Tcl/Tk para Herramientas EDA

Elías Todorovich

INCA/INTIA, Departamento de Computación y Sistemas, Fac. Cs. Exactas, UNCPBA, 7000 Tandil, ARGENTINA
etodorov@exa.unicen.edu.ar

Resumen: En este tutorial se trata de explicar como se usa y la importancia del lenguaje Tcl/Tk para la automatización de tareas en las herramientas de diseño electrónico. Tcl/Tk se está consolidando desde hace unos años en este área de aplicación con importantes ventajas para usuarios y fabricantes. Primero se resumen los principales elementos del lenguaje Tcl. Luego se da una introducción al *toolkit* para el desarrollo de interfaces gráficas: Tk. En tercer lugar, se menciona para varias herramientas de desarrollo de sistemas, como se utiliza este lenguaje para escribir *scripts*. Se trata de presentar cada concepto con la ayuda de un ejemplo sencillo. Por último se presentan dos aplicaciones que también sirven de ejemplo para mostrar la potencia de un este lenguaje: una automatiza una técnica de estimación de consumo de potencia media y en la otra se muestra como hacer una simulación a medida de una tarjeta gráfica sencilla.

1 Introducción

Hasta hace unos años, cada compañía dedicada al diseño electrónico desarrollaba su propio lenguaje de macros o *scripts* para su software. Además de invertir recursos para implementar la funcionalidad de cada herramienta, había que desarrollar estos lenguajes propietarios tan diferentes unos de otros como dos productos no estándares pueden llegar a serlo. En este tutorial se va a ver como se revirtió este problema gracias al uso de un lenguaje común, que crece dentro de una comunidad de programadores y empresas bajo el paradigma de código abierto, aportando soluciones a todas las herramientas informáticas donde se integra.

Tcl (*Tool Command Language)* es un lenguaje para escribir *scripts* y también es un intérprete para dicho lenguaje diseñado para integrarse en otras aplicaciones. Como otros lenguajes de *scripting* (sh, csh, ksh de UNIX, Perl por ejemplo) no es tipado, las variables son *strings* (salvo que se invoque específicamente el comando *expr* para que se evalúan como expresiones matemáticas), soporta listas y arreglos, e implementa estructuras de control y llamadas a procedimientos. Por último, y siendo el tema de este tutorial, Tcl/Tk es el estándar de facto para los *scrips* en herramientas EDA (*Electronic Design Automation*).

Tk es el *toolkit* de Tcl para crear interfaces gráficas y está diseñado para que sea sencillo de manejar, está compuesto por una cantidad de elementos básicos: *frame*, *image*, *button*, *window manager*, etc.

John Ousterhout [1] comenzó el desarrollo de Tcl/Tk en Berkeley, en 1987. Aún en el ámbito de su cátedra, este profesor tenía el problema de tener un lenguaje de *scripts* lleno de flaquezas e incompatible con los demás para cada herramienta que desarrollaba. La

solución que encontró fue construir un lenguaje interpretado capaz de ser integrado a cualquiera de sus herramientas. Luego llevó su proyecto a Sun Microsystems. En general, los programas interactivos necesitan lenguajes de comandos [2]: Sun pretendía crear una alternativa a MS Visual Basic. En 1998, Ousterhout fundó Scriptics, que lidera el desarrollo de Tcl/Tk actualmente [3].

Tcl/Tk está diseñado de manera tal que se puede incorporar como intérprete a otras aplicaciones (Modelsim, Xilinx ISE, Symplify, etc. [4]), no siendo necesario inventar un lenguaje de órdenes para cada aplicación. Naturalmente Tcl/Tk permite la construcción de funciones complejas mediante *scripts* y proporciona elementos de programación generales: variables, control y modularización mediante procedimientos. Otras ventajas son que se le puede agregar primitivas escritas en C, su sintaxis es mínima, es fácil de escribir y de aprender, implementa el acceso a archivos y *sockets*, puede ejecutar programas externos, y existe en múltiples plataformas [5]. También es importante mencionar que Tcl/Tk, además de desarrollarse como un proyecto de código abierto, es de libre distribución.

1.2 Un Primer Encuentro con Tcl

La sintaxis de Tcl es simple. En los ejemplos que siguen está el comando a la izquierda y el resultado de evaluar el mismo a la derecha. Para crear una variable y asignarle un valor se usa `set`:

```
set a 47                47
```

Uno de los puntos fundamentales pata entender Tcl es la forma en que realiza sustituciones, en el siguiente ejemplo, se hace la sustitución de la variable `a`:

```
set b $a                47

set b [expr $a+10]      57
```

También hay que comprender claramente como se separan las palabras desde el punto de vista del intérprete (*quoting*):

```
set b "a is $a"         a is 47

set b {[expr $a+10]}    [expr $a+10]
```

Definir y usar procedimientos permite realizar un diseño modular:

```
proc fac x {
           if $x<=1 {return 1}
             expr $x*[fac [expr $x-1]]
}

fac 4                   24
```

2 El Lenguaje Tcl

Los principales elementos de Tcl se introducen con la ayuda de ejemplos que se pueden escribir en la consola de cualquier herramienta con un intérprete Tcl/Tk (*wish*, *tclsh*).

2.1 Sintaxis y Reglas de Sustitución

Un *script* Tcl es simplemente una secuencia de comandos. Los comandos se separan con saltos de línea ó punto y coma.

Un comando Tcl se compone de una o más palabras separadas por espacios. La primer palabra es el nombre del comando, las demás son sus argumentos y el resultado es un *string*. *Hello Word*, es el primer programa que se escribe en cualquier lenguaje:

```
puts “¡Hola a todos!"
```

Aunque parezca demasiado poco, en el párrafo anterior se resume la sintaxis básica de Tcl. Ahora, para comprender como se interpretan los comandos, hay que ver el mecanismo de sustitución de variables. La sintaxis es:

```
$varName
```

La sustitución puede ser en cualquier lugar de una palabra. Por ejemplo:

```
set b 56              56

set a b               b

set a $b              56

set a $b+$b+$b        56+56+56

set a $b.3            56.3
set a $b4             no existe tal variable
```

El siguiente paso para aprender Tcl, es estudiar el mecanismo de sustitución de comandos. La sintaxis es:

```
[script]
```

El intérprete primero evalúa `script`, y luego sustituye `[script]` por el resultado. La sustitución puede ser en cualquier lugar dentro de una palabra. Por ejemplo:

```
set b 8                  8
set a [expr $b+2]        10
```

Finalmente debe comprenderse como se separan las palabras para el intérprete. Las palabras se separan con espacios ó punto y coma, excepto que se aplique una de las siguientes reglas:

- Las comillas dobles evitan la separación de palabras. El intérprete considera el texto entre las comillas como un sólo argumento, previa sustitución de comandos y variables. Por ejemplo:

```
set x 10
set a "x es $x; y es [expr 10+10]"    x es 10; y es 20
```

- Las llaves evitan la separación y la sustitución: El intérprete considera el texto entre llaves como un sólo argumento, difiere la sustitución para después. Por ejemplo:

```
set a {[expr $b*$c]}                  a = [expr $b*$c]
```

- La barra invertida (*backslash*) escapa caracteres especiales:

```
set a word\ with\ \$\ and\ space      a = word with $ and space
```

- La barra invertida también escapa saltos de línea (sirve para continuar líneas)

2.2 Comandos

Los comandos se pueden aprender a medida que se necesitan. La sintaxis de Tcl es tan simple porque las estructuras de control son comandos, y no forman parte de ella.

Ejemplo: Hacer que una lista *b* quede en el orden inverso al de la lista *a*:

```
set b ""
set i [expr [llength $a] - 1]
while {$i >= 0} {
            lappend b [lindex $a $i]
            incr i -1
}
```

Los principales comandos de control son: `if`, `for`, `switch`, `break`, `foreach`, `while`, `eval`, `continue`, `source`.

3 El *Toolkit* Tk

El hecho de poder escribir interfaces de usuario mediante un *script* acelera el tiempo de desarrollo, es más simple que usando cualquier librería en C, y se puede separar del resto de la aplicación.

El programa "*Hello, world*", utilizando Tk, es el siguiente y el resultado de evaluarlo aparece en la Fig. 1:

```
toplevel .top
button .top.hello -text "Hello, World!" -command "destroy .top"
pack .top.hello
```

Fig. 1. "*Hello, world*" con Tk

La opción `-command` se usa para especificar la respuesta que se produce al usar el botón.

Los componentes gráficos (*Widgets*) se unen unos a otros formando una jerarquía de composición. El siguiente ejemplo es un navegador de archivos (Fig. 2):

```
listbox .list -yscroll ".scroll set" -width 20 -height 20
pack .list -side left
scrollbar .scroll -command ".list yview"
pack .scroll -side right -fill y
wm title . "File Browser"
```

```
if {$argc > 0} {
            set dir [lindex $argv 0]
} else {
            set dir .
}
foreach i [lsort [glob * .*]] {
            .list insert end $i
}
bind .list <Double-ButtonPress-1> {
            browse $dir [selection get]
}
bind all <Control-c> {destroy .}

proc browse {dir file} {
            if {$dir != "."} {
              set file $dir/$file
            }
            if {file isdirectory $file] {
              exec browse $file &
            } else {
              if [file isfile $file] {
                exec emacs $file &
              } else{
                error "can't browse $file"
              }
            }
}
```

Las principales clases de *Widget* implementadas en Tk son: `Frame`, `Menubutton`, `Canvas`, `Label`, `Menu`, `Scrollbar`, `Button`, `Message`, `Scale`, `Checkbutton`, `Entry`, `Listbox`, `Radiobutton`, `Text`, `Toplevel`. Lo mismo que para los comandos, se puede aprender a usar cada uno de estos componentes a medida que se necesita [6].

Por ejemplo: `button` se usa de la siguiente manera

```
button name -text "Text" -command { Tcl_code... }
```

4 Tcl/Tk en Herramientas EDA

Tcl es el estándar de facto para herramientas EDA (*Electronic Design Automation*) y aplicaciones CAD (*Computer-Aided Design*) [3]. Este hecho tiene como beneficio adicional que solamente se tiene que aprender Tcl para automatizar todas estas herramientas. Más aun, con Tcl/Tk se pueden integrar y coordinar herramientas de distintos fabricantes.

Por ejemplo, en el caso de simuladores como Modelsim o ActiveHDL, se puede manejar todos los comandos del mismo, compilar archivos VHDL, cargar y elaborar diseños, ejecutar simulaciones, poner *breakpoints*, tener acceso a las señales del diseño y leer su estado, o asignar valores a las mismas, tener acceso a todos los menús del simulador, y como se verá en un ejemplo al final de este tutorial, se puede crear una interfaz gráfica para controlar una simulación para obtener lo que puede llamarse “simulación a medida”.

4.1 Xilinx ISE

Una busqueda (*.tcl) en la instalación de ISE da 172 archivos encontrados, muchos de los cuales automatizan tareas de la herramienta.

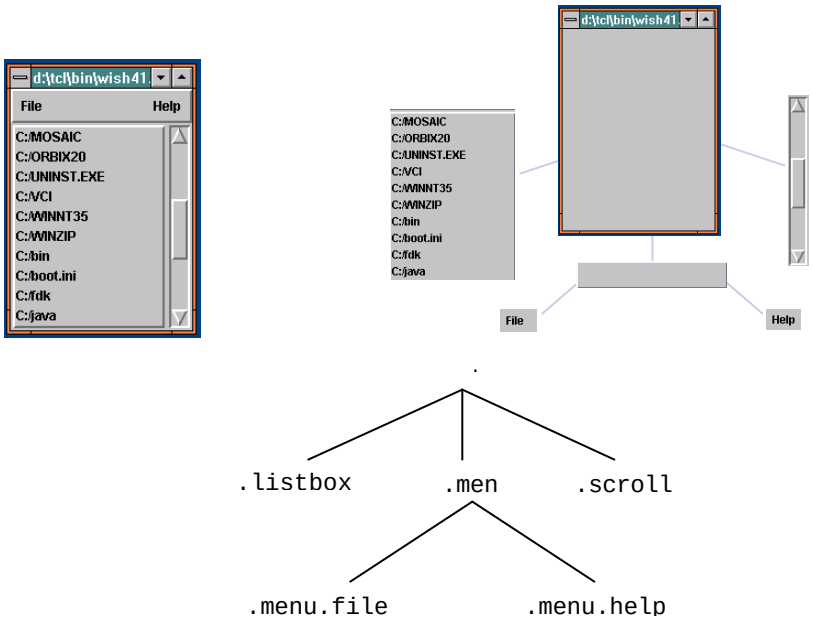

Fig. 2. Navegador de Archivos y su jerarquía de composición de elementos gráficos

XPower. Se trata de una herramienta de estimación de consumo de potencia y análisis térmico para FPGAs y CPLDs, que se distribuye con Xilinx ISE. XPower provee varias formas para la entrada de datos y una de ellas es mediante un *script* Tcl [8].

Cuando se usa XPower por línea de comandos, tiene la siguiente sintaxis:

```
xpwr design[.ncd] [constraint[.pcf]] [options]
```

Para indicar que se quiere utilizar un archivo Tcl se usa: `-t tclscript.tcl`

Mediante comandos Tcl se puede especificar la capacidad de salida de una pista, la frecuencia de una señal, la tensión de operación del dispositivo, etc. Por ejemplo:

```
set netFreq 50
setNetFrequency clk_in_IBUFG $netFreq
return 0
```

4.2 Quartus II

Como en el resto de las herramientas EDA, los *scripts* Tcl pueden automatizar Altera Quartus® II [9], desde tareas simples como compilar un diseño a otras tan complejas como automatizar todo un flujo de diseño utilizando la API de funciones de Quartus II.

Quartus II permite ejecutar *scripts* Tcl interactivamente o por línea de comandos:

```
quartus_cmd -f <script_name>.tcl
```

Este software provee también una forma de documentar las opciones de un proyecto mediante la generación automática del correspondiente *script* Tcl. Esto se puede hacer desde el menú *Project*: *Generate Tcl File for Project*.

En el siguiente ejemplo se ve como se crea un proyecto:

```
if [!project exists filtref] {
          project create filtrofir
}
project open filtrofir
```

4.3 Simplify

Una busqueda (*.tcl) en la instalación de Simplify da 39 archivos. En esta herramienta se puede usar Tcl para controlar la síntesis o realizarla iterativamente con diferentes opciones y tecnologías, y para los archivos de restricciones (*constraint files*) [10] [11].

4.4 Modelsim

Una búsqueda (*.tcl) en la instalación de Modelsim da 139 archivos encontrados, entre ellos, ejemplos y demos. Para que se observe la importancia del lenguaje objeto de este tutorial para Modelsim: su interfaz gráfica está desarrollada en Tcl/Tk y los comandos que se escriben en la consola de la ventana principal se evalúan con el intérprete Tcl/Tk [12]. En el sitio *web* de Model Technologies, [13], se pueden ver algunos ejemplos y en este tutorial se presentan dos aplicaciones de Tcl/Tk usando este simulador.

4.5 ActiveHDL

Este simulador también extiende su capacidad para la construcción de macros y *scripts* mediante Tcl/Tk. El *Design Flow Manager* de este producto está basado en Tcl y permite a los usuarios seleccionar e invocar cualquier combinación de herramientas de otros fabricantes para síntesis e implementación.

5. Ejemplos

5.1 Estimación de Consumo

Una de las técnicas de estimación de consumo medio para circuitos CMOS se basa en estadísticas [14]. Se trata de una aplicación de los métodos de estimación de la media: donde el tamaño de la muestra debe ser lo suficientemente grande como para que el

parámetro estimado (consumo medio) tenga la precisión requerida. En el siguiente fragmento de código Tcl se tiene un ciclo que se repite hasta que la muestra cumpla con un criterio de corte. Se ejecutan una serie de programas externos. El lenguaje en que fueron estos programas (código fuente) no importa.

```
set END_SIM False
while { $END_SIM == False} {
          exec generator.exe
          do simulate.do
          do savevec.do
          catch { exec Transitions.exe } Samples
          exec Update.exe
          catch { exec Cuter.exe } END_SIM
}
```

5.2. Simulación a Medida de una Tarjeta Gráfica

En la aplicación que se presenta en esta sección se modela una tarjeta de video simple en modo texto: PicoVideo, que pertenece a una colección de periféricos simples para CPUs embebidas en FPGA [15]. El problema está en cómo ver la salida generada por el adaptador. Evidentemente, con sólo ver la forma de onda generada, no se puede verificar el funcionamiento de una tarjeta gráfica. A cualquier diseñador le agradaría tener una vista tal como aparecería en el mo nitor como parte de la simulación.

```
proc draw_tester {} {
          toplevel .tester
          wm title .tester "PicoVideo Tester V0.1"
          wm geometry .tester +600+600
          wm minsize .tester 700 700
          wm maxsize .tester 700 700
          wm deiconify .testerwm protocol .tester \
WM_DELETE_WINDOW "destroy .tester"
          label .tester.lb1 -text "Tester initialized..."
          pack .tester.lb1 -side top
          connect_monitor
}

proc connect_monitor {} {
          when -label bp {/picovideo_vgaifce_tb/vts/bpoint} {
            if {[examine -value /picovideo_vgaifce_tb/vts/bpoint]\
== 1} then {
              new_frame }
            }
}

proc new_frame {} {
          if [file exists "frame.ppm"] {
            set img [image create photo -format ppm -file \
"frame.ppm"]
            .tester.lb1 configure -image $img }
}
```

Para comprender el ejemplo es necesario entender cómo se conectan eventos en señales con código Tcl/Tk en Modelsim:

```
when { señal } { script }
```

`when` es un comando de Modelsim que agrega *breakpoints* sobre señales: se hacen determinadas acciones cuando se cumple una condición especificada (valor en la señal o tiempo de simulación). En el ejemplo, cuando se tiene un *frame* completo, la señal `/picovideo_vgaifce_tb/vts/bpoint` adquiere el valor 1. Esto hace que se alcance el *breakpoint* que se ha definido y se dispare el procedimiento `new_frame`.

Tambien es necesario ver cómo controlar el simulador desde la interfaz Tk. Véase este otro ejemplo:

```
set rb $obj.runsome
button $rb -command "transcribe run 1000 ns" -text "Run 1000"
pack $rb
```

Al utilizarse el botón se simulan 1000 ns.

Conclusiones. Estado del Proyecto Tcl/Tk

La disponibilidad del código fuente, distribuciones, documentación, ejemplos y notas de aplicación hacen que cualquier persona interesada pueda aprender y programar en Tcl/Tk inmediatamente. Existen intérpretes de Tcl/Tk en todas las plataformas UNIX/X, y Win32/Win16/Mac desde la versión 7.4/4.1. Se estima que existen 100.000 programadores de este lenguaje, con una comunidad de desarrolladores que contribuyen al proyecto.

Tcl/Tk se está usando en múltiples áreas de Ciencias de la Computación, pero en este tutorial se destaca su presencia en las herramientas EDA, que se está consolidando desde hace unos años con importantes ventajas para usuarios y fabricantes.

No se puede terminar este tutorial sin mencionar los puntos atacables de Tcl/Tk: Debe prestarse atención a las reglas de sustitución que pueden parecer complejas; hay lenguajes competidores de Tcl/Tk como JavaScript o Visual Basic; y como en el resto de los lenguajes interpretados, puede tener problemas de rendimiento.

Referencias

1. Ousterhout, J.K.: "Tcl and the Tk Toolkit", Addison-Wesley.
2. Ousterhout, J.K.: "Scripting: higher level programming for the 21st Century", IEEE Computer, Vol: 31, Issue: 3, Mar 1998, pp: 23-30
3. Distribuciones de Tcl/Tk, información, documentación: http://dev.scriptics.com
4. "EDA/CAD success stories", http://www.tcl.tk/customers/success/edacad.html
5. Welch, B.: "Practical Programming in Tcl and Tk", Prentice Hall.
6. Tk Widgets Very Quick Reference, http://www.pconline.com/~erc/tk_qref.htm
7. GUI-builder para Tcl/Tk, http://spectcl.sourceforge.net/
8. Xilinx, Inc.: "Development System Reference Guide – ISE 5", 2002.
9. Altera Corporation: "Scripting with Tcl in the Quartus II Software", Application Note 195, December 2002.

10. Synplicity, Inc.: “Synplify Pro, User Guide and Tutorial” Ver Capítulo 4: Advanced Techniques, Oct. 2001.
11. Synplicity, Inc.: “Synplify Pro Reference Manual” Ver Capítulo 4: Tcl Commands and Scripts, Oct. 2001.
12. Tutorial de Modelsim, Lección 13: Tcl/Tk and Modelsim
13. Tips, aplicaciones y links de Model Technology , http://www.model.com/resources/tcltk.asp
14. Najm, F. N., Xakellis, M. G.: Statistical estimation of the switching activity in VLSI circuits. VLSI Design, vol. 7, no. 3 (1998) 243-254
15. Lopez Buedo, "PicoDevices: Una Colección de Sencillos Periféricos para Microprocesadores Embebidos en FPGAs", documento interno, Escuela Politécnica Superior, UAM, 2003.