Tcl Distributed Programming[†]

Brian C. Smith, Lawrence A. Rowe, Stephen C. Yen
Computer Science Division
University of California
Berkeley, California 94720
bsmith@cs.Berkeley.EDU, larry@cs.Berkeley.EDU, syen@postgres.cs.Berkeley.EDU

1. Introduction

The Tcl Distributed Programming (Tcl-DP) extension to Tcl/Tk [1][2] introduces a suite of commands for creating client-server systems. In this abstract, we provide examples of using Tcl-DP. We also describe the remote procedure call (RPC) abstractions and distributed object system of Tcl-DP.

2. A Client-Server Example

An id server can be built using Tcl-DP. The Tcl code shown in Figure 1a initializes the id server. The `MakeRPCServer` call of Figure 1a creates a socket on port 4545 that will accept client connections. A socket is an endpoint of network communication in UNIX [5]. The Tcl code for client processes is shown in Figure 1b. The `MakeRPCClient` call of Figure 1b connects to the id server and returns a handle to represent a socket. In line 2 of Figure 1b, the `RPC` call retrieves an unique id from the id server, by remotely invoking the `GetId` procedure defined in line 3 of Figure 1a.

3. Distributed Objects

Tcl-DP features a distributed object system, where an object is a collection of fields. With this system, objects may be distributed to other connected processes. Processes are connected with the `MakeRPCServer` and `MakeRPCClient` commands. A change to a field in a distributed object in one process is automatically propagated to other processes. Tcl-DP also provides *triggers*. A *trigger* is a Tcl expression that is evaluated when a field in a distributed object changes.

[†]This research was supported by the National Science Foundation under Infrastructure Grant No. CDA-8722788. Additional support was pro vided by Fujitsu America and Hewlett-Packard.

4. Implementation

The Tk `send` command uses the X ICCCM protocol [3]. Processes connected by `send` use the X Server to pass messages. Since Tcl-DP uses TCP/IP, communicating processes in Tcl-DP do not need to share an X Server connection. This has advantages. First non-X clients need not connect to an X server to exchange messages. Second, the round-trip message time will be less in Tcl-DP since messages do not pass through the X Server.

Tcl-DP provides three types of commands: distributed object commands, RPC commands, and socket manipulation commands. The distributed object system is implemented entirely in Tcl, using the RPC commands of Tcl-DP. The RPC commands were originally implemented entirely in Tcl, using the lower level socket manipulation commands, but have been partially reimplemented in C for speed. The round-trip time for an `RPC`, measuring only `RPC` related code for machines on the same Ethernet subnet, is around 4 milliseconds. The Tk `send` command takes around 19 milliseconds for the same test. The Tcl-DP extension consists of 3,500 lines of C code and 500 lines of Tcl code.

5. Deadlock and Blocking

Any RPC system should be robust in the event of connection failure. In Tcl-DP, programmers can specify an optional millisecond time-out value for `RPC`'s. `RPC`'s also usually block while waiting for a remote return value and can thus prevent other events, such as X events and incoming `RPC`'s, from getting processed. This situation can easily lead to deadlock. In Tcl-DP, programmers can specify the types of events (Tk, file, timer, RPC, all or none) that should be handled while the system is waiting for `RPC` return values. Tcl-DP also supplies the `RDO`

On zonker.cs.Berkeley.EDU:

```
MakeRPCServer 4545
set lastId 0
proc GetId {} {
    global lastId;
    incr lastId;
    return $lastId
}
```

(a)

On linus.cs.Berkeley.EDU:

```
set server [MakeRPCClient zonker 4545]
set id [RPC $server GetId]
```

(b)

Figure 1: An ID Server and Client

command, which unlike the `RPC` command, does not wait for the return value of a remote evaluation but returns immediately. An `RDO` takes around 500 microseconds to complete.

6. Conclusion

The features of Tcl-DP, including its flexible RPC mechanism, distributed object system, and its integration with Tcl/Tk, has allowed us to quickly implement several client-server applications with the same speed and ease as when one creates applications with Tcl/Tk. Among these have been a network name server and a distributed continuous media system [4].

References

[1] Ousterhout, J. “Tcl: An Embeddable Command Language.” *Proc. USENIX Winter Conference*, January 1990.

[2] Ousterhout, J. “An X11 Toolkit Based on the Tcl Language.” *Proc. USENIX Winter Conference*, January 1991.

[3] Scheifler, R., and Gettys, J., with Flowers, J., Newman, R., and Rosenthal, D. *X Window System: The Complete Guide to Xlib, X Protocol, ICCCM, XLFD (Second Edition)*. Digital Press, 1990.

[4] Rowe, L., and Smith, B. “A Continuous Media Player.” *Third International Workshop on Network and Operation System Support for Digital Audio and Video*. 1992, pp. 334-344.

[5] Kochun, S., and Wood, P. *UNIX Networking*. Hayden Book, 1989.