Astronomical Data Analysis Software and Systems VII
ASP Conference Series, Vol. 145, 1998
R. Albrecht, R. N. Hook and H. A. Bushouse, eds.

Message Bus and Distributed Object Technology

Doug Tody

IRAF Group, NOAO[1], PO Box 26732, Tucson, AZ 85726

Abstract.
In recent years our applications have become increasingly large and monolithic, despite successful efforts to structure software internally at the class library level. A new software architecture is needed to break these monolithic applications into reusable components which can easily be assembled to create new applications. Facilities are needed to allow components from different data systems, which may be very different internally, to be combined to create heterogeneous applications. Recent research in computer science and in the commercial arena has shown us how to solve this problem. The core technologies needed to achieve this flexibility are the *message bus*, *distributed objects*, and *applications frameworks*. We introduce the concepts of the message bus and distributed objects and discuss the work being done at NOAO as part of the Open IRAF initiative to apply this new technology to astronomical software.

1. Overview

In this decade we have seen our software become increasingly large and complex. Although our programs may be well structured internally using hierarchically structured class libraries, the programs have grown large and monolithic, with a high degree of interdependence of the internal modules. The size of the programs and their relatively high level, user oriented interface makes them inflexible and awkward to use to construct new applications. As a result new applications usually have to be constructed at a relatively low level, as compiled programs, an expensive and inflexible approach. The high degree of integration characteristic of programming at the class library level makes it difficult to construct heterogeneous applications that use modules from different systems.

The key technology needed to address this problem, being developed now by academia and commercial consortiums, is known as *distributed objects*. Distributed objects allow major software modules to be represented as objects which can be used either stand-alone or as components of distributed applications. Tying it all together is the *message bus*, which provides flexible services and methods for distributed objects to communicate with one another and share

[1]National Optical Astronomy Observatories, operated by the Association of Universities for Research in Astronomy, Inc. (AURA) under cooperative agreement with the National Science Foundation.

data. Applications are built by linking precompiled components and services together at runtime via the message bus. This paper presents the message bus and distributed object framework being developed by NOAO and collaborators as part of the Open IRAF initiative. This project is funded in part by the NASA ADP and AISR programs.

2. Message Bus Concepts

The message bus is a facility used to bind, at runtime, distributed objects (program components) to construct applications. Components, which are things like data services, display or graphics services, computational services, or program interpreters, execute concurrently and communicate at runtime via the message bus. Components can dynamically connect to or disconnect from the message bus at runtime. The usage of the term "bus" is analogous to the hardware bus in a computer: components are like cards that plug into the bus, and the application program executes in an interpreter component analogous to the CPU. In effect the message bus framework is a virtual machine, built from highly modular, interchangeable components based on an open architecture, with applications being the software available for this virtual machine.

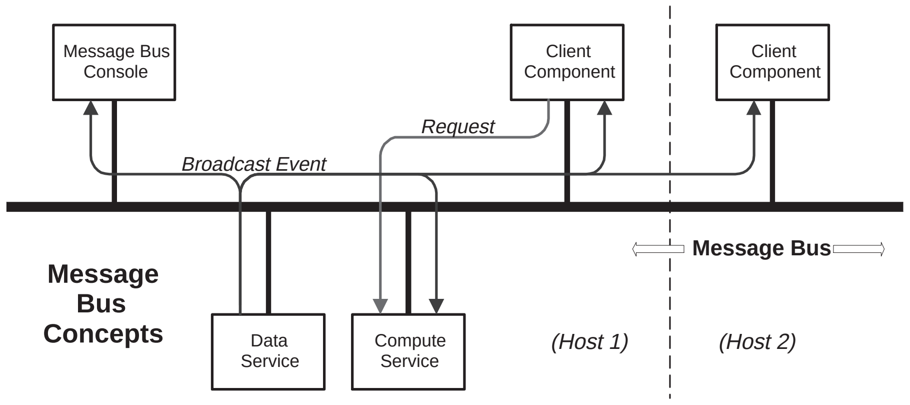

Figure 1. Message Bus Concepts

The message bus is also a means of structuring software, in that it defines a system-level software architecture. Classical structuring techniques are used to construct the individual components, which can be very different from one another internally, for example written in different languages. Applications are composed at a high level, relying upon components for most of their functionality. Components can be large, substantial programs in their own right, capable of independent execution, although from the point of view of the message bus architecture they are merely interchangeable software components which share a standard interface to the message bus. Other components will be custom built compute modules (e.g., implementing science algorithms) developed for the application. *Applications frameworks* are integrated suites of components providing a complete environment for building some class of applications.

3. Message Bus Capabilities

In addition to providing a range of messaging capabilities, the message bus provides facilities for service registration and lookup, autostart of services, and distributed execution. Message bus clients (e.g., components or services) can execute on any host computer connected to the message bus, allowing distributed applications to be easily constructed. The message bus can be either started when an application is run, or "booted" stand-alone in which case a subsequent client will automatically connect to and use the existing message bus. A console client is used to examine and control the state of the message bus and monitor system activity.

As illustrated in Figure 1, most messages fall into one of two primary classes. *Requests* are messages sent to a particular component or object to request that it perform some action (i.e., to invoke a method). Requests may be either synchronous or asynchronous. A synchronous request is similar to a remote procedure call. *Events* are messages which are broadcast by a "producer" client, and received by zero or more "consumer" clients. Clients subscribe to the classes of events they which to receive. The message bus keeps track of the event subscription list for each client and uses it to construct a distribution list when broadcasting an event. In general a producer does not know what other clients, if any, may be consuming the events it generates. Events are inherently asynchronous.

At the simplest level the message bus is responsible only for classifying messages by type and handling message distribution and delivery. The actual content of a message, that is the data contained in a message, is determined by the *messaging protocol* used by the clients. Multiple messaging protocols are possible on the same bus although not necessarily recommended.

The message bus is responsible for the reliable delivery of messages and ensures that messages are received in the order in which they were sent. There is no fixed limit on the size of a message, although other data transport mechanisms such as shared memory may be useful if bulk data is to be transmitted, particularly if it is to be shared by multiple clients. The bus will queue messages for delivery if needed. Point-to-point links are possible in cases where a high data rate is required and only two clients are involved.

4. Existing Messaging Implementations

The first thing we did when we started this project was research on all the existing implementations we could find of anything resembling a message bus. Approximately two dozen were examined, including CORBA, OLE/COM, Tooltalk, some preliminary Java-based implementations, PVM, MPI, ACE, KoalaTalk, EPICS, IMP, Glish, XPA, and others. We were surprised to find that despite all the work in this area, no adequate solution existed. CORBA probably comes closest: it is comprehensive and is based on some very good computer science, but it is expensive, proprietary, bloated, implementations differ, and it is only available on a subset of the platforms we use. CORBA might be a good choice for a local system but it is problematic for use in a product which will be widely distributed and freely available. The chief competition is OLE/COM from Microsoft, which is only available for Windows. Of the freeware implementations we found PVM (Parallel Virtual Machine) to be the most interesting. Although it was developed by the physics community for parallel computation and it is not really a message bus, it is compact, portable, and efficient; the basic facilities provided are good and provide much of the low level functionality needed for a message bus.

5. Open IRAF Message Bus Research

The NASA ADP-funded Open IRAF initiative (http://iraf.noao.edu/projects/) will eventually replace IRAF by a more modern, open system. This will be composed of products which will be usable stand-alone or as modules in other systems. The existing IRAF applications will be migrated to this new framework. The message bus architecture discussed here is being used to develop Open IRAF. In keeping with the Open IRAF approach, the message bus software will be a separate product usable independently of the future IRAF system.

Since no suitable message bus implementation is currently available and the technology is likely to continue to evolve rapidly in any case, our approach has been to develop a message bus API which can be layered upon existing standard low level messaging products. A prototype based on PVM has been developed and is in use now within the Mosaic Data Handling System at NOAO. We are also developing a *distributed shared object* (DSO) facility which will allow data objects such as images to be simultaneously accessed by multiple clients via the message bus. Distributed shared memory techniques are used to provide efficient access to bulk data. Messaging is used to provide the access interface used by clients, to inform clients of any changes to the data object, and to ensure data integrity.

6. Conclusions

The message bus, distributed shared object technology, and applications frameworks based on the message bus provide a powerful way to structure large applications and data systems to control complexity. Applications can be developed at a high level, relying upon sophisticated, well tested components for much of their functionality.

Systems based on the message bus architecture are inherently modular and open, allowing very different sorts of components to be intermixed. Since components can be large and complex products, with few restrictions on how they are implemented internally, it becomes easier for a large number of people to make significant contributions to a data system, reducing the dependence on key personnel.

Frameworks and applications suites layered on a common message bus have the potential to allow different data analysis packages to share the same low level infrastructure. Systems based on the message bus and distributed objects have the potential to combine the resources of disparate data systems groups as well as individual developers in the astronomical community.