Programación Básica de Interfaces de Usuario Utilizando TCL

Ing. Jesús Antonio Álvarez Cedillo.
Ing. Marlon David González Ramírez
Profesores del CIDETEC-IPN

El sistema operativo linux contó, desde su aparición en el mundo, con una herramienta de programación para la creación de interfaces de usuario llamada TCL, que es la abreviación de las palabras en inglés **"tool command language",** y formada por dos grandes secciones:

a. Una sección de órdenes. Un lenguaje de órdenes para el desarrollo de aplicaciones (**Figura 1**).

b. Una sección de lenguaje formada por un intérprete local.

Con TCL viene integrada una herramienta llamada TK, una variación de TCL que permite su uso en entornos gráficos XWindows. TCL y TK fueron diseñados y desarrollados por el profesor John Ousterhout, en la Universidad de Berkeley.

Como un lenguaje para el desarrollo de aplicaciones, su estructura es similar a otros lenguajes existentes en UNIX; además, permite la ejecución de otros programas y proporciona suficientes elementos de programación, tales como variables, ordenes para control de flujo, y la posibilidad de crear procedimientos.

Estas características permiten construir aplicaciones complejas muy fácilmente, ensamblando programas ya existentes. Una característica que distingue a TCL de otros lenguajes de órdenes es la facilidad de incorporar un intérprete común a cualquier tipo de aplicación, así es fácil estructurar una aplicación como un conjunto de órdenes primitivas que serán organizadas por medio de un guión para adecuarse mejor a las necesidades de los usuarios.

Los programas que permiten a TCL tener nuevas características de operación son llamados extensiones; la más importante es la antes mencionada TK, como herramienta de desarrollo en el entorno XWindows, que permite la creación y la manipulación de interfaces de usuario.

La técnica basada en guiones para la programación de nuevas interfaces de usuario presenta las ventajas siguientes:

a. Rapidez de desarrollo.
b. Interfaz de alto nivel con el entorno X.
c. Separación entre la interfaz y la aplicación.

El Nacimiento de TCL

El objetivo principal de Sun Microsystem cuando diseño TCL, fue

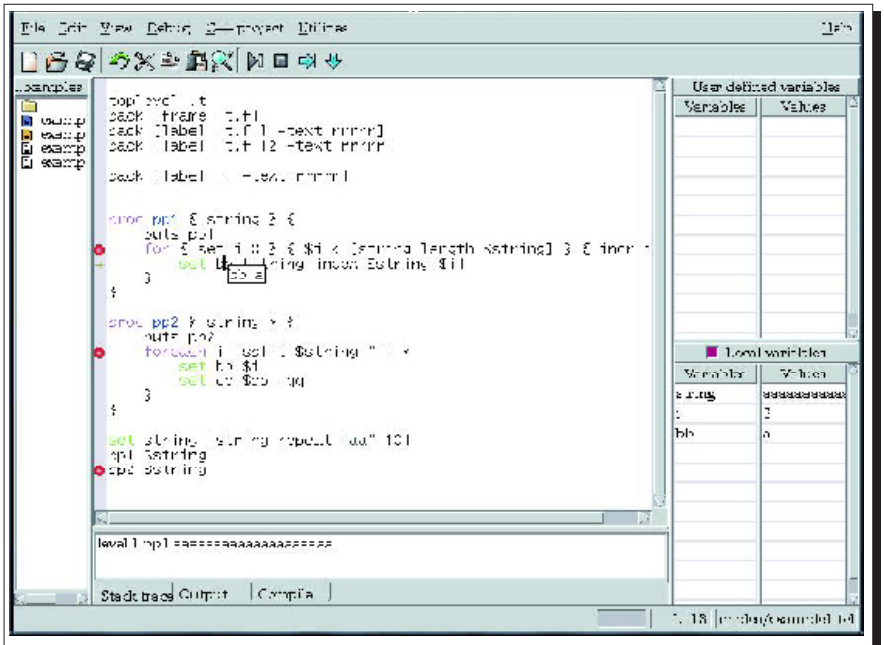

Figura 1. Sección de ordenes usando TCL STUDIO.

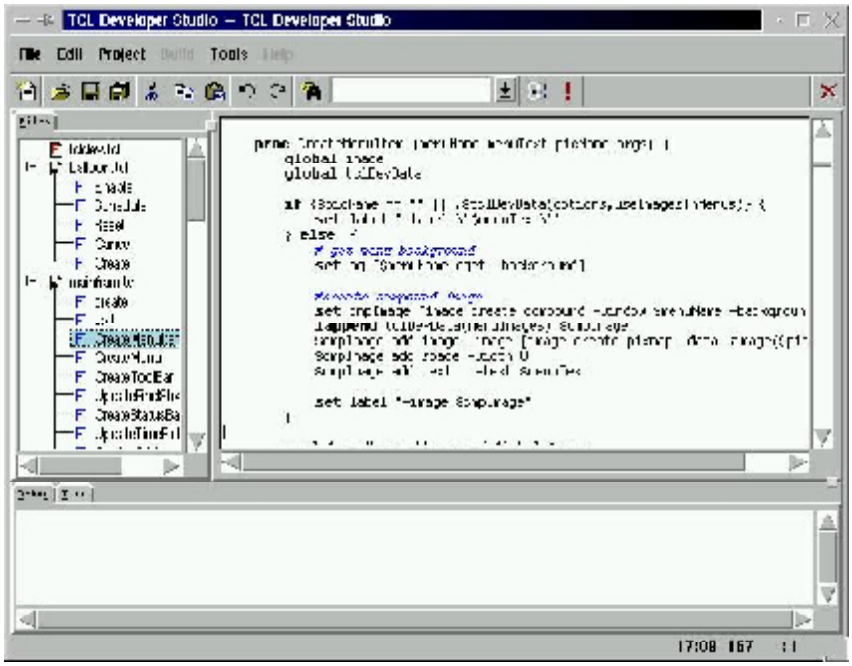

Figura 2. Ejemplo de programación en TCL STUDIO.

crear un lenguaje complementario al lenguaje de programación Java inmune a los virus informáticos y que fuera multiplataforma. Este hecho permitiría crear agentes de software que navegaran por la red Internet de una máquina a otra. En su origen, el profesor John Ousterhout y su equipo pretendían diseñar un lenguaje de muy alto nivel, con la finalidad de enlazar de forma flexible un cierto número de módulos escritos en diferentes lenguajes tales como C y Fortran, al cual le llamaron **meta programación.**

La meta programación permite al sistema que, si existe la modificación de algún módulo, este solo deberá reinterpretarse. TCL enlaza los diferentes módulos y determina si necesitan ser compilados nuevamente. En C++, la modificación de un módulo requiere regenerar las relaciones existentes entre los mismos, característica inexistente en TCL donde la modificación de los módulos no supone la re compilación del resto.

TCL, como una herramienta de órdenes, actúa de manera similar al intérprete Cshell de UNIX. El formato de las órdenes y el uso de comentarios mantienen la misma sintaxis (**Figura 2**). La sintaxis general de una orden TCLes:

orden ***argu1 argu2 argu3...***

La instrucción **orden** es el nombre de la TCL o de un procedimiento desarrollado con anterioridad. Los argumentos se toman siempre como valores de cadena; ello quiere decir que TCL es un intérprete de cadenas, en el que absolutamente todos los argumentos y órdenes se interpretan inicialmente como objetos del tipo cadena de caracteres. TCL no interpreta los argumentos, sino que los pasa directamente a la orden, quien es la encargada de interpretarlos; por ejemplo, en la siguiente orden de asignación

```
set x [expr 3*sin (35)].
```

el procedimiento set actúa como orden, x es un argumento o bien la variable asignada, y [expr 3*sin (35)] es el último argumento; los corchetes no forman parte del argumento, sino que son modificadores que indican que todo el contenido es un solo argumento.

TCL usa el carácter # para indicar que la línea que se está escribiendo corresponde a comentario del programa. Esta línea se omite durante el tratamiento del intérprete, y tiene sentido únicamente como aclaración de algún fragmento del código. Ejemplo:

```
# esto es un comentario
set x 6; # a x se le da el valor de 6
```

Variables

Una variable debe verse como un simple almacén de memoria, al que se le ha añadido una etiqueta que lo identifica en forma única. Como en muchos de los lenguajes de programación usuales, pueden utilizarse caracteres ASCII exceptuando los especiales entre los que debemos señalar los signos de exclamación, interrogación y puntuación, corchetes, llaves, signos de operadores aritméticos, el espacio en blanco, acentos, etc.

Al igual que el lenguaje C, del que TCL ha heredado muchas de sus características, en la formación de los identificadores se hace una distinción entre mayúsculas y minúsculas.

Se usa la orden **set** para la asignación de valores a una variable. Éstas pueden tener cualquier nombre y no es necesario inicializarlas antes de ser usadas. Si la variable que se va a utilizar en la asignación no existe, el intérprete TCL la crea. La sintaxis de la operación de asignación es la siguiente:

set <variable> <valor>

Por ejemplo:

```
set x 6
```

Al igual que se pueden construir variables mediante la orden set, también es posible borrar dichas variables. Para ello se utiliza la orden unset, cuyo formato es el siguiente:

```
unset <variable>
```

Por Ejemplo:

```
unset x
```

Si se busca visualizar el valor de la variable X el mensaje que mostraría seria el siguiente:

```
set X
can't read «x»: no such variable
# No se puede leer la variable, al haber sido destruida
# antes
```

La instrucción **unset** puede referirse a varias variables (separadas por espacio en blanco), en cuyo caso el resultado es la destrucción de todas las variables que se indiquen.

Si en algún momento de la ejecución del programa se desea conocer todas las variables existentes en el mismo, es posible:

```
info vars
```

El intérprete devolverá una lista con todas las variables que se encuentran vivas en ese instante.

Operaciones de Entrada y Salida

En la mayoría de los sistemas operativos, el tratamiento que se hace de los dispositivos de entrada y salida es similar al tratamiento de archivos. Es decir, el sistema operativo asocia a cada dispositivo un archivo, de manera que se consigue el tratamiento uniforme de las entradas y de las salidas independientemente de su origen o destino. Así, al dispositivo estándar de entrada, que suele coincidir con el teclado, le es asignado un archivo cuyo nombre es stdin o entrada estándar; el dispositivo estándar de salida, que suele coincidir con el monitor, tiene asignado el archivo stdout; por último, el archivo asignado al dispositivo por el que se va a dar salida a los errores se llama stderr. Las ventajas de esta forma de tratamiento son claras:

a. Permite conseguir uniformidad en las operaciones
b. Es posible redireccionar las entradas y salidas a archivos distintos al estándar, de manera que un mismo programa puede ser utilizado para tomar las entradas desde teclado, o bien de otro archivo.

La forma de indicar a TCL que se desea realizar una entrada es mediante la utilización de la sentencia gets, cuyo formato es el siguiente:

```
gets <archivo> [<línea>]
```

Donde <archivo> es el archivo del que se va tomar la entrada, y [<linea>] es un parámetro opcional en el que se depositan los valores leídos. Cuando se realiza una lectura mediante la instrucción **gets** se lee una línea completa del archivo de entrada.

Ejemplo:

Orden	Resultado
set x 25	
gets stdin x	
	999
set $x	
	999

En cuanto a las salidas, la orden TCL para realizarlas es:

```
puts <archivo> <línea>
```

Donde <archivo> es el archivo en el que deseamos mostrar la salida, y <linea> es el texto que deseamos mostrar. Cuando la salida deseamos realizarla en la pantalla o en cualquier dispositivo estándar de salida. Entonces el nombre del archivo asociado será stdout.

Por ejemplo:

```
# cálculo del área de un triángulo
puts stdout «Introduzca la longitud de la base"
Introduzca la longitud de la base
gets stdin base
4
1
puts stdout «Introduzca la longitud de la altura»
Introduzca la longitud de la altura
gets stdin altura
5
1
set area [expr $base*$altura/2]
10
puts stdout «El área del triángulo de base $base y altura $altura es $area»

El area del triángulo de base 4 y altura 5 es 10
```

En el ejemplo anterior se muestra la utilización de los agrupamientos mediante comillas dobles. En primer lugar se evalúan las expresiones que se encuentran dentro del agrupamiento, y a continuación se resuelve.

Control de Flujo

Para resolver problemas donde interviene una condicional, de manera similar a otros lenguajes de programación existe la orden if. Cuya sintaxis es la siguiente:

```
if <condición>
then :<sentencia 0 y 1>
else <sentencia Ó y 2>
```

Ante un valor verdadero de la expresión lógica, el algoritmo pasará a ejecutar una secuencia determinada de sentencias.

Por ejemplo:

```
if {$a > $maximo}
then
{
set maximo $a
}
else
{
set a $máximo
}
```

La ejecución de este ejemplo sería la siguiente:

1. En primer lugar, se evalúa la expresión $a > $máximo.

2. Si el resultado de esta expresión fuese VERDADERO (es decir, distinto de 0) entonces se ejecutaría la sentencia que asigna a la variable máximo el valor de la variable a.

3. Si por el contrario la variable tuviera un valor menor o igual a máximo, en este caso se ejecutaría la sentencia del grupo, es decir, la variable a pasaría a tomar el valor de la variable máximo.

Como se puede observar, la estructura de una selección simple permite que el grupo de sentencias que acompañan a las cláusulas then y else sean o bien una sentencia simple, o bien un conjunto de sentencias. Por tanto, se puede concluir que en ese grupo de sentencias es posible incluir otra sentencia condicional simple. Con esta estructura se obtienen las llamadas selecciones anidadas.

En TCL se permiten cláusulas de evaluación a partir de la sentencia **switch**. Incluye la posibilidad de tomar caminos según el parecido que tenga el valor de una cadena al valor esperado. De esta forma, la selección de una alternativa no viene dada por la coincidencia exacta con un valor, sino por la afinidad a dicho valor. El formato de la sentencia **switch** es el siguiente:

```
switch opción cadena
valor1   bloque1
valor2   bloque2
...
valorN   bloqueN
default bloque
```

opción es un modificador que permite determinar la forma de evaluación de los valores que debe tomar la variable. Así, existen varias posibilidades en dicha evaluación:

a) Exact, es la opción por defecto, y debe entenderse como que el valor de la cadena coincide exactamente con el valor indicado. En esta situación, la utilización de la sentencia switch coincide con las alternativas múltiples tradicionales.

b) Glob, en cuyo caso se entenderá que la comparación del valor de la cadena con el valor propuesto se realizará atendiendo a las reglas generales de comparación de cadenas existentes en UNIX

c) Regexp, en cuyo caso la coincidencia del valor de la cadena con los valores indicados se realizará atendiendo al cumplimiento por parte de la cadena de la expresión regular que se señale.

Los ciclos son considerados también en el lenguaje de TCL, y se manejan bajo la misma estructura de las órdenes de bifurcación. El primer tipo de ciclo son los ciclos controlados por contador, que suelen venir representados en los diferentes lenguajes por la estructura **for**.

En este tipo de ciclos, el control de las iteraciones del cuerpo del ciclo se realiza mediante una variable que, en cada pasada por la condición, se incrementa o se decrementa, finalizando la ejecución del ciclo cuando se alcanza un determinado valor.

Por lo tanto, la principal característica de los ciclos controlados por contador es que se conoce el número de iteraciones que se van a realizar del ciclo,

En TCL, la sentencia **for** es similar a la sentencia correspondiente del lenguaje C, con las mismas particularidades. La estructura sintáctica es la siguiente.

```
for <inicial> <test> <final> <cuerpo>
```

Donde <inicial> corresponde a la sentencia o grupo de sentencias de TCL y que sirve como inicialización de nuestra variable contador. Esta sentencia se ejecuta únicamente la primera vez que se ejecuta el cuerpo del ciclo, ignorándose en el resto de iteraciones.

<test> corresponde a la condición de salida del ciclo, y se ejecutará siempre como primera instrucción del cuerpo del ciclo, excepto en la primera iteración en que se ejecuta tras la inicialización. El grupo de sentencias < fina1 > se ejecuta tras el cuerpo del ciclo en cada iteración, y se utiliza para la modificación del valor del contador

Ejemplo: Calcular el factorial de un número n, leído previamente del teclado:

```
#Lectura del número a leer
puts stdout “Introduzca un número entero mayor que cero”
gets stdin numero
#Estudio del factorial
  if $numero<0
  then {puts stdout “No existe el factorial de $numero”}
     else { set factorial 1;
for { set i 1} {$i<$numero} { incr i }
  {set factorial [expr $factorial*$i]}
puts stdout “ $numero != $factorial”}
```

La entrada con centinela es empleada, cuando no se conoce el número de ejecuciones del ciclo, sino que éste va a estar ejecutándose hasta que desde la entrada de datos

se produzca un evento que determine la finalización del ciclo. Esta variable es el centinela. Dependiendo de la forma en que se evalúe el centinela, se puede hablar de dos tipos de ciclos:

a) De evaluación previa, en los que la condición de finalización del ciclo se evalúa al inicio del ciclo, y el cuerpo se repite mientras dicha condición sea verdad. La condición es evaluada incluso antes de entrar por primera vez en el cuerpo del ciclo por medio de la estructura **while**.

b) Evaluación posterior, en los que la condición se evalúa al final de la ejecución del cuerpo del ciclo. En la primera iteración, la condición no es evaluada, por lo que este tipo de ciclos asegura que el cuerpo del ciclo se ejecuta siempre al menos una vez, sentencia **repeat until**

Por ejemplo, deseamos calcular el promedio de las calificaciones de los alumnos de una clase y se tiene un archivo de entrada en el cual existe una marca de fin de archivo que indica cuándo han terminado los datos. Se establece que esta marca será la introducción por teclado del valor 1.

```
gets stdin nota
set media 0
set numAlumnos 1
while {$alumno <> Óý1}{
     incr numAlumnos 1;
   set media {expr media + nota};
     gets stdin nota }
set media {expr media/ 1{ decr numAlumnos
1 }
puts stdout “La nota media es $media”
```

El último tipo de ciclos son los controlados por variable. La evaluación del final del flujo del ciclo se hace mediante una variable que va tomando valores en un rango determinado, indicados en la propia sentencia. Básicamente, este tipo de ciclos son idénticos a los ciclos controlados por contador, solo que ahora no es necesario decrementar o incrementar el valor de la variable al final de la ejecución del ciclo.

El nuevo valor de la variable que hace las veces de contador se extrae secuencialmente de entre los valores de una lista y en este tipo de ciclos se conoce con exactitud el número de iteraciones que se van a realizar del cuerpo del ciclo. La estructura que va a tener este tipo de ciclo es la siguiente:

foreach <variable> <lista> <cuerpo>

<variable> tomará en cada iteración el valor que se encuentre en la lista de valores que se señalen a continuación.

Ejemplo: Se calculará la suma de los diez primeros números primos.

```
set suma 0
foreach primo { 1 2 3 5 7 11 13 17 19 23}{
  set suma {expr suma + primo } }
```

Un procedimiento se define en TCL por medio de la orden **proc**. Su sintaxis es:

proc <nombre> <parámetros> <cuerpo>

Ejemplo:

```
proc factorial { n }
{
set acum 1;
for {set i 1} {$i<=$n} { set acum [expr
$i*$acum]}
return $acum
}
```

Estructuras de Datos Complejas

Para TCL las cadenas de caracteres son el único tipo de dato. Esto es, los objetos son todos una secuencia de caracteres o signos; debido a ello existen múltiples órdenes que permiten el tratamiento de las cadenas de caracteres de una forma mucho más extensa que en C o Pascal.

La sintaxis general para la orden **string** es:

string <operación> <variable> <otros argumentos>

El primer argumento determina qué hacer; el segundo sobre qué cadena se va a realizar la acción, y el resto, si los hay, depende de la operación a realizar. En el caso de que la orden implique la utilización de un índice de valores se comienza a contar desde 0, al igual que ocurre en C. El término **end** se refiere al último carácter de la cadena.

Funciones Básicas

a) **string compare** cadena1 cadena2: Compara cadenas. Devuelve 0 si son iguales, 1 si cadena1 es menor que cadena2 y 1 en otro caso.

b) **string first** cadena1 cadena2: Devuelve el índice en cadena2 en que comienza la primera ocurrencia de cadena 1 o 1 en otro caso.

c) **string index** cadena índice: Devuelve el carácter número índice de cadena.

d) **string length** cadena: Devuelve la longitud de cadena.

e) **string match** modelo cadena: Devuelve 1 si en cadena se encuentra modelo, en caso contrario, devuelve 0.

g) **string range** cadena i j: Devuelve la subcadena de cadena que comienza en i y acaba en el j.

h) **string tolower** cadena: Convierte cadena a mayúsculas.

i) **string toupper** cadena: Convierte cadena a minúsculas.

j) **string trim** cadena caracteres: Elimina M comienzo y M final de cadena los caracteres indicados por caracteres. Por defecto caracteres son los espacios en blanco.

k. **String trimleft** cadena caracteres: Igual que en el caso anterior, pero solo elimina caracteres del comienzo de cadena.

l. **String trimright** cadena caracteres: Como en el caso anterior, pero solo elimina caracteres del final de cadena.

Ejemplo:

```
set x “me llamo Antonio”
set y “, Alvarez”
string length $x
string length $y
```

Una lista es una estructura de datos en la que sus elementos son todos del mismo tipo. La principal característica de las listas, y que las diferencia de otras estructuras de datos es que se desconoce a priori el número de elementos que las componen. Es decir, las operaciones elementales que tendrán estas estructuras de datos serán las operaciones de inserción y borrado. Las listas se implementan en TCL como si se tratara de cadenas.

Debido al tratamiento especial que se hace de las listas, éstas se deben usar únicamente si van a contener pocos elementos, o bien para la construcción de órdenes que se evaluarán posteriormente. Para el manejo de mayores volúmenes de datos las matrices son más adecuadas y eficientes.

Órdenes relacionadas con las listas:

a) **list** argu1 argu2: Crea una lista con todos los argumentos.

b) **index** lista i: Devuelve el elemento enésimo de la lista.

c) **lenght** lista:Devuelve el número de elementos de una lista.

d) **range** lista i j :Devuelve el conjunto formado por los elementos de¡ i al j en la lista.

e) **append** lista arg1 arg2: Añade elementos a lista.

f) **insert** lista i ar1 ar2: Inserta en lista a partir de la posición i los argumentos, generando una lista nueva.

g) **replace** lista i j ar1 ar2: Reemplaza los elementos entre el i y el j de una lista por los argumentos. Genera una nueva lista.

h) **search** modo lista valor: Devuelve el índice del elemento de lista que coincide con valor. El modo puede ser –exact -glob ó regexp. Devuelve 1 si no se encuentra.

i) **sort** indicador lista: Ordena elementos de una lista según el indicador, que puede ser: ascii, integer, real, increasing, decreasing ó command

j) **concat** lista1 lista2 lista3: Junta varias listas en una sola.

El tratamiento que se hace de las matrices en TCL difiere bastante al de otros lenguajes de programación, especialmente en el aspecto de que no se hace mención explícita al tamaño del arreglo en el momento de su declaración, cosa que sí es necesario hacer en otros lenguajes como C o Pascal.

Una matriz es una variable con un índice, el cual es tratado como una cadena de caracteres, por lo que se puede pensar en las matrices como aplicaciones de una cadena (el índice) en otra cadena (el valor del elemento de la matriz). A pesar de ello, internamente se implementa como una tabla hash, estableciéndose una correspondencia entre cada cadena utilizada como índice y cualquier valor ordinal, por lo que el tiempo de acceso a cada elemento es prácticamente el mismo. El orden de los índices viene determinado por el orden de las cadenas de caracteres.

Para señalar el índice del arreglo se delimita con paréntesis. Los elementos de una matriz se definen con **set** y se obtienen por medio de la sustitución $. Por tanto, para que la matriz esté totalmente definida habrá que definir todos y cada uno de los elementos de la misma.

Ejemplo:

```
set matriz (0) Manolo set matriz (2) Pepe
```

La orden **array** devuelve información sobre las variables de matrices y esta formada por las siguientes funciones:

a) **array exist** matriz: Devuelve 1 si matriz es una variable de tipo array

b) **array get** matriz: Devuelve una lista que alterna entre el índice y los valores de la matriz.

c) **array names** matriz modelo: Devuelve los valores del índice de matriz ó solo los coincidentes con el patrón modelo.

d) **Matriz** 1ista Inicializa: Una matriz a partir de una lista que debe tener valores que alternan entre el índice y el valor

e) **array size** matriz: Devuelve el tamaño de matriz

Ejemplo:

```
set media 0
for {set i 0} {$i<100} {incr i} {
   gets stdin nombre ($i);
   gets sidin nota ($i);
   sel media [expr $media+$nota ($i)]
}
set media [expr $media/100.0]
for {set i 0} {$i<100} {incr i}
   if {$nota ($i) <media}
   then {puts stdout "$nombre ($i) ha aprobado"
}
```

Conclusión

La principal aportación de este trabajo es mostrar la capacidad de TCL como lenguaje de programación y sus diversas extensiones a través de un estudio mas detallado de los eventos en cada una de las modalidades soportadas, correspondientes al diseño de una interfaz de usuario. Como puede observarse, este tipo de herramienta es una opción económica, confiable y de trabajo multiplataforma que permite la integración de capacidades inteligentes a la interfaz y la integración de los sistemas en un solo entorno de programación.

Las principales ventajas detectadas fueron las siguientes:

a. Sencillez de programación.

b. Rapidez en el desarrollo de las aplicaciones (Tecnología RAD).

c. Gran velocidad comparado con otros lenguajes interpretados.

d. Facilidad de modificación de las aplicaciones.

e. Multiplataforma.

f. Gran número de extensiones gratuitas.

g. Posibilidad de incorporar nuevos comandos en lenguaje C/C++.

De la misma manera se presentaron los siguientes inconvenientes:

a. Excesivamente lento comparado con los lenguajes compilados.

b. Necesidad del intérprete para ejecutar una aplicación.

c. Difícil de depurar, debido a que, a diferencia de un compilador, el intérprete sólo «traduce» el código que se ejecuta; pudiendo quedar partes del código sin depurar porque el intérprete nunca las haya ejecutado.

Finalmente, el conocer a fondo una extensión gráfica como el TK, permitirá tener como limite en la creación de una interfaz tan solo la imaginación.

Bibliografía

[1] Ousterhout, John. *An Introduction to TCL and TK*. Addison-Wesley Publishing, 1993

[2] Harrison, M.; Mclennan, M. *Effective Tcl/Tk Programming*. Addison-Wesley, 1998.

[3] Página de SUN script: www.sunscript.com