Tcl und Tk - Ein Überblick

Torsten Spindler
Unix-AG, Universität Kaiserslautern
spindler@unix-ag.uni-kl.de

Zusammenfassung

Mit Tcl/Tk lassen sich einfach und schnell kleine Programme erstellen. Im Paper werden zuerst einige wichtige Sprachkonstrukte von Tcl an Hand von Beispielen besprochen. Danach wird Tk vorgestellt.

1 Einleitung

Tcl ist eine interpretierte Skriptsprache, Tk eine Erweiterung von Tcl um grafische Komponenten. Beide sind frei erhältlich und werden von Dr. John Ousterhout entwickelt. Zur Zeit sind die Versionen Tcl 7.6 und Tk 4.2 aktuell, aber es existieren bereits Alpha Versionen von Tcl/Tk 8.0.

Tcl und Tk lassen sich vielseitig verwenden:

- Als eigenständige Skriptsprache können damit kleine Aufgaben erledigt werden.
- Tcl/Tk liegen als C Bibliothek vor und sind erweiterbar. Anwendungen können neue Funktionen in den Interpreter einbauen und Tcl/Tk als Skriptsprache verwenden.
- Unter besonderer Berücksichtigung der grafischen Möglichkeiten von Tk ist die Entwicklung lauffähiger, mit Benutzeroberfläche versehener Prototypen sehr effizient.

In den folgenden beiden Abschnitten werden Tcl und Tk an Hand kleinerer Beispiele vorgestellt.

2 Tcl

Tcl steht für *tool command language*. Diese Bezeichnung deutet schon auf einen Zweck der Sprache hin: Eine Hilfssprache (Skriptsprache) für Anwendungen zu sein. Viele Anwendungsprogramme verwenden eine eigene Sprache um Konfigurationsdateien einzulesen oder dem Benutzer die Möglichkeit zu geben während der Laufzeit selbstgeschriebene Skripte oder Makros ablaufen zu lassen. Beispielsweise kann das Verhalten von ftp durch ein .netrc bestimmt und Emacs durch in elisp geschriebene Programme erweitert werden. Leider brachte diese Entwicklung auch einige Nachteile mit sich:

- Benutzer erlernen für jede Anwendung eine andere Skriptsprache
- Entwickler entwerfen Skriptsprachen, die sich oft nur unwesentlich voneinander unterscheiden

Durch die Verwendung von Tcl können diese beiden Nachteile im wesentlichen vermieden werden. Die Aufgabe des Entwicklers der Applikation reduziert sich von Entwurf und Implementierung einer eigenen Sprache auf die Schnittstellendefinition und -programmierung zwischen Anwendung und Tcl. Ein Benutzer, der einmal Tcl erlernt hat kann dieses Wissen oft nutzen. Natürlich wird dieser Vorteil durch die Ablehnung von Tcl durch Entwickler (z.B. Gnu) wieder ausgeglichen.

Im folgenden wird die Programmierung mit Tcl vorgestellt. Ziel ist es einige grundlegenden Sprachkonstrukte von Tcl zu zeigen und den Leser in die Lage zu versetzen kleinere Skripte selbst zu schreiben. Bei Fragen zu den verwendeten Befehlen sollte die zugehörige man-page zu Rate gezogen werden.

2.1 Starten einer Tcl Sitzung

Die mit der TCL Distribution mitgelieferte Standardshell wird mit dem Befehl tclsh gestartet. Auf manchen Systemen muß auch die Versionsnummer angehängt werden, so startet z.B. tclsh7.5 Version 7.5 von Tcl.

2.2 Grundlagen

Nachdem sich die Shell initialisiert hat, können Kommandos eingegeben werden. Die allgemeine Form eines Befehls ist *Befehl Argument1 Argument2* . . .. Das Leerzeichen dient hierbei zur Trennung der Argumente. Um auch Leerzeichen innerhalb eines Arguments zu verwenden, muß es quotiert (vor Auswertung geschützt) werden. Hierzu stehen in Tcl drei verschiedene Methoden zur Verfügung:

- Backslash: nur das nächste Zeichen wird quotiert
- Anführungszeichen: alle Leerzeichen werden nicht als Trennzeichen interpretiert
- Geschweifte Klammern: alle Zeichen werden nicht interpretiert

Ein Programm setzt sich aus mehreren Befehlen zusammen, welche entweder durch Semikolon oder Zeilenumbruch getrennt werden.

Da es sich um einen Interpreter handelt wird jeder durch einen Zeilenumbruch abgeschlossene Befehl (oder durch Semikolon getrennte Befehlsfolgen) sofort ausgeführt.

2.3 Variablen und Variablensubstitution

In Tcl existiert nur ein Variablentyp: String. Der Zugriff auf Variablen ist dem innerhalb einer Shell sehr ähnlich: Mit dem *set* Befehl werden Variablen Werte zugeordnet und mit *unset* wieder aufgehoben.

```
spindler@calzone [~] tclsh7.6
% puts "Die Tcl Sitzung wurde soeben gestartet"
Die Tcl Sitzung wurde soeben gestartet
% set wert "Dies ist ein Wert"
Dies ist ein Wert
% # Auslesen des Variablenwerts
% set wert
Dies ist ein Wert
% # Die Zuordnung wird augehoben
% unset wert
% set wert
can't read "wert": no such variable
% # Fehlermeldung
```

Zu beachten ist hierbei die Verwendung von Anführungszeichen bei der Zuweisung von "Dies ist ein Wert" an a. Ohne die Anführungszeichen wären die Leerzeichen in "Dies ist ein Wert" interpretiert worden und *set* mit insgesamt 5 Argumenten aufgerufen worden.

Neben einfachen Variablen können auch Felder verwendet werden. Der Index eines Feldes wird in runden Klammern nach dem Feldnamen angegeben. Mit Hilfe des *array* Befehls kann effektiv auf Felder zugegriffen werden.

```
% set wochentag(montag) "Eis essen"
Eis essen
% set wochentag(dienstag) "Vorlesung besuchen"
Vorlesung besuchen
% set wochentag(mittwoch) "Ausruhen"
Ausruhen
% array names wochentag
montag mittwoch dienstag
```

Um auf den Inhalt einer Variablen zuzugreifen wird der *$* Operator verwendet. Auch hier ist die nahe Verwandschaft zu Shellsprachen sichtbar.

```
% puts $wochentag(mittwoch)
Ausruhen
% set wochentag(mittwoch) $wochentag(dienstag)
Vorlesung besuchen
% puts $wochentag(mittwoch)
Vorlesung besuchen
```

2.4 Mathematik und Kommandosubstitution

Da in Tcl jede Variable den Typ String hat, bedarf es eines speziellen Befehls zur Interpretation der Werte als Zahlen: *expr*. Mit Hilfe dieses Kommandos stehen alle Grundrechenarten, sowie einige mathematische Funktionen zur Verfügung.

```
% expr 5 + 6
11
% expr 3 * 3 + 45
54
% expr abs(-5)
5
% expr log10 (100)
2.0
% expr sqrt (25)
5.0
```

Um das Ergebnis einer Rechnung aufzubewahren und später weiterzuverwenden wird die Kommandosubstitution verwendet.

```
% set ergebnis1 [ expr 5 * 6 ]
30
% set ergebnis2 [ expr $ergebnis1 + 50 ]
80
```

Die eckigen Klammern um einen Befehl bewirken, daß zur Laufzeit das Ergebnis dieses Befehls eingesetzt wird. Dies kann ein beliebiger Tcl Befehl sein, oder eine Folge von Befehlen oder ein Prozeduraufruf. Falls mehrere Befehle substituiert werden, ist zu beachten, daß das Ergebnis des Gesamtausdrucks dem des letzten Befehls entspricht.

```
% set wert1 [ set wert2 "Hallo"; set wert3 "Welt"]
Welt
% set wert1
Welt
```

2.5 Listen

Die einzige in Tcl zur Verfügung stehende Datenstruktur sind Listen. Diese werden üblicherweise als in geschweifte Klammern gesetzte und durch Leerzeichen getrennte Werte dargestellt. Die Schachtelung von Listen ist ebenfalls erlaubt.

Zum Aufbau von Listen stehen zwei Befehle zur Verfügung: *list* und *concat*. *List* interpretiert jedes Argument als Element der neuen Liste und *concat* wertet jedes

Argument als Liste aus.

```
% # Aufbau einer Liste mit einer Unterliste
% list die {Sonne scheint} heute nicht
die {Sonne scheint} heute nicht
% concat die {Sonne scheint} heute nicht
die Sonne scheint heute nicht
```

Das Ergebnis des ersten Kommandos ist eine Liste mit 4 Elementen: die, Sonne scheint, heute, nicht.

Der zweite Befehl mit concat liefert hingegen eine Liste mit 5 Elementen: die, Sonne, scheint, heute, nicht.

Da auch Listen intern als String dargestellt werden, kann auch der set Befehl verwendet werden. Um auf die verschiedenen Positionen einer Liste zuzugreifen wird der *lindex* Befehl verwendet. Im folgenden Beispiel werden auch 3 weitere nützliche Kommandos vorgestellt.

```
% set liste {Dies ist ein regnerischer Tag}
Dies ist ein regnerischer Tag
% lindex $liste 0
Dies
% lindex $liste end
Tag
# Ersetzen der Position 3 durch die Elemente 'sonniger und warmer'
% set liste [lreplace $liste 3 3 sonniger und warmer]
Dies ist ein sonniger und warmer Tag
% lsort $liste
Dies Tag ein ist sonniger und warmer
% lsearch $liste sonni*
3
```

Es stehen noch andere spezielle Befehle zur Listenmanipulierung zur Verfügung, z.B. zum Trennen, Bestimmung der Listenlänge.

2.6 Prozeduren, Kontrollstrukturen und Dateien

Folgendes Beispiel soll die Verwendung von Prozeduren, Kontrollstrukturen und den Zugriff auf Dateien kurz beleuchten. Aus Gründen der Klarheit wird auf die Behandlung von Fehlern verzichtet.

```
# Das Programm liest eine Datei ein und summiert deren Inhalt.
# Der Dateiname wird über die Kommandozeile eingelesen.

# Prozedurvereinbarung
# Name: addLine
# Argumentliste: list
# Prozedurrumpf
proc addLine { list } {
        set sum 0
        # Iteration über alle Elemente in der Liste line
        foreach number $list {
                # Erhoehen der Summe innerhalb dieser Zeile
                incr sum $number
        }
        # Alternative, die spezielle Listenbefehle von Tcl
        # ausnutzt
        # set sum [ expr [join [split $1 " "] +] ]
        # Rückgabe der Zeilensume
        return $sum
}
# Der Dateiname wird als erstes Argument von der Kommandozeile
# übergeben
set filename [lindex $argv 0]
# Wert, der bei Erreichen des Dateiendes übergeben wird
set EOF -1
# Die Gesamtsumme
set endSum 0

# Oeffnen der Datei und setzen eines Filedeskriptoren
# r steht für read
set fd [ open $filename r ]
# solange das Dateiende nicht erreicht wurde
while { [gets $fd line] != $EOF } {
        # generiere Zwischensumme aus der Zeile
        set lineSum [ addLine $line ]
        # Erhoehe die Gesamtsumme
        set endSum [ expr $endSum + $lineSum ]
}
# Schliessen der Eingabedatei
close $fd

# Ausgabe des Ergebnisses
puts "Die Gesamtsumme ist $endSum"
```

Das Skript liest die Eingabedatei *filename* zeilenweise ein und berechnet in der Prozedur *addLine* die Summe aus einer Zeile. Anschliessend wird die Gesamtsumme um die Zeilensumme erhöht. Nach Abarbeitung der Eingabedatei wird das Ergebnis ausgegeben.

Bei der Vereinbarung von Prozeduren ist zu beachten, daß alle Argumente call-by-value übergeben werden. Weiterhin sind in einer Prozedur nur lokale Variablen sichtbar. Diese Einschränkung kann durch die Verwendung von *global, upvar* und *uplevel* teilweise wieder rückgängig gemacht werden. Auf eine Diskussion dieser Konzepte wird jedoch an dieser Stelle verzichtet.

Tcl besitzt außer der *foreach* Schleife noch *for* und *while* Schleifen. Zur weitern Strukturierung des Kontrollflußes stehen auch bedingte Anweisung (*if*) und die Mehrfachauswahl (*switch*) zur Verfügung.

Auch die Bearbeitung von Dateien erstreckt sich nicht nur auf das einfache Öffnen und Schließen. Zusätzlich stehen Befehle zur Abfrage des Dateistatus (lesbar, schreibbar, letzter Zugriff, etc.).

2.7 Ausblick

Über die bereits vorgestellten Sprachkonstrukte hinaus bietet Tcl noch einiges mehr. Besonders zu erwähnen sind die Ausnahmebehandlung, die Stringverarbeitung, die Möglichkeit externe Programme zu starten und den Zustand des Tcl Interpreters zu überwachen. Auch bieten die bereits vorgestellten Befehle noch mehr Optionen die in den Man Pages nachgelesen werden können.

3 Tk

Im folgenden Abschnitt wird Tk vorgestellt, die graphische Erweiterung zu Tcl. Neben Tcl wird Tk auch als Erweiterung für eine Reihe weiterer Sprachen angeboten (Scheme, perl, etc). Im folgenden wird aber Tcl die Grundlage sein.

Zur Erzeugung grafischer Benutzeroberflächen bedient sich Tk folgender Mittel:

- Widgets
- Geometry Manager

Tk stellt *widgets* zur Verfügung, die mit Hilfe von *geometry manager* dargestellt und verwaltet werden können. Als Widget werden im folgenden die Elemente einer grafischen Benutzeroberfläche verstanden. Daß sind z.B. Knöpfe, Eingabefelder und Rahmen.

3.1 Starten einer Tcl/Tk Sitzung

Tcl/Tk Skripte werden in dem Interpreter *wish* ausgeführt. Auch hier kann das Programm auch mit angehängter Versionsnummer aufgerufen werden.

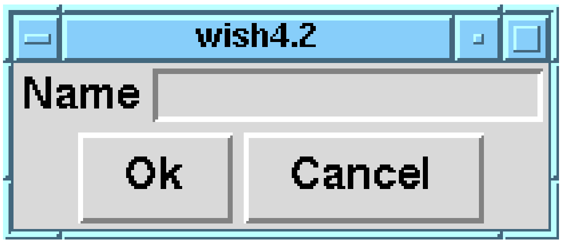
3.2 Ein einfaches Beispiel

Im folgenden wird ein kleines Skript zur Eingabe eines Namens vorgestellt.

Die in einem Programm erzeugten Widgets werden in Tk in einer Baumstruktur abgelegt. Wurzel dieses Baums ist das Widget mit der Bezeichnung ".". Die Nachfolger von . heissen .Bezeichnung, wobei Bezeichnung keinen Punkt enthalten darf. Weitere Nachfolger werden mit .Bezeichnung.Bezeichnung als Namen versehen. Nachfolger eines Widgets sind nur in ihrem Vorfahr Widget sichtbar, außer es handelt sich um ein toplevel Widget. Diese unterstehen direkt dem Window Manager.

Im Beispiel besteht der Widget Baum aus . als Wurzel und den Nachfolgern .controls .entries .name .eingabe-name .ok und .cancel.

```
# Programm zum Einlesen eines Namen

# Funktionalität
proc quit {} {
        # Ausgabe des Ergebnisses
        puts "Your name is $name."
        # Beenden
        destroy .
}

proc cancel {} {
        # Ausgabe des Ergebnisses
        puts "No name."
        # Beenden
        destroy .
}

# Zuerst werden die benoetigten Widgets erzeugt

# Zwei Rahmen zur Strukturierung des Layouts
frame .controls
frame .entries
# Name des Eingabefeldes
label .name -text Name
# Das Eingabefeld
# In der Variablen name wird der Wert des Eingabefelds
# gespeichert
entry .eingabe-name -textvariable name
# Zwei Knoepfe zum Steuern der Anwendung
button .ok \
        -text Ok \
        -command quit
button .cancel \
        -text Cancel \
        -command cancel

# Darstellung der Widgets

pack .ok .cancel -in .controls -side left
pack .name .eingabe-name -in .entries -side left
pack .controls .entries -side bottom
```

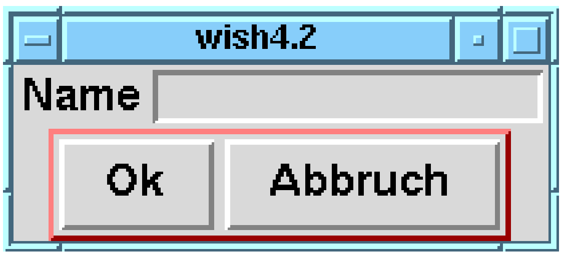


Die wesentlichen Bestandteile eines Tcl/Tk Skriptes sind schon bei diesem einfachen Beispiel vorhanden:

- Erzeugung und Konfiguration der Widgets
- Darstellung der Widgets

Der Befehl *frame .controls* erzeugt ein Widget vom Typ *frame* mit dem Namen *.controls*. Nach Abarbeitung dieses Befehls existiert im Interpreter ein neuer Befehl: *.controls*. Dieses neue Kommando stellt eine neue Funktionalität zur Verfügung, die je nach Widget Klasse unterschiedlich ist. Allen gemein ist die Möglichkeit der Neukonfiguration des Widgets.

```
# Der Rahmen wird sichtbar
.controls configure \
         -borderwidth 2 \
         -relief raised \
         -background red
# Eindeutschen eines Knopfes
.cancel configure -text Abbruch
```



Nach der Erzeugung der verschiedenen grafischen Komponenten müßen diese noch dargestellt werden. Dazu werden in Tk *geometry manager* verwendet. Im wesentlichen existieren zwei verschiedene: Packer und Placer.

Beim Placer muß genau angegeben werden wie breit und hoch ein widget entweder absolut oder relativ zu einem anderen ist. Der Packer hingegen arrangiert die Widgets so, daß der zur Verfügung stehende Platz ausgenutzt wird. Im Beispiel werden die Widgets *.ok* und *.cancel* in den Rahmen *.controls* hineingepackt. Die Rahmen *.controls* und *.entries* werden wiederum in ihr Elternwidget *.* gepackt. Durch die Verwendung der *-in* Option wurde die Darstellungshierarchie von der internen Struktur im Widget Baum getrennt.

Der Kontrollfluß wird momentan bei der Erzeugung der Widgets durch die -*command* Option festgelegt und ist minimal.

3.3 Weitere Widgets

Neben den bereits vorgestellten Widgettypen *frame, label, button* und *entry* gibt es noch eine Vielzahl weiterer:

- *checkbutton* - Ein Knopf, der zwei Zustände hat
- *radiobutton* - Mehrere Knöpfe, von denen nur einer gedrückt sein kann
- *menu* - Verschiedene Arten von Menüs (pop-up, pull-down, cascade)
- *listbox* - Auswahlbox
- *text* - Darstellen und Einlesen von Text
- *message* - Nachrichtenbox (einfacher als *text*)
- *canvas* - Darstellen von Grafik
- *scrollbar* - kontrolliert den Inhalt anderer Widgets
- *scale* - Einstellung von Werten durch Schieberegler

3.4 Eventverarbeitung

Bei einer Vielzahl von Widgets ist bereits eine Eventverarbeitung vorhanden. So kann man bei einem *entry* Widget Text eingeben, nachdem es den Fokus erhalten hat. Ein Knopf reagiert auf einen Druck mit der linken Maustaste mit einem bestimmten Verhalten. Mit Hilfe des *bind* Befehls kann der Programmierer selbst festlegen auf welche Ereignisse ein widget reagieren soll.

```
# Generiere einen Knopf
label .bunt \
        -text "Ein gelber Knopf" \
        -bg yellow
# Für label sind keine Ereignisse vordefiniert
# Bei Drücken der Maustaste 1 wird der Hintergrund blau
bind .bunt <1> { \
                .bunt configure \
                        -bg red \
                        -text "Ein roter Knopf" }
# Bei Drücken der Maustaste 3 wird der Hintergrund blau
bind .bunt <3> { \
                .bunt configure  \
                        -bg blue \
                        -text "Ein blauer Knopf" }
```

Im Beispiel wird ein *label* widget modifiziert. Der erste Aufruf von *bind* bewirkt, daß bei jedem Druck der linken Maustaste der Hintergrund rot wird, der zweite Aufruf läßt den Hintergrund bei Druck der rechten Maustaste blau erscheinen.

3.5 Ausblick

Tk verfügt auch über Methoden um die Selektion und den Fokus zu manipulieren. Weiterhin stehen Funktionen zur Interaktion mit dem Window Manager zur Verfügung. Letztendlich steht mit der *send* Primitive noch ein Möglichkeit zur Verfügung mehrere Tk Skripte, die auf einem Bildschirm laufen, miteinander zu verknüpfen.

Einen guten Einblick in die Möglichkeiten von Tk geben die in der Tk Distribution vorhandenen Demo Skripte, welche in lib/tk[Versionsnummer]/demos zu finden sind.