Construction d'IHM

Tcl/Tk

Alexis Nédélec

Ecole Nationale d'Ingénieurs de Brest
Technopôle Brest-Iroise, Site de la Pointe du Diable
CP 15 29608 BREST Cedex (FRANCE)
e-mail : nedelec@enib.fr

Table des Matières

Tcl: Introduction	3
Interpréteur de Commandes	6
Structures de contrôle	13
Fonctions	15
Tk: Introduction	20
Placement de Widgets	24
Gestion d'Evénements	34
Ressources de Widgets	48
Extensions	49
Canaux de Communication	58
Conclusion	76
Bibliographie	78

Tcl: Introduction

Qu'est Tcl/Tk (tikle ticket):

- ▷ John Ousterhout (1988) Université de Berkeley (Californie)
- ▷ **T**ool **C**ommand **L**anguage: un langage de commandes
- ▷ **T**ool**K**it: avec des composants d'IHM

Objectif initial de Tcl

- ▷ faire le lien entre programmes compilés sur plate-forme Unix
- ▷ se substituer aux shells (capacités de programmation trop limitées)
- ▷ sorte de glu pour faire collaborer des programmes dans une application

Aujourd'hui Tcl se suffit à lui-même pour développer des applications:

- ▷ traitement de chaînes de caractères
- ▷ manipulation de systèmes de fichiers
- ▷ contrôle de flux de données
- ▷ création de connexion réseau

Tcl: Introduction

C'est un langage de script constitué :

- ▷ d'une collection de mots-clé
- ▷ de règles syntaxiques
- ▷ d'un interpréteur du langage

qui se veut "universel" pour contrôler

- ▷ le traitement de données
- ▷ l'échange de données
- ▷ la communication application-utilisateur
- ▷ la gestion des Interfaces Homme-Machine (Tk)

Script Tcl :

- ▷ suite de commandes Tcl
- ▷ séparées par ligne ou point-virgule

Syntaxe Tcl

Syntaxe d'une commande Tcl:

- ▷ Nom de commande
- ▷ suivi d'arguments éventuels:
  - ◇ une sous-commande
  - ◇ une option
  - ◇ une liste d'arguments
- ▷ retourne une chaîne de caractères

Exemples de commandes Tcl:

```
string compare -nocase NOcase noCASE
set set set
set version [info tclversion]
```

Interpréteur Tcl

Lancement de l'interpréteur

```
{logname@hostname} tclsh
%
```

Exemples de commandes Tcl

```
{logname@hostname} tclsh
% set set set
set
% puts set
set
% set version [info tclversion]
8.4
% string compare -nocase NOcase noCASE
0
```

Tcl + Tk

Interpréteur Tcl + Extension Tk: `wish` (**wi**ndowing **sh**ell)

```
{logname@hostname} wish
%
```

Exemples de commandes Tcl+Tk

```
{logname@hostname} wish
% set version [info tclversion]
% message .msg -text  "Version Tcl/TK $version" -bg green -w 400
% pack .msg
% button .but -text OK -command exit
% pack .but
```

Commandes

Interprétation de commandes

▷ en fonction de la place de chaque terme dans une instruction

```
% set set set
set
```

▷ premier terme: nom de commande (`set`)

▷ deuxième terme: nom de variable (pour la commande `set`)

▷ troisième terme: chaîne de caractères (pour la variable `set`)

Affichage:

```
% puts "Valeur de la variable set : \"$set\" "
Valeur de la variable set : "set"
```

Commentaires:

```
% puts $set; # Affiche la valeur de la variable set
set
```

Variables

Variables simples:

▷ `set x pi; set pi 3.14; set y $pi`

Listes:

▷ `set punch [list rhum "citrons vert" {sucre de canne}]`

Tableaux associatifs:

▷ `set style(Hendrix) Blues; set style(Santana) Latino`

```
% set punch [list rhum "citrons vert" {sucre de canne}]
rhum {citrons vert} {sucre de canne}
% lindex $punch
rhum {citrons vert} {sucre de canne}
% set style(Hendrix) Blues; set style(Santana) Latino
Latino
% array names style
Santana Hendrix
% array get style
Santana Latino Hendrix Blues
```

Substitution

Avant exécution d'une commande:

▷ Tcl interprète tous les arguments

Deux opérations d'interprétation:

▷ Substitution (symbole `$`): remplacer les expressions dans une instruction

▷ Interpolation (symbole `[]`): exécuter les instructions imbriquées

Exemples de substitution:

```
% expr 1 + 1
2
% set a  "expr"
expr
% set b  "1 + 1"
1 + 1
% set operation "$a $b"
expr 1 + 1
```

Interpolation

Instructions imbriquées:

```
% set operation [$a $b]
2
% set texte "Le résultat de $b  est  [$a $b]"
Le résultat de 1 + 1  est  2
```

Annulation d'interpolation (symbole {}):

```
% set texte {Le résultat de $b  est  [$a $b]}
Le résultat de $b  est  [$a $b]
```

En résumé pour évaluer une instruction l'interpréteur

▷ déréférence toutes les variables précédées d'un `$`

▷ exécute toutes les commandes imbriquées sauf celles entre accolades

▷ ne fait qu'une seule passe sur une ligne de commande

Evaluation d'Instructions

Pour forcer un nouveau passage dans l'interpréteur:

▷ commande `eval`

Cette commande traite ses arguments comme une instruction à évaluer

```
% set operation {expr 1 + 1}
expr 1 + 1
% eval $operation
2
```

Utilisation de la commande `eval`

▷ pour lancer des commandes non-prédéterminées

```
proc EvaluationPostFix {nom} {
  eval ${nom}PostFix
}
```

Structures de Contrôle

▷ syntaxe proche du C:

◇ `if, then, else, switch`

◇ `while, for, foreach`

```
if {$OnOff=="On"} {puts "On"} {puts "Off"}
if {$OnOff=="On"} then {puts "On"} else {puts "Off"}
```

```
if {$OnOff=="On"} {            |    switch $OnOff {
  puts "On"                    |      On  { puts "On" }
} elseif  {$OnOff=="Off"} {    |      Off { puts "Off" }
    puts "Off"                 |
} else {                       |      default {
    puts "Unknown"             |        puts "Unknown"
}                              |      }
                               |    }
```

Structures de Contrôle

```
set Guitaristes [list Hendrix Santana "Van Halen"]
set counter [llength  $Guitaristes]

while {$counter>=0} {
  puts  [lindex  $Guitaristes $counter]
  incr counter -1
}
for {set i 0} {$i < [llength $Guitaristes]} {incr i 1} {
  puts [lindex $Guitaristes $i]
}
foreach guitariste $Guitaristes {
    puts "$guitariste"
}
```

Fonctions

Création de fonctions dans un fichier source (`fonctions.tcl`)

```
proc factorielle {nb} {
  if {$nb>1} {
    return [expr $nb * [factorielle [expr $nb - 1]]]
  }
  return 1
}
```

Appel du script (commande `source`):

```
% source fonctions.tcl
% factorielle 4
24
```

Variables Locales/Globales

```
proc localvar {} {              |    proc globalvar {} {
                                |      global var
  set var 11                    |      set var 11
  return [incr var]             |      return [incr var]
 }                              |   }
```

Tests local/global:

```
% set var 1                     |  % set var 1
1                               |  1
% localvar                      |  % globalvar
12                              |  12
% incr var                      |  % incr var
2                               |  13
```

Niveaux d' Evaluation

Visibilité de variables

```
proc exclamer {mot} {
   set x "$mot !"
   dupliquer
   return [string toupper $x]
}
proc dupliquer {} {
   uplevel {
      set x "$x $x"
   }
}

% exclamer hello
HELLO ! HELLO !
```

Niveaux d' Evaluation

Plusieurs niveaux d'évaluation **relatif**

▷ pour remonter le bloc d'exécution d'une fonction

▷ de plusieurs niveaux par rapport à la fonction elle-même

```
proc exclamer {mot signe} {
   set x ""
   dupliquer
   return [string toupper $x]
}
proc dupliquer {} {
   ponctuer
   uplevel { set x "$x $x" }
}
proc ponctuer {} {
   uplevel 2 { set x "$mot $signe" }
}
```

Niveaux d' Evaluation

Niveaux d'évaluation Absolu:

▷ pour descendre le bloc d' exécution d'une fonction

▷ à partir du niveau global d'exécution (`#0`)

▷ valeur par défaut: `uplevel #1`

```
proc dupliquer {} {
   ponctuer
   uplevel #1 { set x "$x $x" }
}
proc ponctuer {} {
   uplevel #1  { set x "$mot $signe" }
}
```

Tk: Introduction

Structure d'application Tk

- ▷ gestion d'IHM
  - ◇ création d'une hiérarchie de composants
  - ◇ placement des composants sur l'IHM
  - ◇ affichage dans l'application
- ▷ gestion des interactions
  - ◇ implémentation des actions
  - ◇ connexion des actions sur les composants

```
{logname@hostname} wish
% message .msg -text  "Version Tcl/TK: [info tclversion]" \
          -bg green -w 400
.msg
```

Création de widgets

Instanciation de widget:

```
nomDeClasse nomInstance -uneOption saValeur -uneAutreOption ...
```

Classes de Widgets usuelles:

- ▷ toplevel, canvas, frame, menu
- ▷ label, button, checkbutton, radiobutton, menubutton
- ▷ entry, text, spinbox, scale, scrollbar, listbox
- ▷ message,image

Exemple de création d'arbre de widgets:

```
frame .cadre -borderwidth 4 -relief sunken
label .cadre.icone -bitmap warning
message .cadre.msg -justify left\
                   -text "Attention.\n Vous allez lancer une action"
```

Hello World

Mon premier programme:

- ▷ création d'une arborescence de widgets
- ▷ agencement des widgets (**`pack, grid, place`**)
- ▷ interaction utilisateur avec un widget

```
{logname@hostname} wish
% source hello.tcl
```

```
TO DO: Help action
TO DO: Help action
```

Hello World

```
# variables Tcl
set version [info tclversion]
# widgets Tk
message .msg -text  "Version Tcl/TK $version" -bg green -w 400
button .okButton -text OK -command exit
button .helpButton -text Help
# packing
pack .msg
pack .okButton
pack .helpButton
# interaction Tk
bind .helpButton <Double-ButtonPress-1> help
# actions Tcl
proc help {} {
   puts "TO DO: Help action"
}
```

Placement de Widgets

Commandes de placement de widgets:

▷ `pack`: accolage des widgets par leurs bords

▷ `grid`: placement sur une grille

▷ `place`: positionnement géométrique relatif ou absolu

```
# packing
pack .okButton  .helpButton -side left -padx 10 -expand true -fill x
# pack .okButton  .helpButton -side left
# pack .okButton  .helpButton -padx 10
# pack .okButton  .helpButton  -expand true
# pack .okButton  .helpButton  -fill x
```

Commande Pack

Options de la commande pack:

`-side`	positionnement dans la cavité
`-after`	placement après un composant
`-before`	placement avant un composant
`-anchor`	ancrage dans l'espace alloué
`-expand`	extensibilité dans la cavité
`-fill`	remplissage de l'espace
`-in`	insertion dans un autre parent
`-ipadx`	ajout d'espace horizontal à l'intérieur du widget
`-ipady`	ajout d'espace vertical à l'intérieur du widget
`-padx`	ajout d'espace horizontal à l'extérieur du widget
`-pady`	ajout d'espace vertical à l'extérieur du widget

Commande Pack

Valeurs des options de la commande `pack`:

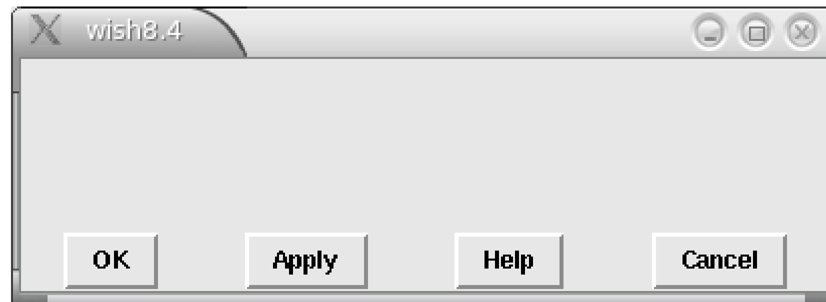
`-side`	`top, bottom, right, left`
`-after`	le widget
`-before`	le widget
`-anchor`	`n, ne, e, se, s, sw, w, nw, center`
`-expand`	`true, false`
`-fill`	`x, y, both, none`
`-in`	le widget
`-ipadx`	valeur entière
`-ipady`	valeur entière
`-padx`	valeur entière
`-pady`	valeur entière

Commande Pack

Exemple d'agencement de widgets par la commande `pack`

```
button .okButton -text OK
button .helpButton -text Help
button .cancelButton -text Cancel
button .applyButton -text Apply

pack .okButton  .helpButton  .applyButton .cancelButton -side left
pack .okButton  .helpButton  .applyButton .cancelButton -anchor s
pack .applyButton -before .helpButton
pack .okButton  .helpButton  .cancelButton .applyButton -expand true
```



Commande Grid

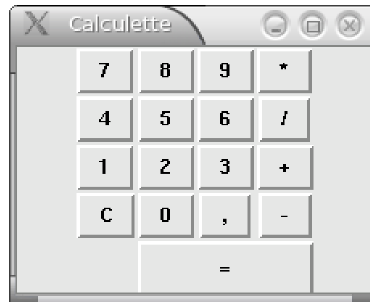
Options de la commande `grid`:

`-row`	numéro de ligne pour le placement
`-column`	numéro de colonne pour le placement
`-rowspan`	étalement sur plusieurs lignes
`-columnspan`	étalement sur plusieurs colonnes
`-sticky`	alignement sur la cavité (`n,nw...`)
`-in`	insertion dans un autre parent
`-ipadx`	ajout d'espace horizontal à l'intérieur du widget
`-ipady`	ajout d'espace vertical à l'intérieur du widget
`-padx`	ajout d'espace horizontal à l'extérieur du widget
`-pady`	ajout d'espace vertical à l'extérieur du widget

Commande Grid

Exemple d'agencement de widgets par la commande **grid**

```
wm title . "Calculette"
for {set i 0} {$i <= 9} {incr i} {
  button .btn$i -text $i -borderwidth 2 -relief raise
}
foreach symbol {+ - * / = C ,} {
  button .btn$symbol -text $symbol -borderwidth  2 -relief raise
}
grid .btn7 .btn8 .btn9 .btn*
grid .btn4 .btn5 .btn6 .btn/
grid .btn1 .btn2 .btn3 .btn+
grid .btnC .btn0 .btn, .btn-
grid .btn= -row 4 -column 1 -columnspan 3 -sticky ew -ipadx 3 -ipady 3
```



Commande Place

Options de la commande `place`:

`-x, -relx`	abscisse absolue, relative
`-y, -rely`	ordonnée absolue, relative
`-width, -relwidth`	largeur absolue, relative
`-height, -relheight`	hauteur absolue , relative
`-bordermode`	prise en compte de la bordure
`-in`	insertion dans un autre parent
`-anchor`	point d'ancrage

Commande Place

Exemple d'agencement de widgets par la commande `place`

```
wm title . "Pensee du Jour"

destroy .msg
message .msg -text "Etre ou ne pas etre Place" \
             -justify center -bg grey75 -relief ridge
button .okButton -text OK

place .msg  -relx 0.5 -rely 0.5 \
            -anchor center \
            -relwidth 0.75 -relheight 0.50

place .okButton -in .msg -relx 0.5 -rely 1.05 -anchor n
# move .msg and .okButton together
place .msg  -x 100 -y 100
```

Sous-Commandes

Sous-commandes d'agencement de la commande `pack`:

- ▷ `configure`: modifier les valeurs d'option
- ▷ `slaves`: retourne la liste des widgets assujetties à une fenêtre
- ▷ `forget`: retirer des widgets de la liste
- ▷ `info`: retirer des widgets de retournela liste
- ▷ `propagate`: redimensionnement ou non

Avec en plus pour la commande `grid`:

- ▷ `rowconfigure, columnconfigure`, avec options:
  - ◇ `-minsize`: taille minimum de ligne/colonne
  - ◇ `-pad`: espacement à ajouter
  - ◇ `-weight`: poids lors de redimensionnement

Sous-Commandes

```
grid columnconfigure . 3 -minsize 50
grid rowconfigure . 4 -minsize 50
```

Et les sous-commandes:

- ▷ `size`: nombre total de lignes, colonnes
- ▷ `bbox`: informations sur les positions des cellules sur la grille
- ▷ `location`: informations sur la cellule contenant une position
- ▷ `remove`: retrait temporaire de cellules

```
# retrait de .btn1 .btn2
grid remove .btn1 .btn2
# reaffichage de .btn1 .btn2
grid .btn1 .btn2
```

Gestion d'Evénements

Programmation événementielle

- ▷ succession dévénements: file d'attente
- ▷ scrutation de boucle d'événements
- ▷ table de liaisons (événements, widgets, sript Tcl associé)

Liaison Evénement/Script Tcl/Tk

- ▷ commande **bind**

```
bind .msg <ButtonPress> { puts "Ou suis-je sur l'ecran dans le widget \
                                %W : X: %X Y: %Y"}
bind .msg <Motion> { puts "Ou vais-je sur le widget %W : x: %y y: %y"}
bind .okButton  <ButtonPress> { puts "Je te quitte"; exit}
```

Descripteurs d' Evénements

Modificateur+Type d'événement+bouton souris ou touche clavier

- ▷ `<Modifier-EventType-ButtonNumberOrKeyName>`

Modificateurs

- ▷ touche clavier: `Control`, `Alt`, `Shift`, `Lock`, `Option`, `Meta`
  - ◇ `<Control-Shift-KeyPress-a>`, `<Control-A>`
- ▷ bouton souris: `B1...B5...`
  - ◇ `<Button2-Motion>`, `<B2-Motion>...`
- ▷ multi-click: `Double,Triple,Quadruple`
  - ◇ `<Double-ButtonPress-1>`, `<Double-Button-1>...`
- ▷ combinaison:
  - ◇ `<Control-Alt-Shift-KeyPress-a>`
  - ◇ `<Control-Alt-KeyPress-A>`
  - ◇ ...

Types d' Evénements

22 Types

<KeyPress>,<Key>,<KeyRelease>	action sur une touche clavier
<ButtonPress>,<Button>	action sur un bouton de souris
<Motion>,<ButtonRelease>	action sur un bouton de souris
<<MouseWheel>	action sur la roue crantée de souris
<Activate>, <Deactivate>	activation, désactivation de fenêtres
<Map>,<Unmap>	affichage de fenêtre à l'écran
<Expose>	exposition de zone cachée
	de nouvelle fenêtre
<Enter>,<Leave>	entrée/sortie de pointeur de souris

Types d' Evénements

...cntd	
`<FocusIn>, <FocusOut>`	focus d'événements sur une fenêtre
`<Visibility>`	changement d'état de visibilité
`<Circulate>`	changement d'ordre de fenêtres
`<Reparent>, <Destroy>`	modification de parent, descruction de widget
`<Gravity>`	modification après redimensionnement de parent
`<Colormap>`	modification de la palette de couleurs
`<Property>`	changement de propriétés de widgets
`<Configure>`	modification d'options de configuration

Séquences de Substitution

Exemple de script pour obtenir des informations sur l'événement.

```
bind . <Key> {puts "%A --> nom symbolique: %K, valeur: %N"}
```

Résultat en appuyant sur la touche `*` avec focus sur la fenêtre racine:

```
* --> nom symbolique: asterisk, valeur:42
```

Liste des séquences de substitution:

`%A`	caractère ASCII	`%R`	id. de fenêtre racine
`%B`	largeur de bordure	`%S`	id. de sous-fenêtre
`%D`	incrément de roue crantée	`%T`	type d'événement
`%E`	champ `send_event`	`%W`	nom de widget concerné
`%K`	nom de touche symbolique	`%X`	abscisse/fenêtre racine
`%N`	valeur décimale de touche	`%Y`	ordonnée/fenêtre racine

Séquences de Substitution

`%a`	`above` de `<Configure>`	`%o`	redirection des fenêtres
`%b`	n^o du bouton de souris	`%p`	`place` de `<Circulate>`
`%c`	`count` de `<Expose>`	`%s`	état (`state`) du widget
`%d`	`detail` d'un événement	`%t`	temps
`%f`	`focus` de `<Enter>,<Leave>`	`%w`	largeur du widget
`%h`	hauteur du widget	`%x`	abscisse en pixels
`%k`	code de touche (`keycode`)	`%y`	ordonnée en pixels
`%m`	mode de `<Enter>,<Leave>,` `<FocusIn>,<FocusOut>`	`%%`	le symbole % lui-même
		`%#`	n^o de la dernière requête

Evénements Multiples, Virtuels

Combinaison d'événements, séquence de raccourcis clavier :

```
bind .  <Control-x><Control-s> {SaveFile}
bind .  <Control-x><Control-w> {SaveFileAs}
```

...

Déactivation d'événements existant par la commande `break`:

```
bind .myTextWidget <Control-b> {break}
```

Evénement Virtuel:

- ▷ expression pour regrouper une séquence d'événements
- ▷ notation entre double crochets angulaire `<<my_virtual_event>>`

Définition d'événement virtuel par la commande `event`:

```
event add <<Sauvegarder>> <Control-x><Control-s>
```

Utilisation d'un événement virtuel:

```
bind .myTextWidget <<Sauvegarder>> {SaveFile}
```

Evénements Multiples, Virtuels

Exemple d'utilisation pour application multi-plate-forme:

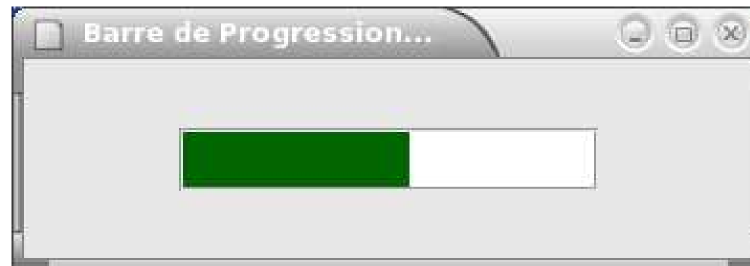
```
switch $tcl_platform (platform) {
  unix {
    event add <<Sauvegarder>> <Control-x><Control-s>
  }
  windows {
    event add  <<Sauvegarder>> <Shift-F12>
  }
  macintosh {
    ...
  }
}
```

`tcl_platform`: variable globale de Tcl de type tableau associatif

Interception d'Evénements

```
wm title . "Barre de Progression ..."
frame .cadre -width 200 -height 30 \
             -borderwidth 2 -relief groove -bg white
frame .barre  -height 26 -bg DarkGreen
place .cadre -anchor center -relx 0.5 -rely 0.5

for {set cpt 0} {$cpt <= 100} { incr cpt 5} {
  after 100
  place .barre -in .cadre -x 0 -y 0 -relwidth [expr $cpt/100.0]
  update idletasks
}
```



Interception d'Evénements

Contrôle du flux d'événements sur l'application

▷ `update`:

◇ forcer l'exécution des événements de la file d'attente

▷ `update idletasks`:

◇ forcer l'exécution des tâches de repos de la file d'attente

▷ `grab`:

◇ restreindre les événements sur un widget

▷ `focus`:

◇ rediriger les événements sur un widget

▷ `tkwait`:

◇ blocage jusqu'à modification d'une variable :

```
tkwait variable une_variable_a_modifier
```

utile pour les boîtes de dialogue modales

Interception d'Evénements

```
wm title . "Fenêtre de Saisie"
label .labelNom -text "Nom"
entry .nom -width 25 -textvariable nom
label .labelPrenom -text "Prénom"
entry .prenom -width 25 -textvariable prenom
grid .labelPrenom .nom  -row 0 -pady 4 -sticky e
grid .labelNom .prenom  -row 1 -pady 4 -sticky e

focus .nom
bind .nom <KeyPress-Return> {focus .prenom}
```

Interception d'Evénements

```
wm title . "Fenêtre Modale"

label .labelSS -text "Numéro de Sécurité Sociale"
entry .entrySS -width 25 -textvariable numero
pack  .labelSS .entrySS
focus .entrySS
bind .entrySS <KeyPress> {
    set saisie "${numero}%A"
    if {![regexp {^(1|2)\d{0,12}$} $saisie]} {
      bell
      invalidWarningDialogBox
      destroy .invalidWarning
      break
    }
}
```

Interception d'Evénements

```
proc invalidWarningDialogBox {} {
  set userOK 0
  set oldFocus [focus]
  catch {destroy .invalidWarning}
  set invalid [toplevel .invalidWarning]
  message $invalid.msg -aspect 500 -justify center\
                     -text "Ce n'est pas un numéro de SS valide"
  button $invalid.okButton -text OK -command {set userOK 1}
  pack  $invalid.msg  $invalid.okButton -pady 4
  grab  $invalid
  focus   $invalid
  tkwait variable userOK
  grab release  $invalid
  focus $oldFocus
}
```

Interception d'Evénements

Vérification que l'utilisateur rentre des chiffres:

```
if {![regexp {^(1|2)\d{0,12}$} $saisie]} {...}
```

Si non, affichage de la boîte de dialogue

▷ `invalidWarningDialogBox`

Validation de l'utilisateur:

▷ `button $invalid.okButton -text OK -command {set userOK 1}`

Attente de validation

▷ `tkwait variable userOK`

Ressources de Widgets

Gestion des ressources, options de widgets

▷ fixer des options aux widgets:

◇ `button .btn -width 100`

▷ fixer des valeurs à des ressources:

◇ `option add *btn.width 100`

▷ récupérer des ressources de widgets:

◇ `option get .btn background Color`

Stockage de ressources

▷ dans des fichiers (`myOptionsFile.opt`):

◇ `*btn.background:  yellow`

▷ chargés dans l'application

◇ `option readfile myOptionsFile.opt`

Extensions/Intégration

Extension:

▷ pour ajouter des commandes dans d'autres langages (`[Incr Tcl]`)

Intégration:

▷ pour incorporer Tcl dans une application (`tclsh,wish`)

Tcl/Tk:

▷ ni plus ni moins qu'une bibliothèque de fonctions C

▷ l'interpréteur évalue et exécute des commandes écrites en C

▷ regroupées dans des bibliothèques partagées

▷ constituant des extensions, modules, packages

Chargement de modules dans l'interpréteur

▷ commande `load`

API Tcl

▷ pour définir et manipuler les objets de base

◇ `variables, listes, tableaux...`

Extension TEA

Tcl **E**xtension **A**rchitecture

- ▷ du code C compilé (`*.o`)
- ▷ mis en bibliothèque partagée (`extensionNameLib.so`)

Conditions de définition de module, bibliothèque partagée

- ▷ une fonction d'initialisation (`extensionName_Init`)
- ▷ nom et numéro de version associé à la fonction d'initialisation

Modèle d'extension sécurisé:

- ▷ fonction d'initialisation (`extensionName_SafeInit`)

Exemple de création et utilisation d'extension:

```
{logname@hostname} gcc -fpic -c addition.c -I/usr/local/include
{logname@hostname} gcc -shared addition.o -o mathLib.so
{logname@hostname} tclsh
% load ./mathLib.so
% addition 1 1
2
```

Extension TEA

Structures et fonctions de l'API d'extension Tcl:

- ▷ `Tcl_Interp`: pointeur sur l'interpréteur
- ▷ `Tcl_Obj`: représentation des objets Tcl
- ▷ `Tcl_PkgProvide`: déclaration de module
- ▷ `Tcl_CreateObjCommand`: enregistrement de nouvelle commande
- ▷ `Tcl_WrongNumArgs`: vérification de la bonne utilisation de commande
- ▷ `Tcl_GetIntFromObj`: pour associer un type C aux `Tcl_Obj`
- ▷ `Tcl_GetObjResult`: pour demander à l'interpréteur un pointeur (`Tcl_Obj`)
- ▷ `Tcl_SetIntFromObj`: pour associer le résultat au pointeur (`Tcl_Obj`)

Extension TEA

```
#include <tcl.h>
int Math_Init (Tcl_Interp *interp );
int AdditionObjCmd ( ClientData client, Tcl_Interp *interp,
                     int objc, Tcl_Obj *CONST objv[]);
int Math_Init (Tcl_Interp *interp ) {
  if (Tcl_InitStubs(interp,"8.4",0) == NULL) return TCL_ERROR;
  // module declaration
  Tcl_PkgProvide(interp, "Math", "1.0");
  // command register
  Tcl_CreateObjCommand(interp,"addition",AdditionObjCmd, NULL,NULL);
  return TCL_OK;
}
```

Extension TEA

Fonction d'initialisation

- ▷ pointeur sur l'interpréteur (`Math_Init (Tcl_Interp *interp )`)
- ▷ déclaration du module `Math` et de sa version (`Tcl_PkgProvide`)
- ▷ enregistrement de la commande `addition` (`Tcl_CreateObjCommand`)

Problème de compatibilité de versions Tcl

- ▷ `Tcl_InitStubs`: au début de l'initialisation
- ▷ `-DUSE_TCL_STUBS`: option de compilation à rajouter

```
gcc -fpic -c -DUSE_TCL_STUBS addition.c \
                       -I/usr/local/include
gcc -shared addition.o -o mathLib.so
```

Implémentation du code C de la commande `addition`:

- ▷ `int AdditionObjCmd (ClientData, Tcl_Interp,int, Tcl_Obj *CONST)`

Extension TEA

```
int AdditionObjCmd ( ClientData client, Tcl_Interp *interp,
                     int objc, Tcl_Obj *CONST objv[]) {
  Tcl_Obj * resultObj;
  int firstArg, secondArg, result;
  if (objc==1 || objc > 3) {
    Tcl_WrongNumArgs(interp,1,objv,"x y"); return TCL_ERROR;
  }
  if (Tcl_GetIntFromObj(interp, objv[1],&firstArg) != TCL_OK)
    return TCL_ERROR;                   // get command first arg value
  if (Tcl_GetIntFromObj(interp, objv[1],&secondArg)!= TCL_OK)
    return TCL_ERROR;                   // get command second arg value
  result = firstArg + secondArg;        // C instruction for result
  resultObj = Tcl_GetObjResult (interp); // get pointer to object command
  Tcl_SetIntObj( resultObj, result);    //  set result in object command
  return TCL_OK;
}
```

Extension TEA

Wrapper SWIG:

Simplified **W**rapper and **I**nterface **G**enerator

Pour générer du code conforme au modèle TEA

▷ définition des fonctions, variables à transformer en Tcl (`*.i`)

▷ génération de code C correspondant à l'API Tcl (`swig -tcl *.i`)

SWIG peut générer des extensions pour d'autre langages:

▷ `Python, Perl, Java, Ruby, Scheme, PHP ...`

Fichier d'interface (`puissance.i`) :

```
%module Puissance
%{
#include <math.h>
%}
long puissance(long, long);
```

SWIG

Fichier source (`puissance.c`):

```
#include <math.h>
long puissance (long x, long y)
{
  return pow(x,y);
}
```

Génération de code, compilation et édition de liens

```
swig -tcl  puissance.i
gcc -fpic -c puissance.c  puissance_wrap.c -I/usr/include
gcc -shared   puissance.o  puissance_wrap.o -o puissance.so
tclsh
% load ./puissance.so Puissance
% puissance 2 3
8
```

Modèle d'Intégration

Incorporer Tcl/Tk dans une application C:

▷ `Tcl_Main`, `Tk_Main`

Interpréter des commandes, script Tcl dans une application:

▷ `Tcl_Eval`, `Tcl_EvalFile`, `Tcl_VarEval`, `Tcl_GlobalEval`..

```
int main(int argc, char* argv[]) {
  // init C application
  ...
  Tcl_Main(argc,argv, initialisationTcl);  //init Tcl
}
int initialisationTcl(Tcl_Interp* interp) {
  if (Tcl_init(interp) == TCL_ERROR) return TCL_ERROR;
  Tcl_CreateObjCommand(...);
  Tcl_SetVar(...);
  ...
}
```

Canaux de Communication

Entre applications Tk:

▷ commande **send**, uniquement sous Unix

▷ `send autreApplication.tcl commmandeAutreApplication $arg1 $arg2 ...`

Informations sur les applications Tk en cours

▷ `winfo interps`

Exemple de communication entre deux applications

▷ un script Tcl/Tk pour créer une jauge **scale.tcl**

▷ un script Tcl/Tk pour créer une spinbox **spinbox.tcl**

```
# spinbox.tcl
wm title . Spinbox
set x_spbox 0

proc spbox2scale { val } {
  send scale.tcl updateScale $val
}
```

Entre applications Tk

```
# spinbox.tcl cntd ...

spinbox .spbox -from -10 -to 10 \
               -incr 1 -textvariable x_spbox \
               -justify center -width 5 \
               -command { spbox2scale $x_spbox }

pack .spbox -anchor center

proc updateSpinbox { val } {
  global x_spbox
  set x_spbox $val
}
```

Entre applications Tk

```
# scale.tcl
wm title . Scale
set x_scale 0

proc scale2spbox { val } {
  send spinbox.tcl updateSpinbox $val
}
scale .scale -from -10 -to 10 \
             -orient horizontal -variable x_scale \
             -command { scale2spbox}
place .scale -relx 0.5 -rely 0.5 -anchor center

proc updateScale { val } {
global x_scale
set x_scale $val
}
```

Entre applications Tk

Problème de sécurisation d' exécution de scripts:

▷ lancement de processus externes "dangereux" (`[exec rm -rf *]` !)

Création d'un interpréteur sécurisé et alias de commandes:

▷ `set securit [interp create -safe]`

▷ `$securit alias send send`

Intercepter les erreurs:

▷ `if {[catch {$securit eval $demande} resultError]} {...}`

```
proc helpCommand  {} {
  global ça
  set demande "send help.tcl faitQuelqueChoseAvec $ça"
    if {[catch {$securit eval $demande} resultError]} {
      leretour ""
      puts resultError
    }
}
```

Entre applications Tcl

Pour échanger et faire circuler des données

- ▷ exécution de processus externes (commande `exec`)
  - ◇ communication synchrone
  - ◇ exécution de programmes "courts"
  - ◇ interception des erreurs d'exécution
- ▷ communication entre processus (commande `open`)
  - ◇ communication synchrone, asynchrone
  - ◇ si temps d'exécution "long", échange de données importantes
- ▷ communication réseau (commande `socket`)
  - ◇ modèle client/serveur

Processus Externes

Demande d'exécution d'un processus externe:

▷ appel de la commande `exec` dans un script Tcl

▷ commande assimilable à l'utilisation des shells Unix

Exemple d'uitlisation pour:

▷ lister (`ls`) les fichiers

▷ d'un répertoire (`dirName`)

▷ avec éventuellement des options (`options`)

```
proc directoryListing {dirName {options ""} } {
    set currentDirectory [pwd]
    cd $dirName
    set result [exec ls $options]
    cd $currentDirectory
    return $result
}
```

Processus Externes

Interception d'erreur

▷ commande `catch`

```
% catch {exec cat toto} reponse
% puts $reponse
cat: toto: No such file or directory
```

Exécution de programme "long":

▷ problème de blocage jusqu'à fin d'exécution des processus

▷ utilisation de la commande `open`

Communication entre Processus

Comme pour une simple commande d'ouverture de fichier

▷ `open`: lecture/écriture (sens de la communication)

Associée à la commande

▷ `fileevent` : contrôle des échanges de données

Qui permet de lier une procédure à un événement

▷ équivalent de la commande `bind` de Tk

Exemple de lecture asynchrone de données:

```
proc readLatexAsync {file} {
    global waiting
    set fid [open "| latex $file" "r"]
    fconfigure $fid -buffering line
    fileevent $fid readable "writeInformation $fid"
    vwait waiting
}
```

Communication entre Processus

Le programme précédent permet

- ▷ d'exécuter une commande externe (`latex`)
- ▷ de manière asynchrone (`open` au lieu de `exec`)
- ▷ par une connexion en lecture (`open "| latex $file" "r"`)
- ▷ en réceptionnant les données ligne à ligne (`fconfigure -buffering line`)
- ▷ pour déclencher un traitement (`writeInformation`)
- ▷ dès qu'une ligne est lue sur le canal d'entrée (`fileevent $fid readable`)
- ▷ en se mettant en attente d'événements (`vwait`)
- ▷ jusqu'à la modification d'une varaible (`waiting`)

Liaison arrivé/traitement de données

- ▷ `fileevent $fid readable "writeInformation $fid"`

La liaison de l'événement peut se faire dans les deux sens

- ▷ envoi,reception de données (`writable,readable`)

Communication entre Processus

Traitement des données

- ▷ appel de procédure (`writeInformation`)
- ▷ jusqu'à la fin de fichier (`[eof $id]`)
- ▷ en mettant fin à la boucle d'attente (`set vwait 1`)

```
set lineNumber 1
proc writeInformation {id} {
  global lineNumber waiting
    if {[eof $id]} {
      set waiting 1
      close $id
      return
    }
  gets $id ligne
  puts "$lineNumber- $ligne"
  incr lineNumber
}
```

Communication entre Processus

Lecture du Document Latex

```
% readLatexAsync myDocument.tex
1- This is TeX, Version 3.14159 (Web2C 7.4.5)
2- (./myDocument.tex
3- LaTeX2e <2001/06/01>
4- Babel <v3.7h> and hyphenation patterns for american, french, german, ngerman, b
5- asque, italian, portuges, russian, spanish, nohyphenation, loaded.
6- (/usr/share/texmf/tex/latex/base/article.cls
7- Document Class: article 2001/04/21 v1.4e Standard LaTeX document class
8- (/usr/share/texmf/tex/latex/base/size10.clo)) (./myDocument.aux) [1]
9- (./myDocument.aux) )
10- Output written on myDocument.dvi (1 page, 276 bytes).
11- Transcript written on myDocument.log.
12-
```

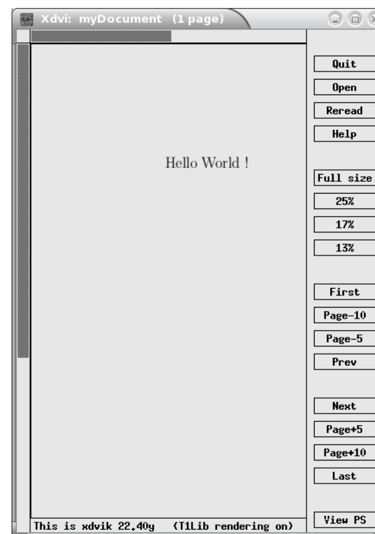
Communication entre Processus

Avec comme exemple de document Latex:

```
\documentclass {article}
\begin{document}
{\LARGE Hello World !}
\end{document}
```

Visualisation du résultat

```
% catch {exec xdvi myDocument.dvi}
```



Communication Réseau

Connexion réseau en mode Client/Serveur

▷ commande `socket`

▷ en mode serveur `socket -server`

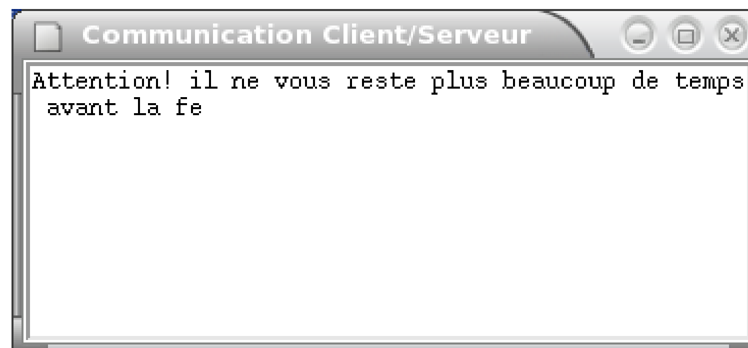
Côté Serveur

▷ `socket -server clientConnection 9000`

Côté Client

▷ `set connexion [socket -async localhost 9000]`

```
{logname@hostname} wish textServer.tcl &
{logname@hostname} wish textClient.tcl &
```



Communication Réseau

```
wm title . "Communication Client/Serveur"
socket -server clientConnection 9000
proc clientConnection { sockId clientAdr clientPort } {
    fconfigure $sockId -blocking 0
    fileevent  $sockId readable [list receptionInput $sockId]
}
proc receptionInput {sockId} {
    if {[eof $sockId]} {
        fileevent  $sockId readable {}
        close $sockId
        after 3000 exit
    }
    set data [read $sockId]
    if {[string length $data]>0} {
      .txt insert end $data
    }
}
text .txt -bd 2 -relief sunken -bg white -width 50 -height 10
pack .txt -side top -fill both -expand 1
```

Communication Réseau

Coté Serveur (`serveur.tcl`):

▷ lancement du serveur et initialisation de la connexion

◇ `socket -server clientConnection 9000`

Initialisation de la connexion (`clientConnection`):

▷ connexion en mode non-bloquant du client qui se connecte:

◇ `fconfigure $sockId -blocking 0`

▷ liaison arrivée de données sur le serveur:

◇ `fileevent $sockId readable [list receptionInput $sockId]`

▷ et appel de la procédure de traitement

◇ `receptionInput`

Communication Réseau

Rôle de la procédure de traitement:

- ▷ afficher les données lues
  - ◇ `set data [read $sockId]`
- ▷ dans une IHM Tk
  - ◇ `.txt insert end $data`
- ▷ en cas de fin de fichier
  - ◇ désinstaller la liaison
    - ○ `fileevent $sockId readable`
  - ◇ fermer la connexion
    - ○ `close $sockId`
  - ◇ sortir de l'application
    - ○ `after 3000 exit`

Canaux de Communication

```
set delai 100
set data "Attention! il ne vous reste plus beaucoup de temps\
          avant la fermeture de la connexion"
set data [split $data {}]

set connexion [socket -async localhost 9000]
fconfigure $connexion -buffering none

foreach x $data {
  puts -nonewline $connexion $x
  after $delai
}

close $connexion
exit
```

Canaux de Communication

Côté Client (`client.tcl`):

▷ connexion au serveur en mode asynchrone

◇ `set connexion [socket -async localhost 9000]`

▷ découpage des données lettre par lettre

◇ `set data [split $data {}]`

▷ envoi des données une à une :

◇ `foreach x $data`

▷ sur le canal de communication

◇ `puts -nonewline $connexion $x`

▷ toutes les secondes

◇ `after $delai`

Conclusion

Intérêts de Tcl/Tk

- ▷ ceux des langages interprétés
- ▷ style de programmation très simple
- ▷ manipulation de chaînes de caractères
- ▷ manipulation de listes, tableaux
- ▷ structures de contrôle, procédures
- ▷ communication entre applications
- ▷ palette de composants graphiques (widgets)
- ▷ programmation événementielle
- ▷ extension pour définir de nouvelles commandes
- ▷ intégration dans des langages de programmation
- ▷ création de nouveaux interpréteurs
- ▷ ...

Conclusion

Avantages de Tcl/Tk

- ▷ langage de commandes-outils
- ▷ syntaxe très simple
- ▷ apprentissage facile
- ▷ développement rapide

Bibliographie

Pour plus d'informations:

▷ **Bernard Desgraupes**: "TCL/TK Apprentissage et référence" Ed. Vuibert 2002 (`www.vuibert.fr`)

▷ `http://www.tcl.tk`

▷ `http://freealter.org/doc_distrib/tcltk-8.3.2/TclCourse`