Married with Tcl

1st European Tcl/Tk User Meeting, Hamburg, 15 June 2000

Jürgen Schönwälder

Computer Science Department
Technical University Braunschweig
Bültenweg 74/75
38106 Braunschweig
Germany

Phone: +49 531 391 3289
Email: `schoenw@ibr.cs.tu-bs.de`
Web: `http://www.ibr.cs.tu-bs.de/~schoenw/`

Tcl Extensions for Network Management (Tnm)

- Tnm is a Tcl extension for scripting (SNMP) management applications.
- First public Tnm release back in 1993, based on Tcl 6.X.
- Current CVS snapshot of the scotty 2.0 package based on Tcl/Tk 8.3.1.
- Scotty is being used on many sites world-wide
  - to educate people in network management,
  - to automate network management tasks,
  - to drive test-suites, or
  - to run management simulators.
- Tnm provides powerful and efficient asynchronous event-driven APIs.
- Simple management application (tkined) included in the source distribution.

Example: SNMP heartbeat polling in Tcl/Tnm

```
package require Tnm 3.0
namespace import Tnm::*

proc poll {s} {
    $s get sysUpTime.0 {
        if {"%E" != "noError"} {
            syslog error "[%S cget -address]:[%S cget -port] SNMP heartbeat failed: %E"
        }
    }
}

proc heartbeat {ip {port snmp}} {
    set s [snmp generator -address $ip -port $port]
    set j [job create -interval 5000 -command [list poll $s] -exit [list $s destroy]]
    return $j
}

proc check {} {
    foreach j [job find] {
        if {[$j cget -status] != "running"} {
            puts "$j:\t[$j cget -status]\t[$j cget -interval]\t[$j cget -time]"
        }
    }
}

foreach agent {switch bridge xxxx} { heartbeat $agent }
job create -interval 300000 -command check
```

How I got married with Tcl...

- Needed a flexible prototyping environment for research on network management.
- Stumbled across Tcl (actually TclX) on one of the first free BSD releases.
- Read the classic Tcl paper and immediately realized that an embeddable language like Tcl meets my requirements.
- Learned about Tk which was much easier to use as any other GUI toolkit at that time.
- Started to study the Tcl/Tk source code and was very impressed by their quality.
- There was a active community and it was easy to get into direct contact with the people maintaining the source.

$\Longrightarrow$ Decided to build software on top of Tcl and Tk in early 1993.

$\Longrightarrow$ Lots of positive excitement since it was possible to make progress very fast.

Being married with Tcl...

- Multiple safe Tcl interpreters:
  - Nice toy when every interpreter had to run in a Web browser.
  - The practical value of supporting it was marginal.

- Cross platform portability:
  - Forcing applications to interface with a completely new IO system is a high price.
  - The glue language started to dictate how to write applications or extensions.

- Thread support:
  - Making code thread-safe requires manual code inspection and is error prone.
  - Event-driven programming works just fine for an embeddable command language since there is hardly a benefit from computational parallelism.

- Byte compiler and `Tcl_Objs`:
  - Nice and welcome speed improvements plus binary data support.
  - Requires to rewritten your extension and do reference counting debugging.

Things that probably went wrong...

- Underestimation of the value of the Tk toolkit:
  - In the early 90's, many people started to use Tcl because of Tk.
  - Tk did not really evolve through all the years.
  - Tk nowadays looks a little bit old-fashioned and lacks standard widgets in the core.
  - The integration of the well tested "plus" patches alone took several years!
- Tcl as an embeddable command language:
  - The original design was not a general purpose programming language.
  - The original design still shows, even after several years of language "evolution."
  - There is no consistent syntax across all commands.
  - Some command name choices (`fconfigure`) seem to have happened ad-hoc.
- Modularization and compatibility:
  - The core implementation has become increasingly twisted and complex.
  - Dynamic loading capabilities and the package system are not effectively used.
  - Backwards compatibility should not be used as an argument to keep old design mistakes around forever.

Would I fall in love again?

- I still believe that scripting languages play a key role in practical network management.
- I still believe that using an existing host language is better than trying to design a special purpose domain specific language.
- Today, I would take a serious look at Python since it has a much "better" language design and hopefully provides more robustness for maintaining collections of scripts.
- Python also comes with a much more complete distributions to get things going faster.
- However, robust event-handling is a core language requirement - Tcl shines here.
- I would take a serious look at GTK+ for building GUIs.

$\Longrightarrow$ Tcl/Tk has serious competitors today and my answer is not unlikely to be different.

Some lessons learned...

- I learned a lot about good C programming from reading the Tcl sources. Thanks!

- Always separate the scripting API wrapper from the C/C++ code that implements the functionality.

  $\Longrightarrow$ Tk is a really bad example in this respect!

- Supporting multiple different Tcl/Tk versions is hard, even though the Tcl maintainers are very disciplined.

- Cross platform portability is still a pain, especially if it makes solutions on the primary platform more complex (combined with a lack of knowledge about the other platforms).

- Compatibility with common language constructs is very useful — but only if these common language constructs are well designed.

  $\Longrightarrow$ The Tk % substitution mechanism to access event details is flawed!


Thanks for listening!
⟹ Now beat me up. ⟸