Die Tcl-Bibliothek Tcllib

Arbeitsmittel

In der Tcllib sammeln Entwickler praktische Werkzeuge, die typische Programmieraufgaben lösen oder ganze Netzwerkprotokolle implementieren. Da sie sich auf eine reine Tcl-Erweiterung ohne kompilierten Code beschränkt, ist die Bibliothek sofort einsatzbereit. Dieser Streifzug stellt einige Teile vor. Carsten Zerbst

Die Tcllib ist selbst unter Tcl-Entwicklern noch recht unbekannt – schade, denn sie hat eine Menge zu bieten. Von Datenstrukturen über Mathematik bis hin zu Netzwerkprotokollen enthält sie knapp 100 Tcl-Pakete. Viele Linux-Distributionen liefern die Tcllib bereits mit, sie ist aber auch bei Sourceforge [1] erhältlich, aktuell in Version 1.6.1. Linux-Programmierer laden am besten das Tar-Archiv. Nach dem Auspacken muss Root die Bibliothek mit dem mitgelieferten grafischen Installer aufspielen: »wish installer.tcl«.

Alles drin

Neben den Bibliotheken enthält das Archiv auch Beispielanwendungen sowie die Dokumentation im Manpage- und HTML-Format. Wer vor der Installation einen Blick auf die Dokumentation werfen möchte, findet sie auf der Sourceforge-Seite unter [2].

Die Tcllib hat mit der Tklib einen kleinen Bruder, der GUI-Funktionen sammelt. Wer diese Library ebenfalls nutzen möchte, muss sie derzeit noch aus dem CVS-Repository holen. Auch dies ist mit wenigen Schritten erledigt:

```
cvs -d:pserver:anonymous@cvs.sourceforge.➥
net:/cvsroot/tcllib login
cvs -z3 -d:pserver:anonymous@cvs.sourcefo➥
rge.net:/cvsroot/tcllib co tklib
cd tklib
wish installer.tcl
```

Für den anonymen CVS-Zugang ist kein Passwort notwendig, ein einfaches Return genügt als Reaktion auf die Nachfrage »CVS password:«.

Eine EDV-Weisheit sagt: Computer helfen uns Probleme zu lösen, die wir ohne sie nicht hätten. Auch beim Programmieren wird das deutlich. Immer wieder müssen Entwickler Aufgaben bewältigen, die wenig mit dem eigentlichen Zweck ihrer Software zu tun haben. Zum Beispiel die typische Frage, was die Anwendung gerade tut – beim Debugging die wichtigste Erkenntnis.

Der einfachste Ansatz sind »puts stdout *Meldung*«-Anweisungen an allen kritischen Stellen. Leider muss man diese später löschen oder auskommentieren. Einfacher und eleganter geht es mit dem Logger-Paket. Mit ihm definiert der Entwickler Logkanäle, in die das Programm Nachrichten mit einem Loglevel von »debug« bis »critical« schreibt. Dank des Loglevels genügt ein Parameter, um die Ausgabe von unerwünschten Debug-Informationen auf ein Anwender-verträgliches Maß zu reduzieren. Logger kennt mehrere Ausgabeformate und schreibt auf Wunsch direkt in eine Datei. Neben dem reinen Beobachten der Anwendung ist die Dauer einzelner Aufrufe oft interessant. Hier hilft das Profiler-Paket, es speichert für alle Prozeduren die Anzahl der Aufrufe, die dazu benötigte Zeit und weitere Timing-Informationen.

Das Skript in Listing 1 verwendet beide Pakete. In den Zeilen 5 und 6 fordert es sie mit »package require« an. Den Profiler muss das Programm vor allen weiteren Aufrufen initialisieren, das erledigt »::profiler::init« in Zeile 11. Statt einer kompletten Anwendung enthält das Beispiel nur einen Namensraum mit zwei Prozeduren »A« und »B« sowie den Logkanal »$log«. Diesen legt es per »logger::init *Name*« an.

Logging und Profiling

Da das Logger-Paket seine Funktionen in einem Namensraum versammelt, muss die Kanalvariable in geschweiften Klammern stehen. Zeile 16 setzt den Loglevel auf »info« und unterdrückt damit alle »debug«-Meldungen. Erst nach dem dritten Aufruf setzt ihn Zeile 26 auf »debug« herab. Ab diesem Zeitpunkt sind alle Meldungen in der Ausgabe zu sehen. Mit einem eigenen Logger je Prozedur oder je Namensraum ist der Überblick jederzeit gewährleistet.

Das Ergebnis ist in Abbildung 1 zu sehen. Sie zeigt die Logging- und Profiler-

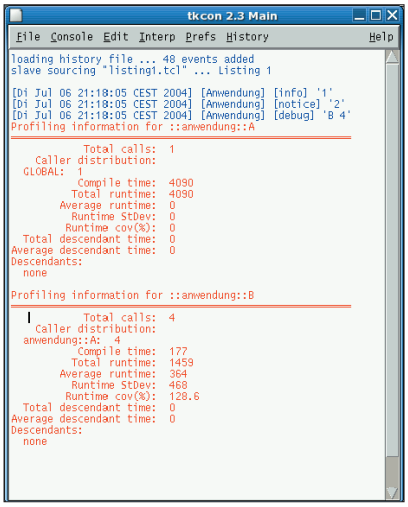

Abbildung 1: Mit dem Logger-Paket sind ausführliche Log-Ausgaben in Tcl einfach erledigt (blau). Auch das Profiling einer Applikation (rot) ist mit dem passenden Modul in wenigen Codezeilen erledigt (Listing 1).

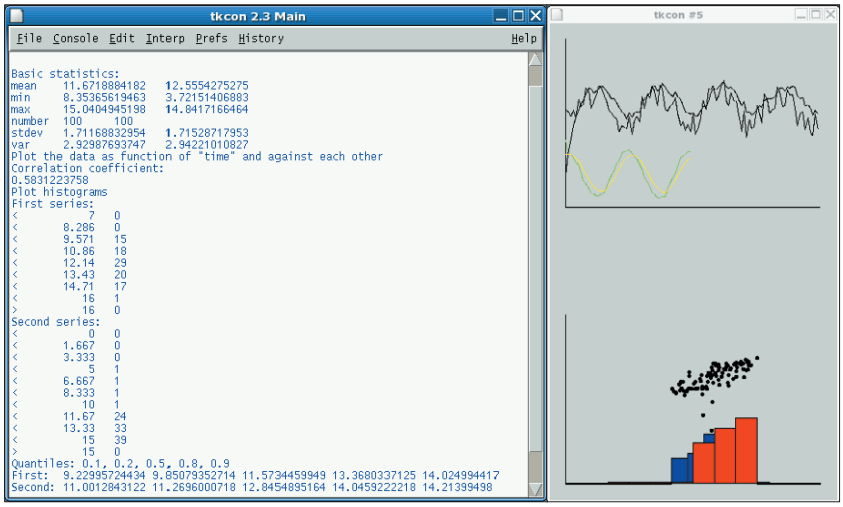

Abbildung 2: Das Statistikpaket gibt harte Zahlen (links) oder bunte Grafiken (rechts) aus. Die auszuwertenden Daten stammen in diesem Beispiel von einem einfachen Zufallszahlengenerator.

Ausgaben von Listing 1. Mit »::profiler::print« gibt Zeile 37 Informationen über die Prozeduren aus. Neben der Anzahl der Aufrufe und ihrer Herkunft sind mehrere Zeitinformationen enthalten. Auf der Suche nach Performance-Killern sind die hauptschuldigen Programmteile damit schnell entlarvt.

Messkurven

Wer viel misst, muss seine Ergebnisse auch auswerten. In der Mathematikabteilung der Tcllib findet sich hierfür das Statistikpaket »::math::statistics«. Es erwartet die Daten in Form einfacher Listen. Wer nur die üblichen Mittelwerte und die Standardabweichung benötigt, wird ebenso fündig wie jene, die auf der Suche nach Korrelationen oder einem Histogramm sind. Für den schnellen Überblick über die Daten sind sogar Diagrammfunktionen enthalten.

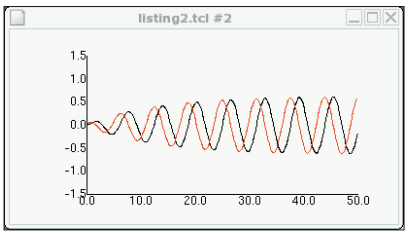

Abbildung 3: Auslenkung und Geschwindigkeit (y-Achse) eines gedämpften Schwingers im Laufe der Zeit (x-Achse). Die Werte wurden mit dem Calculus-Paket aus der Tcllib berechnet und mit Plotchart aus der Tklib dargestellt.

In der Dokumentation des Moduls findet sich ein ausführliches Beispiel. Dessen Ergebnisse sind in Abbildung 2 zu sehen. Das Programm benutzt zwei Zeitreihen, für die es typische Statistikwerte berechnet. Die Diagramme stellen den Zeitverlauf, die Korrelation sowie die Häufigkeitsverteilung dar.

Neben dem für Statistik findet sich mit »::math::calculus« ein weiteres interessantes Mathematikpaket in der Tcllib. Es enthält numerische Integrationsverfahren wie Simpson für Flächen oder Runge-Kutta für lineare Differenzialgleichungen, dazu noch Matrizenberechnungen wie die Lösung von linearen Gleichungssystemen oder die Bestimmung des Eigenwerts.

Dieses Paket ersetzt zwar keine ausgewachsenen Mathematikprogramme wie Matlab oder das freie Scilab [3], für kleinere Aufgaben genügt Calculus jedoch vollkommen.

Schwungvoller Schwinger

Das Beispiel in Listing 2 berechnet den Einschwingvorgang eines gedämpften Schwingers mit Hilfe von Runge-Kutta (Zeile 25). Als Eingabe für die Integration dient eine Funktion, die den jeweils aktuellen Zustandsvektor bestimmt, hier die Tcl-Prozedur »schwinger« ab Zeile 17. Die Schleife in Zeile 24 integriert schrittweise und schreibt Zeit, Position und Geschwindigkeit in die Listen »abzisse«, »xordinate« und »vordinate«.

Der zweite Teil des Programms verwendet das Plotchart-Paket (Zeile 39) aus der Tklib, um ein x-y-Diagramm zu

Listing 1: Logging und Profiling

```
01 #!/bin/sh
02 # Beispiel für Profiler und Logger \
03 exec tclsh $0 $@
04
05 package require logger
06 package require profiler
07
08 puts stdout "Listing 1\n"
09
10 # Initialisierung
11 profiler::init
12
13 namespace eval anwendung {
14   variable log
15   set log [logger::init Anwendung]
16   ${log}::setlevel info
17
18   proc A {} {
19     variable log
20     ${log}::info "1"
21     B 1
22     ${log}::notice "2"
23     B 2
24     ${log}::debug "erscheint nicht"
25     B 3
26     ${log}::setlevel debug
27     B 4
28   }
29
30   proc B {i} {
31     variable log
32     ${log}::debug "B $i"
33   }
34 }
35
36 anwendung::A
37 puts stderr [::profiler::print ::anwendung::*]
```

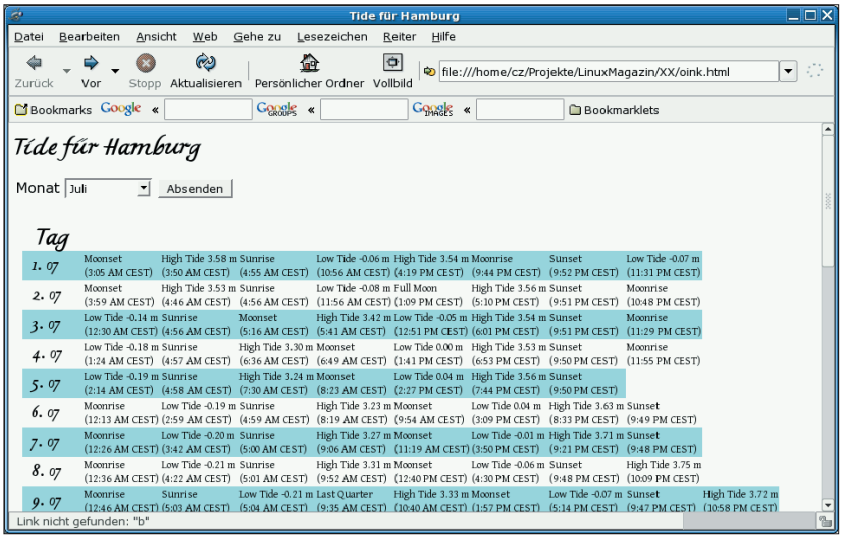

Abbildung 4: Das CGI-Skript in Listing 3 gibt Gezeiteninformationen zum gewünschten Monat als HTML-Seite aus. Die Daten dazu stammen aus CSV-Dateien.

zeichnen. Dieses Paket benötigt ein Tk-Canvas-Widget (Zeile 41). Ab Zeile 44 erzeugt das Programm ein »XYPlot«-Diagramm, es benötigt dazu neben dem Canvas die Minimal- und Maximalwerte sowie den Abstand zwischen zwei Markierungen für die x- und für die y-Achse. Die Schleife ab Zeile 48 übergibt dann die einzelnen Datenpunkte mit dem »plot«-Befehl an das Diagramm. Das Ergebnis ist in Abbildung 3 zu sehen. Das Plotchart-Paket enthält neben den x-y-Diagrammen noch Balken-, Torten-, Polar- und 3D-Diagramme.

Feder im Netz

Tcl spricht mit dem »socket«-Kommando bereits TCP/IP. Für die Protokolle und Paketstrukturen der höheren Schichten sind Tcllib-Pakete zuständig: Zum Beispiel FTP (Dateitransfer), CVS (Versionierung), NNTP (Zeitabfrage), POP3 (E-Mail abholen) oder SMTP (Mail versenden) sowie verbreitete Prüfsummen- und Verschlüsselungsalgorithmen wie MD5, SHA1 oder DES.

Interaktive Webseiten

Um HTML-Seiten bequem per Tcl-Code zu erzeugen, liefert Tcllib drei Pakete mit: »ncgi«, »html« und »javascript«. Mit ihrer Hilfe kann ein Programm Benutzereingaben auswerten oder HTML-Seiten mit Javascript erzeugen. Ein typisches Beispiel ist in Listing 3 zu sehen, es fungiert als CGI-Skript. Sinn des Skripts ist es, Tidenwerte wie Hoch- und Niedrigwasser oder Auf- und Untergangszeiten von Sonne und Mond anzuzeigen. Die Werte wurden mit dem freien Programm Tide [4] berechnet und liegen in je einer CSV-Datei (Comma-separated Values) pro Monat vor.

Das Programm besteht im Wesentlichen aus drei Teilen: Die Zeilen 15 bis 20 lesen Benutzereingaben, Zeilen 46 bis 60 werten die CSV-Dateien aus und die Zeilen 25 bis 41 sowie 65 bis 95 erzeugen die HTML-Ausgabe. Wie bei CGIs üblich sind die Teile vermengt, daher gerät das Skript etwas unübersichtlich.

Listing 2: Einfacher Schwinger

```
01 #!/bin/sh
02 # Beispiel für ::math::calculus und Plotchart \
03 exec wish $0 $@
04
05 package require  math::calculus
06
07 # Zustandsvektor xvec enthält x und x' (v)
08 set xvec  {0.0  0.0}
09 set t      0.0
10 set tstep  0.25
11 set steps  200
12
13 set abzisse   {}
14 set xordinate {}
15 set vordinate {}
16
17 proc schwinger {t xvec} {
18   set x  [lindex $xvec 0]
19   set x1 [lindex $xvec 1]
20   return [list $x1 [expr {-0.2*$x1 - $x +
   0.1*cos($t)}]]
21 }
22
23 # Schritte mit Euler berechnen
24 for {set i 0} {$i < $steps} {incr i} {
25   set result [::math::calculus::rungeKuttaStep \
26     $t $tstep $xvec schwinger]
27
28   lappend abzisse $t
29   lappend xordinate [lindex $result 0]
30   lappend vordinate [lindex $result 1]
31
32   set t     [expr {$t+$tstep}]
33   set xvec  $result
34 }
35
36 #
37 # Ausgabe mit Plotchart
38 #
39 package require Plotchart
40
41 canvas .c -background white -width 400 -height 200
42 pack   .c -fill both
43
44 set xyDia [::Plotchart::createXYPlot .c \
45   {0.0 50 10} {-1.5 1.5 0.5}]
46 $xyDia dataconfig v -colour  "red"
47
48 for {set i 0} {$i < [llength $abzisse]} {incr i } {
49   $xyDia plot x [lindex $abzisse $i] [lindex
   $xordinate $i]
50   $xyDia plot v [lindex $abzisse $i] [lindex
   $vordinate $i]
51 }
```

Das Neueste

Die Diskussionen über eine Modernisierung von Tk gehen weiter. Adrian Davis bringt mit Gridplus [5] einen Beitrag für das Layout von Widgets, mit dem das Programmieren ansehnlicher Oberflächen besser gelingt als mit den Tcl-eigenen Layout-Managern. Ein weiteres Ziel ist die einfachere Integration von Tcl-Applikationen in KDE. George Peter Staplin beschreibt in einem Newsgroup-Posting [6] die nötigen Schritte, um Icons ins KDE-Systray einzubinden.

Außer für die GUI-Seite arbeiten die Entwickler auch in den Tiefen von Tcl und den Tcl-Modulen. Zum Beispiel implementiert das Tcldes-Paket [7] die komplette Triple-DES-Verschlüsselung in reinem Tcl-Code. Die Pgtcl-Erweiterung für PostgreSQL [8] wurde in der neuen Version 1.4 an den aktuellen Stand von PostgreSQL angepasst und deutlich beschleunigt. Das im Artikel beschriebene CGI-Skript läuft zwar mit allen Webservern, besonders elegant ist aber die Integration mit den Apache-Tcl-Modulen Websh [9] oder Rivet [10]. Letzteres liegt seit kurzem in Version 0.4 vor.

Das »ncgi«-Paket wertet die Benutzereingaben aus. Es liest sie in Zeile 15 mit »::ncgi::parse« und fragt in Zeile 19 mit »::ncgi::value« den gewünschten Monat ab. Als Grundeinstellung dient der aktuelle Monat, das Jahr ist fest auf 2004 gesetzt (Zeile 18).

Um die HTML-Tags zu erzeugen, steckt das »html«-Paket im Paket. Die Kommandos geben je ein HTML-Stück zurück, die das CGI mit »append« zur kompletten Seite zusammensetzt. Der Code in Listing 3 mischt die Kommandos aus dem HTML-Paket mit normalen Strings. Der größte Vorteil des HTML-Pakets ist, dass es automatisch das Öffnen und Schließen der Tags übernimmt.

Steht der gewünschte Monat fest, muss das Skript die Daten einlesen. Die Tcllib enthält verschiedene zusätzliche Datentypen wie Queue, Stack und Matrix. Auf Basis der Matrizen gibt es wiederum ein Paket zum Einlesen von CSV-Dateien. Das Kommando »::structure::matrix::matrix« definiert in Zeile 55 eine neue Matrix namens »daten«, die Zeile 59 per »::csv::read2matrix« mit dem Inhalt der CSV-Datei füllt.

In einer Matrix kann das Programm Werte wahlfrei auslesen, suchen und löschen. Im Beispiel genügt es, alle Einträge als HTML-Tabelle zu formatieren. Das Ergebnis ist in Abbildung 4 zu sehen. Raum für Verbesserungen ist ausreichend vorhanden: Wer will, steckt mehr Arbeit in die Formatierung und die Lokalisierung der Ausgabe.

Hier wird geholfen

Neben den vorgestellten Highlights hat die Tcllib noch viele weitere kleine Helfer zu bieten. Sie beschleunigen die Entwicklungsarbeit erheblich, da man Standardprobleme nicht mehr selbst implementiert muss. Neben der mitgelieferten Dokumentation finden sich für die Pakete der Tcllib meist recht ausführliche Beispiele im Tclers Wiki [11]. *(fjl)* ■

Infos

[1] Tcllib: [http://tcllib.sf.net]
[2] Dokumentation: [http://tcllib.sf.net/doc/]
[3] Scilab: [http://scilabsoft.inria.fr]
[4] Tide: [http://www.flaterco.com]
[5] Gridplus: [http://www.satisoft.com/tcltk/gridplus/]
[6] Icons im KDE-Systray: [http://groups.google.com/groups?selm=cca2a7$re8$1@news.xmission.com]
[7] Tcldes: [http://tcldes.sf.net]
[8] Pgtcl 1.4: [http://gborg.postgresql.org/project/pgtcl/]
[9] Websh: [http://tcl.apache.org/websh/]
[10] Rivet: [http://tcl.apache.org/rivet/]
[11] Tclers Wiki: [http://wiki.tcl.tk]
[12] Listings: [ftp://ftp.linux-magazin.de/pub/listings/magazin/2004/09/Feder-Lesen/]

Listing 3: CGI-Skript

```
01 #!/bin/sh
02 # Beispiel für ncgi, html und javascript \
03 exec tclsh $0 $@
04
05 package require ncgi
06 package require html
07 package require csv
08 package require struct
09
10 proc letzter_tag_des_monats {monat jahr} {
11   clock format [clock scan "$jahr-$monat-01 + 1
   month - 1 day"] -format %d
12 }
13
14 # Eingabe einlesen
15 ::ncgi::parse
16
17 # Monatsvorgabe ist der aktuelle Monat
18 set jahr 2004
19 set monat [::ncgi::value "monat" \
20   [clock format [clock second] -format %m]]
21
22 #
23 # Oberen Teil erzeugen
24 #
25 ::html::init
26
27 set html [::html::head "Tide für Hamburg"]
28
29 append html "<link rel='stylesheet'
   type='text/css' href='format.css'>"
30 append html "<style type='text/css'>"
31 append html [::html::bodyTag]
32 append html [::html::h1 "Tide für Hamburg"]
33 append html [::html::openTag form
   "action='listing3.tcl'"]
34
35 set monate {Januar 01 Februar 02 März 03 April 04 \
36   Mai 05 Juni 06 Juli 07 August 08 September 09 \
37   Oktober 10 November 11 Dezember 12}
38
39 set eingabe [::html::select monat "size=1"
   $monate $monat]
40 append html "Monat $eingabe <input type='submit'
   value='Absenden'>"
41 append html [html::closeTag]
42
43 #
44 # Gewünschten Monat aus der CSV-Datei lesen
45 #
46 set dateiname [file join daten "$jahr-$monat.csv"]
47 if {![file exists $dateiname]} {
48   append html "Keine Daten für Monat $monat
   vorhanden!"
49   append html [html::closeTag]
50   append html [html::closeTag]
51   puts stdout $html
52   return
53 }
54
55 struct::matrix::matrix daten
56 daten add columns 5
57
58 set fd [open $dateiname]
59 ::csv::read2matrix $fd daten ","
60 close $fd
61
62 #
63 # Ausgabe als Tabelle
64 #
65 append html [html::openTag table]
66 append html [html::hdrRow Tag ]
67 set letzterTag [letzter_tag_des_monats $monat $jahr]
68 set zeile 0
69 set gerade false
70 for {set tag 1} {$tag < $letzterTag} {incr tag} {
71   if {$gerade} {
72     set id gerade
73   } else {
74     set id ungerade
75   }
76   set gerade [expr {!$gerade}]
77
78   append html [::html::openTag tr "id='$id'"]
79   append html [::html::cell "id='tag'" " 
   $tag\. $monat"]
80
81   set tagPattern [format "$jahr-%02i-%02i" $monat
   $tag]
82   while {[regexp $tagPattern [daten get cell 1
   $zeile]]} {
83     append html [html::cell "" \
84       "[daten get cell 4 $zeile] \
85        [daten get cell 3 $zeile]<br>\
86       ([daten get cell 2 $zeile])"]
87     incr zeile
88   }
89
90   append html [::html::closeTag]
91 }
92
93 append html [html::closeTag]
94 append html [html::closeTag]
95 puts stdout $html
96 return
```