Components in the GNOME Project

Miguel de Icaza

miguel@gnu.org

Jonathan Blandford

jrb@redhat.com

In most modern desktops, there is a mechanism to facilitate modular programming through reuse of Components and objects. The current GNOME desktop has much of the infrastructure necessary to provide this style of programming. The future direction of development in GNOME will focus on providing a large set of flexible components for rapid development.

Motivation

The success of UNIX systems in part from allowing users to create new and more complex results by joining smaller components. These components are joined using UNIX pipes to create large numbers of powerful applications.

Although pipes and filters are still invaluable to the poweruser, they have their limitations in today's more complex desktop environment. For example:

- Pipes and filters do not scale well with more complex applications.
- The information flow is unidirectional.
- There is limited support for complex information representations. Characters, lines, and entire files are the basic unit of information exchange. In addition, there is no standard mechanism to check the type on the data that is being transferred.

One way to avoid these problems is to implement a component based programming framework. Unfortunately, UNIX has traditionally lacked a standardized architecture for such a framework. GNOME addresses this lack by providing such a component model, built heavily upon CORBA.

CORBA is the Object Management Group's (OMG) object broker. It allows applications to communicate to each other independent of what platform they are on or who wrote them. More information on CORBA is available at the OMG's homepage (http://www.omg.com/). The implementation of CORBA used in GNOME is ORBit. More information on ORBit can be found at ORBit's homepage (http://www.labs.redhat.com/orbit/).

The Uses of CORBA

CORBA is used in the GNOME system in a number of places:

Exporting an Application's API

CORBA is used in GNOME to export the internal engine of an application to the rest of the system. Any CORBA client (GNOME compliant applications, regular Unix applications, remote clients running on a different operating system) can use the services that these application provides.

Currently, gmc, guppi, and gnumeric support this feature and many other applications are following this tendency. For example, a CORBA aware application can manipulate a spreadsheet in gnumeric. All of the Gnumeric spreadsheet's features are accessible trough the CORBA interfaces defined on the GNOME::Gnumeric module.

The exporting of the interface is essential in building a true component system.

In addition to exporting the API, GNOME is generating a list of standard interfaces that other applications can be written to. For example, the interface Desktop::Editor defines an interface to an editor. This means that an application (such as a Mail client or an IDE program) that is developed using these interfaces can decouple itself from a specific implemenation of the interface.

The user can select its favorite implementation of the Desktop::Editor interface, and this tool will integrate with the rest of the system.

General IPC and RPC Mechanism

Another common task among developers is to have separate processes communicate at some level: either to split functionality, or to separate the processing of some task, or to maximize parallelism. CORBA provides a standard, and easy to use Remote Procedure Call (RPC) system that allows different applications to communicate with each other.

Oftentimes in the UNIX world, little IPC/RPC protocols are hand-crafted individually to communicate between two or more applications. These implementations have strong drawbacks: These protocols have to be debugged every time that they are implemented; They are difficult to maintain: as time passes, people tend to extend these protocols to address new functionality. These protocols tend to get bloated and they become difficult to maintain, to document and to standarize. We do say that these hand-crafted protocols do not scale.

Keeping code simple and maintainable is one of the major goals of the GNOME project: the code we are writing needs to be ready to be modified, enhanced and extended for a long period of time. It is more important to provide an easy to maintain framework to us than anything else.

Additionally, if another developer needs to use these hand-crafted protocols, their only option is to figure out what the protocol is by reading the source code or hoping that the protocol is documented (and many times they are not).

GNOME has tried to minimize these issues by using CORBA as its communication mechanism. CORBA was designed to tackle these problems on a larger scale, making it more than powerful enough to handle GNOME's needs. Once an application designer has learned how to use CORBA in GNOME, they can use it everywhere. In addition, GNOME will handle the actual activation of the call, so the application does not need to worry about it.

Currently, both the GNOME Control Center and the GNOME Panel use CORBA heavily to communicate with their embedded applications. Additionally, a few games use CORBA as a communication protocol for networked play. Other examples of CORBA (and more specifically ORBit) used in this fashion, are the Mod_CORBA Apache module (http://www.netrinsics.com/Mod_Corba.html) and the Dents DNS server. (http://www.dents.org).

Mod_CORBA is a package that exports much of APACHE's module API via an interface. With this package, you can talk to apache in any CORBA supporting language. It will work without recompiling, or even restarting APACHE. Dents will allow you to access a DNS server via CORBA. It will facilitate such things as remote administration and control of the server. Both of these packages are different from other config tools, as they modify a "live" application. This means that changes happen to the current server on the fly, rather then just modifying one of its config files and restarting it.

Scripting

It is a very important goal of the GNOME project to provide scripting support throughout the system. This support will allow an application developer and advanced users to automate common tasks and rapidly control GNOME features. This is similar to the role that macros and Visual Basic play in the Windows world. The user can write a script to manipulate a gnumeric spreadsheet, perform customized spell checking, or automate some repetitive tasks (this only requires your scripting language to support CORBA).

All of the major scripting languages available on UNIX (such as Perl, Python and Java) have CORBA bindings so they can control components remotely. Additionally, GNOME and GTK+ were written with scripting in mind, and as a result, have a large number of language bindings. See the GTK+ homepage (http://www.gtk.org) for a complete listing.

Defining system services

Many of the procedures carried on most Unix systems these days do not have a standard interface and they rely entirely on tradition. Many times programmers write scripts with various degrees of robustness that manipulate system-configuration settings. The less experience the programmer has, the more likely his scripts that manipulate system information will be prone to subtle errors, race conditions, broken file updates, non-atomic operations and so on. For example, consider the task of programatically adding users, internet services, querying the mail queue, querying the printer queue and so on.

We want to define standard interfaces for various system-related tasks as part of the GNU project. CORBA-based servers would encapsulate all of the details for administering a system and taking the necessary steps to carry their job correctly. Instead of relying on a script to add users, the application writer would invoke various methods on the the System::admin::user interface. To queue messages, instead of relying on a specific mail transfer agent program with some specific semantics to be installed in some magical location, the user would just invoke the System::Mail::deliver interface. The same applies to querying the input mail (by using the GNOME::Mailer interfaces), and the same applies to querying any of the system services (instead of depending on system-specific hacks).

Bonobo

Bonobo is the GNOME Document model. It lets document-based applications embed themselves in each other. Its design is influenced heavily on OLE2 and Active X by Microsoft and has a similar functionality. It is currently in Alpha state, and should be used only by developers, but has been used successfully to embed a Guppi graph within Gnumeric.

For those unfamiliar with OLE2, it provides a framework for applications to embed themselves within the same GUI framework. This facilitates, for example, a graph to be embedded in a word-processing document, or a spell checker to be embedded in a plotting program. Bonobo also provides the ability to wrap GTK-style objects around a component, allowing an application developer to use one in his application. A more technical description is available in the BONOBO cvs repository.

Implementation Notes

In this section, we will discuss a few issues about actually implementing CORBA in a GNOME environment.

ORBit: a CORBA Implementation

ORBit is the main CORBA implementation used in GNOME. It was written to be small and lightweight, and a lot of care has been taken to keep it so. ORBit is written in straight C, and currently only has C bindings (although C++ and Python bindings are coming along rapidly). As a result of this, it is mostly used by programs and libraries that are written in C/C++. Other languages will have other CORBA implementations better suited for their language bindings. However, they will still be able to communicate with ORBit based applications.

The Name Server and libGNORBA

ORBit provides, as part of its services, a generic name server implementation and all of the mechanism necessary to use CORBA in a GNOME environment. However, a lot of the complexity that goes with it is unnecessary for the average application developer, and there is a steep learning curve involved with using CORBA. The GNORBA library was thus created to provide a clean, simple wrapper around common CORBA services in GNOME.

The GNORBA Library is composed of two parts: the name server and the initialization code. The initialization code handles initializing the ORB for the application developer. It also sets up a security mechanism to prevent unwanted connections to the ORB. The authorization scheme is based upon a cookie-passing mechanism, similar to the one that the X windows system uses.

The main naming server in GNOME is the gnome-name-server process. It provides naming services for applications running on the desktop and it is centered around the X-display. Various instances of the name server can be created and activated in different contexts allowing developers to create naming services for their specific applications. CORBA based GNOME applications will start the name server automatically when needed, and applications can register their activation mechanism with it.

Currently, there is no way to stretch this name server accross separate X displays. In addition, the name server is unsuited for applications that do not have connections to the X-server.