Using Tcl/Tk for a language-driven approach to application design

by

Mark Moraes
D. E. Shaw & Co.
mark-moraes@deshaw.com

Overview

- Tcl.
- Useful (essential?) Tcl extensions.
- The language-driven approach.
- Example: a data communication problem.

What is Tcl?

Tcl (pronounced tickle) is a freely redistributable, embeddable, scripting language designed by Dr. John Ousterhout. It can be linked into applications to serve many different purposes.

- configuration files
- interactive command language
- glue
- prototyping

Applications can enhance the language with application-specific commands.

What is Tcl? (contd.)

Tcl is a programming language that was intended to be:

- simple for humans
- simple to interpret
- simple to interface to C

Strings are the only currency in Tcl.

Tcl overview.

- Commands separated by newlines or semi-colons.
- Commands can be continued onto the next line with backslash.
- A command consists of

  `command_name argument ...`

- Command names are strings.
- Arguments are strings, all passed by value.
- A command returns a string.

Tcl overview. (contd.)

Strings have several forms:

- Simple word eg. `hello`
- A *list* of words eg. `{hello world}`
- Lists can be nested. eg. `{hello {entire world}}`
- The value of variables eg. `$i`
- The result of procedures eg. `[expr $i+5]`
- An expanded list of words eg. `"hello $i [fmtclock [getclock]]"`

Tcl overview. (contd.)

- Control structures are just commands.
- Arithmetic and conditional expressions via `expr` command.

```
proc countdown { i } {
    while {$i > 0} {
        puts $i
        set i [expr $i-1]
    }
    puts READY.
}
countdown 10
```


Tcl overview. (contd.)

- Variables are local to procedures.
- Variables can be *linked* to other variables in enclosing stack frames. This makes "call by name" possible. It also reduces the need for pointers.
- Global is a special case of this notion.
- One can put *traces* on variables to monitor variable accesses.
- An associative array is a powerful form of variable. eg. `$bag(apple)`, `$bucket($liquid)`, `$matrix(0,1)`

Tcl overview. (contd.)

- Exception handling (`catch` and `error`)
- Auto-loading of procedures. Can build libraries.
- Access to internals – `info`, `rename`

Tcl overview. (contd.)

- everything in Tcl is a string
- programs and data are indistinguishable.
- create code on the fly.
- store code for later evaluation.
- use code in arguments to functions.

Going overboard with this is dangerous!

Tcl overview. (contd.)

Core language comes with useful primitives to interface to:

- read/write/stat files
- exec processes
- manipulate lists
- regular expression matching

The *Tk* extension/toolkit.

- Powerful, incredibly simple, attractive X Windows toolkit.
- Event dispatcher.
  - File descriptors. X is a special case.
  - Timers.
  - When-Idle procedures

The *TclX* extension.

- many Unix primitives `unlink`, `cd`
- some useful list primitives `lassign`, `lvarpop`
- keyed lists. slower than associative arrays but printable, good for small, performance-insensitive data, network transmission or persistent storage/retrieval.
- profiler
- ...

The *Tcl-DP* extension.

I think of this as "networking done right". Takes advantage of the Tk event loop.

- remote Tcl procedure call.
- simple interface to sockets (make connection, write, read).
- distributed objects.

The *db* extension.

- Berkeley library for hash and B-tree on disk. B-trees allow duplicate keys, ordered walk.
- Keys and values are strings.
- By using TclX *keyed lists* as values, one can get field structure in records.

The language-driven application philosophy.

- Think of the solution in terms of primitives and glue.
- The application is a Tcl interpreter.
- If necessary, write a set of powerful, general primitives in C, make them a Tcl extension.
- Glue them together with Tcl.
- Glue Tcl applications together with Tcl-DP.

Advantages of the Tcl-driven approach.

- Much faster prototyping.
- Code re-use. Powerful Tcl library for hash, dynamic strings.
- Fewer lines of code, higher-level.
- Effort focused on solving the core problem.
- Highly configurable applications, no ad-hoc config/command languages.
- Client-server happens as a side-effect, almost!

Drawbacks of the Tcl-driven approach.

- Performance. Tcl can be **SLOW**.
  - Sensible choice of primitives.
  - Design with care to avoid Tcl inner-loops.
  - Tcl compiler, faster interpreter coming soon?!
- Requires knowledge of two languages and the language-driven paradigm.
- Testing Tcl is hard, bugs can lurk for a long time.
- Tcl has some quirks. (lists sometimes passed by name, sometimes by value)

Drawbacks of the Tcl-driven approach. (contd)

- Many extensions from different authors, non-standard, namespace problems.
- Evolving.

Example: a communication system.

- Need to communicate data to applications on mainframes from a Unix environment. The data consists of real-time prices (of financial instruments).
- Each of the mainframes speaks a somewhat different protocol, based on BISYNC. Framing:

  `SOH header STX data ETX SUM`
  return `ACK` or `NAK`.
- The format for data-bytes are totally different, but all text.
- Different checksum algorithms for SUM.

Example: a communication system. (contd.)

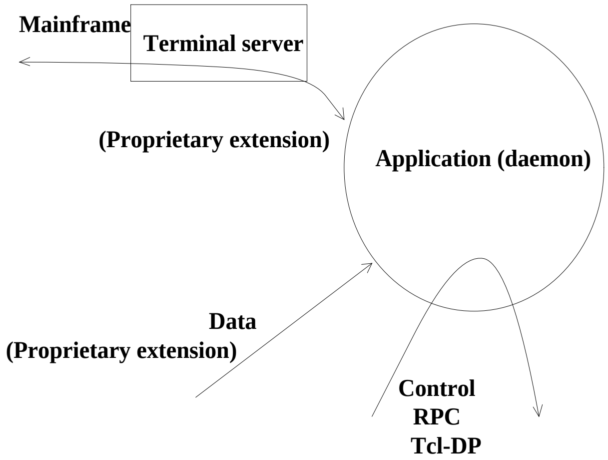
I wrote a small packet framing/de-framing extension for Tcl. 1100 lines of C.

- `pkt_open`: opens the line, conditions it, sets up callback whenever data appears. The internal callback checks the checksum, responds with ACK or NAK, and then invokes a user-specified Tcl command with the header and data from the packet.

- `pkt_write`: frames header and data suitably, calculates checksum, sends packet, waits for ACK or NAK, retries a few times on NAK before giving up and returning error.

- `pkt_close`, `pkt_debug`, `pkt_close`

Example: a communication system. (contd.)



Example: a communication system. (contd.)

Hindsight:

- It's worked well for over a year.

- A new data link was added with a day of programming by another programmer.

- The packet library had to be enhanced when one of the machines *improved* the protocol (multi-byte checksums, flow control!)

ftp locations

- `comp.lang.tcl` Usenet newsgroup FAQ posting.
  `rtfm.mit.edu:/pub/usenet/news.answers/tcl-faq/`

- db on
  `ftp.cs.berkeley.edu:/ucb/4bsd/db.tar.Z`

- PostScript for these slides on
  `ftp.cs.toronto.edu:/pub/moraes/decus-tcl.ps.gz`