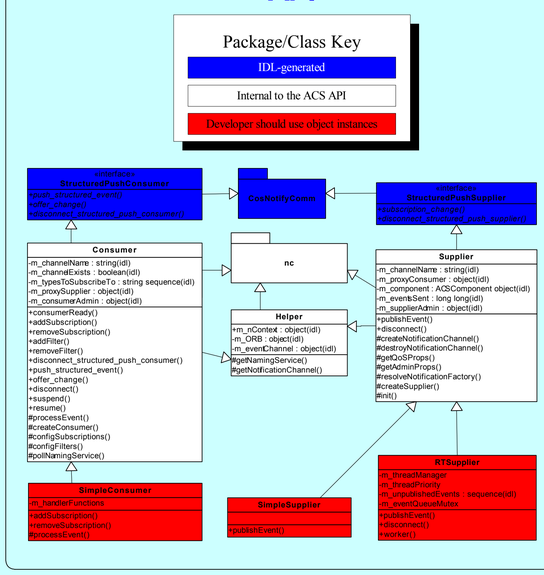
Definition of a StructuredEvent



As you can see in the figure, the CORBA *any* value field (denoted by a background with horizontal lines) is where the ALMA event described previously is stored within the *StructuredEvent* IDL struct. Fields with a cyan background are automatically filled-out by the ACS APIs and fields denoted by a dotted background are not currently being used.

A single CORBA notification channel can support any number of consumer and supplier objects connected to it. That is, if a supplier sends an event to the channel and there are multiple consumer objects subscribed to that particular event type, they will each receive a separate copy of the event. Similarly if multiple suppliers send the same type of event, a single consumer which has subscribed to that event type will receive events from all the suppliers. This is called a many-to-many event channel model and is quintessential to meeting ALMA software requirements.

UML Diagram of the Notification Channel API



Channel Administration and Monitoring

The *ACSEventAdmin* IDL interface along with it's corresponding GUI (see below) was created and implemented in the Python programming language to administer and monitor all ALMA event channels. It supports the following operations:

- Creating new notification channels
- Reconfiguring a channel's properties at run-time
- Destroying notification channels
- Obtaining a list of all active channels
- Obtaining the total number of suppliers and consumers connected to a channel
- Obtaining the total number of events that have been published on a channel
- Obtaining a list of all types of ALMA events that have been published on a channel
- Monitoring all events using the callback design pattern