Cooperating Applications through Tcl/Tk and DCE

David Richardson
`normanb@citi.umich.edu`

Center For Information Technology Integration
University of Michigan

ABSTRACT

By integrating some aspects of Tcl/Tk with the Open Software Foundation's Distributed Computing Environment, it should be possible to create a framework for cooperating desktop applications. This paper proposes to create such a framework through two steps: by modifying the Distributed Computing Environment RPC compiler to automate development of Tcl/Tk interfaces and by creating a ''Session Manager'' which coordinates inter-application communication.

Introduction

At the University of Michigan's Center for Information Technology Integration (CITI), we are developing distributed computing infrastructure and applications. For the past two years, this development has included working with the Open Software Foundation's (OSF) Distributed Computing Environment (DCE). The DCE is a large, complex system, offering a wide variety of services and interfaces. As with many large systems, developing with the DCE can be a complex and tedious task. It should be possible to simplify this task by integrating some aspects of Tcl/Tk with the DCE. Leveraging off of the rapid development possible with Tcl/Tk, and the distributed services provided by the DCE, it should be possible to develop a new family of applications. These applications would be characterized by their ability to cooperate in a distributed environment.

DCE Background

The Distributed Computing Environment (DCE) is a collection of services designed to simplify the development of distributed and cooperating applications in a heterogeneous environment. It consists of a thread package; an RPC facility; a security service; a cell-wide abstract name space, the Cell Directory Service (CDS); a global name space, the Global Directory Service (GDS); and a suite of services comprising a distributed file system (DFS). It is possible to use these services to create different types of applications, with client-server and peer-to-peer the most popular. A typical DCE server takes advantage of the name space services to ease the burden on the client. The server registers itself at a well known name, in a manner independent of its physical location. A client looks up the name for the service it is interested in, and is given a binding which can be used to contact a server. Because name space entries can be augmented with various attributes, it is possible for the client to pick and choose which instance of a possibly replicated service would best fit its needs.

A Framework for Cooperating Applications

I am interested in creating a framework for cooperating desktop applications. This framework should simplify the task of allowing applications to communicate with each other in a reliable, secure manner. Applications should not be constrained by Tk's current implementation of `send`, which limits communication to applications that are attached to the same X display. Users interested in cooperative work should be able to communicate without needing to know each other's physical location. The framework should also make it easier for applications to request general services, without having to know a particular user's preferences for that service. This paper proposes one possibility for such a framework. The framework includes two major components: the DCE RPC compiler should be modified to generate Tcl wrappers for a subset of RPC interfaces, and a ''Session Manager'' should be created that manages the desktop name space and provides a clearinghouse for inter-application communication.

DCE RPC Compiler

The DCE RPC compiler needs to be modified to generate Tcl wrapper stubs for an arbitrary subset of RPC interfaces. Past experience with writing Tcl interfaces to RPC-based applications has shown that much of the developer's time is spent on an almost rote process of writing the wrappers/conversions between Tcl strings and C-level RPC function call arguments. Generating these types of wrappers falls in line with the RPC compiler's normal task of generating marshaling stubs. This modification should simplify extending Tcl/Tk applications with DCE services.

Session Manager

The ''Session Manager'' (Tk-sm) is a framework for cooperating desktop applications. It provides a communication service that does not depend on X, manages a name space of applications that is location independent, and allows applications to request general services. The Tk-sm is a Tcl/Tk application that has been augmented with a set of DCE services. These services allow the Tk-sm to register applications in the DCE global name space. The DCE services also let the Tk-sm act as a communication proxy, allowing Tcl/Tk applications to communicate with Tcl/Tk applications that are not attached to the same X display. Additionally, the Tk-sm holds a user profile that contains a database of the user's favorite editor, what to do with certain file types, etc. The Tk-sm is able to use this profile to execute generic service requests based on user preferences.

Upon user login, the Tk-sm registers itself in the DCE global name space at a well known location, such as `/.../cell/user/normanb/Tk-sm` (where `/.../` is the root of the DCE global name space and `cell/` describes the local administrative domain). Tk-sm then creates and manages a name space of the user's applications. The Tk-sm uses this name space to support location independent cooperative work, and to provide communication services that are not X-based. While some Tcl/Tk applications might be extended with communication services provided by DCE, it is likely that most will continue to rely on the current X-based `send` mechanism. The Tk-sm can support these ''naive'' applications by creating ''proxy interpreters'' used to represent Tcl/Tk applications running on different X displays. Naive applications `send` to these proxies, and the Tk-sm takes care of the remote transmission using its DCE RPC services.

The name space is similar to Tk's current X interpreter registry, looking something like:

```
/.../cell/user/normanb/Tk-sm/
    edit paper.ms
    Tk-sm
    proxy-ric {edit paper.ms}
    hq
    ppres talk1
    tk-window-manager
    etc.
```

Name space entries might represent local applications, binding points for applications that have been extended with DCE, or proxies for remote applications. Because proxy names are controlled by the local Tk-sm, there is no risk of name space collisions. The framework supported by this name space allows applications to communicate in many different ways:

(a). As the Tk-sm is a Tcl/Tk application, naive Tcl/Tk applications can continue to `send` to each other and to the Tk-sm.

(b). Because the Tk-sm registers itself in the global name space relative to a user's principal, it is location independent. This simplifies cooperative work. Users who wish to communicate only need to know each other's principal, not the physical host they happen to be working at that day.

(c). The Tk-sm coordinates cooperative work. The name space acts as a rendezvous, either for applications that have been extended with an RPC mechanism and the ability to query the name space, or for naive applications through the proxy mechanism.

Consider the case where another user, ric, and I want to work together on a paper using the naive application `edit`. Assume that `edit` communicates with multiple instances of itself by using `send` to reflect changes in a shared document. The Tk-sm would be used to establish proxies, after which our name spaces look something like:

```
/.../cell/user/normanb/Tk-sm/
    edit paper.ms
    wish #2
    hq
    proxy-ric {edit paper.ms}

/.../cell/user/ric/Tk-sm
    edit paper.ms
    proxy-normanb {edit paper.ms}
    wish #2
    tutorial
```

As I use `edit` to make changes to the shared document, `edit` behaves just as it normally would by `send`ing updates to the interpreter `proxy-ric {edit paper.ms}`. My Tk-sm intercepts those `send`s, marshals them in DCE RPC, and transmits them to ric's Tk-sm. His Tk-sm then unmarshals the message and `send`s it to his instance of `edit paper.ms`, which continues to behave just as it normally would. The same process works in reverse when ric makes changes.

Applications that have been extended with DCE services would not have to rely on proxies, but instead would use a subset of the services provided by the Tk-sm, such as the security and profile features discussed below.

(d). The Tk-sm provides a platform for experimenting with authentication and authorization. Along with being a name space manager, the Tk-sm could take advantage of the DCE security services to act as an access control list (acl) manager. This could range from a coarse, all-or-nothing mechanism, to individual acl's for individual applications. At the minimum, it is easy to imagine a system which allows a remote user to ''knock'' on my Tk-sm, and in response I would grant or deny various permissions. If desired, any communication between our applications could take place over the DCE secure RPC. If the target application is knowledgeable about authentication and authorization, the Tk-sm could hand off the remote user's credentials and let the target application manage its own acl's.

(e). A user profile containing preferences for applications such as editors, debuggers, and drawing programs could be used to provide general services. Rather than having to know which editor a particular user prefers, and how to make that editor jump to a particular line, a debugger could make a general request to the Tk-sm. This request would be in some canonical form. The Tk-sm would query its database, and either invoke a new editor or `send` a message to an already running instance. As described above, this would include remote applications. For example, ric and I might be using different editing applications. The Tk-sm would then be used to translate from the canonical editor messages to ones appropriate to our individual preferences.

Conclusion

By integrating some aspects of Tcl/Tk with OSF's DCE, it should be possible to create a framework which simplifies and extends the way desktop applications can cooperate. As a start, I propose modifying the DCE RPC compiler to generate Tcl wrappers, and creating a ''Session Manager'' to coordinate the interaction between distributed applications. This session manager would then be used as a framework for experimenting with various cooperation mechanisms.