Client/Server mit Tcl

Uwe Berger

Uwe Berger

- Informatiker, PPS-Software
- Freizeit: Hard- und Software-Hacking
- BraLUG e.V.; Brandenburger Linux Infotag (06.11.2010)
- Beginn meiner Tcl-Geschichte: 1995/6
- Suche nach einer einfachen Möglichkeit komplexe Anwendung zu erstellen
  - damals nur Pascal-Erfahrungen
  - C war noch ein Fremdwort
- zufällig das Tcl-Buch in die Hand bekommen

Demo: *Cube-Edit/Cube-Simulator*

http://bralug.de/wiki/3D-LED-Display

Inhalt

- Tcl/Tk – ein paar allgemeine Bemerkungen
- Etwas Tcl-Rüstzeug
- Schrittweise zu Client/Server-Anwendungen
- Praktische Übungsaufgaben?
- Diskussion

Tcl/Tk – ein paar allgm. Bemerkungen

- Tcl/Tk - Was ist das
- Eigenschaften/Besonderheiten
- Tcl Syntax-Prinzipien
- Zusatzpakete
- Ein paar Worte zum Programmierstil
- Einsatzgebiete
- Informationsquellen

Tcl/Tk – Was ist das?

- Tcl - Tool command language
- eine Scriptsprache wie bash, Perl, php, etc.
- eine Interpretersprache (tclsh, wish)
- auf allen bedeutenden OS-Plattformen verfügbar
- John Ousterhout, ca. 1988
  - http://www.tcl.tk/about/history.html
  - Simulation von Schaltkreisen
  - sollte neben Java auch „Websprache“ werden
- Open Source

Tcl/Tk – Was ist das?

- Tk - Toolkit
- Programmierung von grafischen Benutzeroberflächen
- wurde ursprünglich für Tcl entwickelt (ebenfalls J.Ousterhout; 1988)
- Plattformübergreifend verfügbar (soweit grafische Oberfläche vorhanden)
- Anbindung auch für Perl, Python, Ruby und einige weitere vorhanden
- Open Source

Tcl Eigenschaften/Besonderheiten

- nach dem 2. Hinsehen, sehr einfache Syntax-Regeln
- durchgängig Polnische Notation (Befehl Parameter ...)
- "Alles ist ein String"
  - alle Datentypen können als String bearbeitet werden (intern natürlich auch Zahlenformate)
  - gleiches gilt auch für Programmcode (z.B. Evaluierung von Scriptfragmenten zur Laufzeit)
- sehr gute Unterstützung bei der Verarbeitung von Listen und Arrays

Tcl Eigenschaften/Besonderheiten

- ereignisgesteuerte Mechanismen für Socket- und Datei-Schnittstellen
- Seriell-, Parallel- und Netzwerk-Schnittstellen werden als Dateien behandelt
- Zeit- und Benutzerdefinierte Ereignisse
- einfache Ausnahmebehandlung (Fehlerbehandlung)
- einfache Client-/Server-Programmierung
- ... man findet immer wieder neue interessante Dinge ...

Tcl Syntax-Prinzipien

- immer(!): Kommandowort Parameter1 Parameter2 ...
- Variablen beginnen mit $ und müssen vor der ersten Verwendung gesetzt sein
- Zeichenketten stehen in doppelten Anführungsstrichen
- Ein Kommando wird von einem Zeilenende oder Semikolon begrenzt
- Backslash (\) am Zeilenende -> Befehl geht auf nächster Zeile weiter

Tcl Syntax-Prinzipien

- geschweifte Klammern schützen den Inhalt vor Interpretation (von außen nach innen); sind keine Strukturelemente
- Code in eckigen Klammern wird zuerst ausgeführt (von innen nach außen)
- Variablen nur lokal gültig und Gültigkeitsbereich muss explizit erweitert werden (global, upvar, uplevel)
- Kommentarzeichen (-befehl): # (am Zeilenanfang oder hinter einem Semikolon)

Zusatzpakete (Spracherweiterungen)

- Warum:
  - Performance
  - Hard-/Software-Schnittstellen
  - Zusammenfassung von komplexen Code
- Wie:
  - Packages in Tcl/Tk geschrieben(z.B. Tcllib, BWidget)
  - Erweiterungen in compilierten Bibliotheken (meist in C geschrieben
  - `package require Tcllib 1.0`

Programmierstil

- "Es gibt 1000 Wege nach Rom...", jeder muss seinen eigenen Tcl-Stil finden
- geschachtelte Klammern ({}, []) nur soweit, wie man Code noch versteht
- überschaubare Prozeduren/Funktionen schreiben
- globale Variablen, Variablenfelder
  - set gvar(x) 1; set gvar(y) 1; .... global gvar
- Zuerst GUI etc.; viele "Objekte" haben Eigenschaften, die zur Laufzeit einfach modifiziert werden können
- Zusatzpakete vermeiden

Einsatzgebiete

- Eigentlich überall da, wo es nicht auf extreme Performance ankommt → Interpretersprache
- man kommt mit wenigen Codezeilen schnell zum Ziel, da die einzelnen Syntax-Elemente sehr flexibel und mächtig sind
- es gibt ein paar Mechanismen, bei denen man sich in anderen Programmiersprachen sehr schwer tun würde
- GUI-Entwicklung
- auch für sehr komplexe Anwendungen tauglich
- Tcl ist keine „tote“ Sprache!

Bücher, Manuals, "Kochbücher"

- "Tcl und Tk. Entwicklung grafischer Benutzerschnittstellen für X Window System"; Ousterhout; Addison-Wesley
- "Effektiv Tcl/Tk programmieren"; Harrison, McLennan; Addison-Wesley
- http://www.activestate.com
- http://www.tcl.tk
- http://wiki.tcl.tk
- Tcl-Newsgroup (engl.): comp.lang.tcl

Etwas Tcl-Rüstzeug

- Starten eines Tcl-Script
- Kommandozeilenparameter
- Konfigurationsdateien
- Script-Evaluierung zur Laufzeit
- Befehl after
- Befehl catch
- Sicherer Interpreter
- Dateiereignisse
- Synchrone/Asynchrone Kommunikation

Starten eines Tcl-Script

- Interaktiv in der Tcl-Shell (tclsh, wish) selbst; z.B.:
  - `% source tcl_script.tcl`
- Script als Parameter der Tcl-Shell:
  - `tclsh tcl_script.tcl`
  - `wish tk_script.tcl`
- Als ausführbares Script mit Angabe des Interpreters:

```
#!/usr/bin/tclsh
package require Tk
...
```

Kommandozeilenparameter

- Gemeint ist z.B.: `meinscript.tcl -a 1234`
- Es ist kein „getopt“, wie in C, im Standardsprachumfang vorhanden
- Globale Interpretervariablen: `argc, argv, argv0`
- Beispiele/args/args.tcl
- Beispiele/args/source.tcl
- Parameterverarbeitung je nach Gegenheiten und Erfordernissen
- Defaultwerte nicht vergessen!

Konfigurationsdateien

- Konfigurationsdateien dann sinnvoll, wenn viele Parameter übergeben werden müssen
- Warum einen eigenen Syntax ausdenken und den entsprechenden erforderlichen Parser selbst schreiben?
- Tcl hat bereits einen Parser eingebaut!
- Tcl-Befehl: `source meinkonfig.cfg`
- Beispiele/konfig/konfig.tcl
- Achtung: Stichwort „Sicherer Interpreter“
- Defaultwerte nicht vergessen

Script-Evaluierung zur Laufzeit

- Wir erinnern uns: „Alles ist ein String“
- Programmcode muss nicht schon beim Start vollständig feststehen
- Warum nicht erst ein paar Script-Teile erst zur Laufzeit generieren und ausführen lassen?
- Syntax: eval <script>
- Beispiel/eval/eval.tcl
  - Achtung, was passiert bei der Eingabe von z.B.

```
puts [exec ls]
```

Der Befehl `after`

- Syntax: `after <ms>`
- Eigentlich nur der Pause-Befehl, aber...
- `after <ms> <script>` führt ein Script nach x ms aus; das Hauptprogramm läuft zwischenzeitlich weiter
- mit `after cancel, after info` können die im Hintergrund wartenden „Task“ verwaltet werden
- Beispiele/after/after.tcl
- `after idle` startet das Script im nächsten Eventloop
- Einsatzgebiete z.B. zyklisches Abfragen von Schnittstellen oder Zuständen (Polling)

Der Befehl `catch`

- Nichts ist schlimmer für den Anwender, als eine kryptische Fehlermeldung!
- Beispiele/catch/ohne_catch.tcl
- `catch` unterdrückt Fehlermeldungen des Interpreter
- Es können aussagekräftigere Meldungen im Fehlerfall ausgegeben werden
- Syntax: `catch <script> [var_errmsg]`
- Beispiele/catch/mit_catch.tcl

Sicherer Interpreter

- Es gibt nicht nur gute Menschen!
- Immer sämtliche Eingabe-Daten geprüfen, bevor sie verarbeitet oder gar ausgeführt werden.
- Stichwort: „Sicherer Interpreter“
- Syntax:
  - Erzeugung: `set safe [interp create -safe]`
  - Kommandoumfang: `$safe alias <func> <safe_func>`
- Bespiele/safe_interpr/safe_interpr.tcl

Dateiereignisse

- Syntax: fileevent <id> <readable|writetable> [script]
- Das angegebene Script wird aufgerufen, wenn die „Datei“ beschreib- oder auslesbar ist
- Zur Erinnerung: auch externe Schnittstellen werden als Dateien behandelt...
- Beispiele/fileevent/fileevent.tcl

Synchrone/Asynchrone Kommunikation

- Im Sinne von Interprozesskommunikation
  - Synchron: Senden und Empfangen von Daten erfolgen in fester Reihenfolge; Sender wartet bis Antwort da ist, ist also blockiert
  - Asynchron: Senden und Empfangen von Daten erfolgt zeitlich versetzt; Sender muss nicht auf Ergebnis warten und kann weiterarbeiten
- Achtung: Synchronität kann auch gewollt sein!
- wenn asynchron, dann wieder `fileevent`
- Beispiel: siehe später bei Client/Server

Schrittweise zu Client/Server-Anwendungen

- Verteilte Programmsysteme - ein paar allgemeine Bemerkungen
- Aufruf von externen Programmen
- Kommunikation via Pipes
- Client/Server
- gesicherte Client/Server-Verbindungen

Verteilte Programmsysteme

- Monolithische Programme können zu umfangreich und damit unwartbar werden!
- Monolithische Programme sind schwer testbar (Umfang, Seiteneffekte)
- Warum das Rad immer neu erfinden?
- Was hat ein GUI mit der eigentlichen Anwendung zu tun?
- Braucht jeder Anwender die gleiche Oberfläche und Funktionalität?
- Standardisierte Schnittstellen

Aufruf von externen Programmen

- Nicht für alle Dinge existieren unter Tcl/Tk entsprechende Befehle, aber vielleicht gibt es ja ein externes Programm...?
- Ein Programm hat zu viele (unverständliche) Optionen?
- Befehl: `exec <cmd>`
- Ausgaben des externen Programms landen auf der Standardausgabe, die man mit Tcl einlesen kann
- Beispiele/exec/exec.tcl, Beispiele/exec/exec_tk.tcl
- Problem: externes Programm wird immer wieder neu gestartet...

Kommunikation via Pipes

- … und `exec` blockiert das Hauptprogramm
- Deshalb externes Programm einmal starten und via Pipe kommunizieren (lesend und schreibend)
- Tcl-Befehl: `open "|programm" "r+"`
- Wie bekommt man mit, dass externes Programm Daten via Standardausgabe ausgibt: `fileevent`
- Achtung, Datenpufferproblematik: `fconfigure`
- Beispiele/pipe/pipe.tcl

Client/Server

- Client-Server-Modell: Verteilung von Aufgaben innerhalb eines Netzes
- Server:
  - bietet einen Dienst (ständig) an
  - passiv: wartet auf die Anforderung eines Clients
  - typische Beispiel: Mail-, Web-, DB-, Fileserver

Client/Server

- Clients:
  - nutzen einen Dienst
  - sind aktiv: fordern den Dienst eines Servers an
  - typische Beispiele: E-Mail-Client, News-Reader, Web-Browser
- Regeln der Kommunikation sind in Protokollen festgelegt (Bsp. http, pop3, imap, ntp, ftp, nfs, smb...)
- diese Anwendungsprotokolle, setzen wiederum auf Transportprotokollen auf (z.B. TCP/IP, UDP)

Client/Server

- Vorteile von Client/Servern
  - jede Komponente ist hochspezialisiert auf seine Aufgabe ausgerichtet
    - z.B. Server Datenbereitstellung/-verwaltung
    - z.B. Visualisierung/Weiterverarbeitung der Daten
  - kompakte, überschaubare Anwendungssysteme
  - effektive Resourcennutzung
  - Skalierbarkeit
  - Austauschbarkeit (z.B. hunderte E-Mail-Clients, dutzende E-Mail-Server)

Client/Server

- Peer-to-Peer:
  - sind Programme, die gleichzeitig Client/Server sind und untereinander kommunizieren
  - jeder Partner ist gleichberechtigt und kann alles

Client/Server mit Tcl

- Eigentlich haben wir bereits alles Grundwissen...
- ...bis auf den Befehl `socket`
- Client:
  - `socket <ip-adresse> <port>`
- Server:
  - `socket -server <script> <port>`
- Rückgabewert von `socket` in beiden Fällen die Kennung des Übertragungskanals („Alles ist eine Datei!“)
- Alles andere bleibt wie bereits kennengelernt!

Client/Server mit Tcl

- Grundgerüst eines Client/Server:
  - Beispiele/client_server/0_*.tcl
- Aber:
  - Sicherer Interpreter
  - asynchrone Kommunikation
  - mehrere Client-Verbindungen zu einem Server
  - Beispiele/client_server/1_*.tcl

Tcl-Zusatzpakete für Client/Server

- einige interessante/verbreitete Pakete:
  - Expect: http://expect.nist.gov/
  - TclCurl
  - Tcllib (auch diverse Verschlüsselungsalgorithm.)

Gesicherte Client/Server-Verbindungen

- Bitte nie blind den gesendeten Daten vertrauen; siehe auch Thema „Sicherer Interpreter“
- Kontext mittels verschiedener Algorithmen verschlüsseln
- ssh (Secure Shell):
  - Mittels Portforwarding
  - Prinzip: http://wiki.tcl.tk/11542
  - Zusatzpakete:
    - sshcomm: http://wiki.tcl.tk/13351
    - multissh: http://wiki.tcl.tk/14734
    - TclCurl

Gesicherte Verbindungen

- SSL (Secure Sockets Layer)/TSL (Transport Layer Security):
  - http://de.wikipedia.org/wiki/Transport_Layer_Security
  - diverse Zusatzpakete
    - tls: http://www.sensus.org/tcl/tls.htm
    - SSLtcl
    - tclssl
  - pop3 via TSL: http://wiki.tcl.tk/13352

Übungsbeispiele

- LED-Matrix-Server: Uebung/led_matrix/matrix_server.tcl
  - einen Client schreiben (z.B. einen Text ausgeben)
  - Server um einige Funktionen erweitern
- ein eigenes kleines Client/Server-System schreiben?

Fragen?

Danke für die Aufmerksamkeit!

http://bralug.de/wiki/BraLUG_auf_dem_CLT2010