Kooperatives Multitasking in Eigenregie

Kursmakler

Das Perl Object Environment versetzt ein Skript in die Lage, intern kooperatives Multitasking ohne Zutun des Betriebssystem-Schedulers zu betreiben. Das Beispiel für eine POE-Anwendung ist ein simpler, aber ruckfrei operierender Aktienticker. Der grafischen Oberfläche verleiht GTK ein Gesicht. Michael Schilli

Bild: Deutsche Börse AG

Applikationen mit grafischer Oberfläche arbeiten typischerweise Event-basiert: In einer Hauptschleife wartet das Programm auf Ereignisse wie Mausklicks und Tastatureingaben. Es ist wichtig, dass das Programm diese Events verzögerungsfrei abarbeitet und sofort wieder in die Haupt-Eventschleife eintritt, damit die Oberfläche nur unmerklich kurz nicht benutzbar ist.

Das hier vorgestellte Aktienticker-Programm kontaktiert periodisch die Finanz-Webseite von Yahoo, um die neuesten Börsenkurse einzuholen, Abbildung 1 zeigt es. Ein Request dauert, je nach Netzanbindung, inklusive der DNS-Auflösung des Servernamens schon mal ein paar Sekunden. Sehr gut wäre, wenn während dieser Zeit die Oberfläche der Applikation weiterackern kann.

Das Ziel könnte der Entwickler mit Multiprocessing oder Multithreading erreichen. Beides erhöht allerdings die Komplexität eines Programms beträchtlich: Kritische Sektionen wären vor Parallelzugriffen zu schützen, um die Integrität der Daten zu gewährleisten, und schwer zu analysierende Fehler schleichen sich ein. Wer mal einen Core Dump mit 200 laufenden Threads untersuchen musste, der weiß, wovon die Rede ist.

Eine alternative Möglichkeit, die ohne beide Nachteile auskommt, ist kooperatives Multitasking mit POE, dem Perl Object Environment [2] um Hauptentwickler Honocho Rocco Caputo. In der als State Machine implementierten Umgebung läuft zu jedem Zeitpunkt genau ein Prozess mit nur einem Thread, aber ein auf Benutzerebene realisierter „Kernel“ sorgt dafür, dass mehrere Aufgaben quasi zeitgleich abgearbeitet werden.

Kurs halten

Fürs Einholen von Aktienpreisen nimmt man in Perl üblicherweise das CPAN-Modul Yahoo::FinanceQuote:

```
use Finance::YahooQuote;
my @quote = getonequote($symbol);
```

Das Modul arbeitet jedoch synchron, was dem Wunsch nach ruckfreier Funktionsweise abträglich ist. Die Funktion »getonequote« setzt einen HTTP-Request an den Yahoo-Server ab, wartet auf die Antwort und kehrt erst dann mit einem Ergebnis zurück.

Während der Wartezeit sollte allerdings die Oberfläche besser in Schuss gehalten werden – wie's der Teufel will, zieht womöglich gerade jemand ein anderes Fenster über den Ticker. Das Fenster muss den eben verdeckten Teil seines Zuständigkeitsbereichs neu zeichnen (der so genannte Refresh). Der laufende Thread gönnt sich aber eine Phase der Untätigkeit, was ein gräuliches Loch auf dem Desktop erzeugt – so nicht.

Asynchron glücklich

Es wäre geschickter, einen Web-Request auszuschicken und, ohne auf das Ergebnis zu warten, die Aufmerksamkeit gleich wieder der grafische Oberfläche

Abbildung 1: Der GTK-basierte Aktienticker kontaktiert periodisch die Finance-Site von Yahoo.

zu schenken. Ist irgendwann die Antwort des Yahoo-Servers eingetroffen, sollte das eine Art Alarm auslösen. Also schnell das Fenster des Aktientickers aktualisieren – und wieder zurück in die GUI-Hauptschleife.

Genau dies leistet das POE-Framework. Es besteht aus einem Kern, in dem einzelne Applikationen Sessions registrieren. Dort springen State-Maschinen von Zustand zu Zustand und schicken einander Nachrichten. I/O-Aktivitäten erfolgen asynchron: Statt blockierend über ein File-Handle Daten einzulesen, sagt man: Hallo Kernel, ich will die Datei einlesen, wecke mich, wenn die Daten verfügbar sind.

Mit Volldampf auf der Daten-Überholspur

Zwar findet kein echtes asynchrones Schreiben oder Lesen statt (POE nutzt unter der Haube lediglich nicht-blockierende Syswrite- beziehungsweise Sysread-Funktionen), aber die bereitstehenden Daten werden mit Volldampf rausgepustet oder eingesogen. Der kooperative Aspekt bei POE ist, dass die Sessions sich darauf verlassen, dass keine ihrer Mitbewerberinnen rumtrödelt. Wenn eine Aufgabe den Prozessor nicht voll auslastet, muss die Session die Kontrolle freiwillig an den Kernel zurückgeben. Eine einzige unkooperative Stelle im Programm zöge das ganze System in Mitleidenschaft.

Das Multitasking mit einem einzigen Thread erleichtert die Programmentwicklung erheblich – kein Lock muss her, keine Überraschungen mit Race Conditions passieren und wenn doch mal ein Fehler auftritt, ist er meist leicht ausgemacht. POE arbeitet auch schön mit den Haupt-Eventschleifen einiger grafischer Umgebungen zusammen: Perl/Tk und Gtkperl erkennt POE automatisch und bindet sie nahtlos in den kooperativen Reigen ein. So sorgt der Kernel dafür, dass Ereignisse aus dem GUI ebenso ihre Zeitscheibchen bekommen wie die anderer explizit definierter Sessions. Damit ist auch das Refresh-Problem gelöst: Das GUI-Toolkit kann sich rechtzeitig darum kümmern.

Alarmierender Kernel

Der Aktienticker benötigt den Zustandsautomaten POE::Session nach Abbildung 2. Die Initialisierung »_start« baut unter anderem die GTK-Oberfläche auf und setzt den Aliasnamen »ticker«, damit die Session später leicht identifizierbar ist. Von dem Ausgangszustand geht die Kontrolle an den Kernel über. Alle 60 Sekunden (per Alarm) oder sobald jemand den »Update«-Knopf der Oberfläche drückt, kommt der »wake_up«-Zustand an die Reihe. Er stößt einen weiteren Zustandsautomaten vom Typ POE::Component::Client::HTTP an und gibt dann sofort die Kontrolle an den Kernel zurück.

»PoCoCli::HTTP« ist eine so genannte Komponente (Component) aus dem POE-Framework: ein Zustandsautomat, der seine eigene Session (im Listing 2, Zeile 61, »useragent« genannt) definiert, im »request«-Zustand Webanfragen entgegennimmt und dann im POE-Framework mitspielt, bis er eine HTTP-Antwort vollständig erhalten hat. Dann teilt er dem Kernel mit, dass die »ticker«-Session, die ihn aufgerufen hat, einen ihm vorher mitgeteilten Zustand (»yhoo_response«) anspringen soll.

Veranlasst der Kernel die »ticker«-Session dazu, nimmt sie die bereitliegende HTTP-Antwort entgegen, frischt die Aktien-Widgets in der Anzeige auf und gibt die Kontrolle umgehend an den Kernel zurück. Ab Zeile 59 startet die Komponente »POE::Component::Client::HTTP« mit »spawn()« und legt fest, dass im Server-»UserAgent«-String »gtkticker/0.01« auftaucht und dass hängende Anfragen nach 60 Sekunden abgebrochen werden.

Yahoo von Hand

Im Listing 2 definiert Zeile 9 die URL von Yahoos Aktienkursservice. Dessen CGI-Schnittstelle nimmt zwei Parameter entgegen:

- Einen Formatparameter (»f=«) mit den geforderten Feldern »s« (Symbol), »l1« (Aktienkurs) und »c1« (Veränderung in Prozent seit dem letzten Börsentag) und
- einen Symbolparameter, der die Börsenkürzel der geforderten Aktiengesellschaften Komma-separiert enthält, zum Beispiel »YHOO,MSFT,TWX«.

Der Yahoo-Server antwortet darauf in der Art:

```
"YHOO",45.38,+0.35
"MSFT",27.56,+0.19
"TWX",18.21,+0.75
```

Gtkticker nimmt die Antwort einfach zeilenweise und an den Kommata auseinander und schleust alles auf die grafische Oberfläche.

Konfiguration im Heimatverzeichnis

Zeile 11 benennt mit ».gtkicker« im Heimatverzeichnis des Benutzers die Datei mit den Symbolen, die der Ticker anzeigen soll. Die Zeilen 25 bis 31 lesen die Datei zeilenweise ein und verwerfen mit

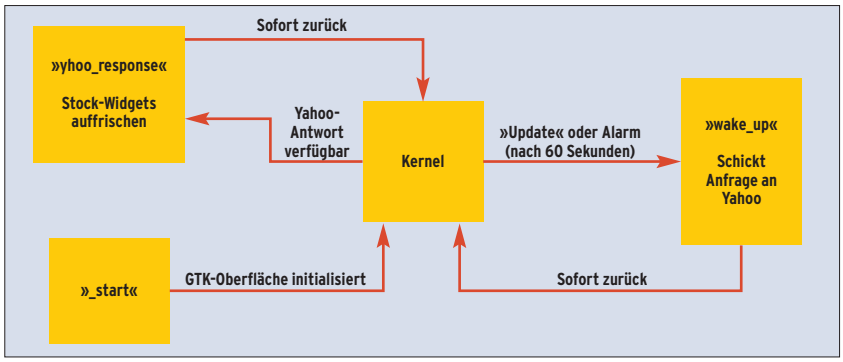

Abbildung 2: Der Zustandsautomat von Gtkticker: Nach dem Initialisierungszustand »_start« geht die Kontrolle an den Kernel. Alle 60 Sekunden kommt der »wake_up«-Zustand an die Reihe, der einen weiteren Zustandsautomaten anstößt und die Kontrolle an den Kernel zurückgibt.

»#« beginnende Kommentarzeilen (Zeile 28). Die implizite For-Schleife am Ende der Zeile 29

```
... for /(\S+)/g;
```

führt den Ausdruck links von ihr für alle Wörter einer Zeile aus und setzt jeweils das Börsensymbol in die Variable »$_«. Der Push-Aufruf stapelt die Werte aus »$_« im Array »@SYMBOLS« – so dürfen auch mehrere Symbole durch Leerzeichen getrennt in einer Zeile stehen. Listing 1 zeigt zeigt eine Beispieldatei.

Trotz POE-Frameworks verwendet Gtkticker beim Lesen der Konfigurationsdatei die regulären synchronen I/O-Funktionen, denn dieses File ist kurz und der POE-Kernel läuft noch gar nicht.

Listing 1: Konfigurationsdatei

```
01 # ~/.gtkticker
02 TWX
03 MSFT
04 YHOO AMZN RHAT
05 DODGX
06 JNJ COKE IBM SUN
```

Der Tanz beginnt

Zeile 33 definiert den Zustandsautomaten des Tickers. Der Parameter »inline_states« weist mit einer Hashreferenz den Zuständen Funktionen zu, die der Kernel anspringt – falls die Zustände erreicht sind. Dann schiebt Zeile 44 mit

```
$poe_kernel->post("ticker", "wake_up");
```

über die von »POE« exportierte Variable »$poe_kernel« den Zustand »wake_up« für die »ticker«-Session in den Kernel. Zeile 45 startet mit

```
$poe_kernel->run();
```

die Kernel-Hauptschleife, die das Programm bis zum Shutdown nie mehr verlässt. Das war's!

Die zuvor gezeigte Konstruktion des »POE::Session«-Objekts hat einen Seiteneffekt: Die dem »_start«-Zustand zugewiesene und ab Zeile 48 definierte Routine »start()« wurde ausgeführt. Sie setzt den Aliasnamen der Session auf »ticker« und springt sodann in »my_gtk_init()«. Diese ab Zeile 82 definierte Funktion baut die GTK-Oberfläche zusammen.

GUIs mit GTK

»Gtk« ist ein CPAN-Modul von Marc Lehmann, der auch freundlicherweise das Manuskript für diesen Artikel durchsah. Eigentlich hat »Gtk2« jenes Modul schon abgelöst, doch die neue Version spielt mit POE noch nicht so recht zusammen. Das ehrwürdige GTK erledigte den Job dagegen hervorragend.

Ein Objekt der Klasse »Gtk::Window« ist das Hauptfenster der Applikation. Oben

Listing 2: Gtkticker

```
001 #!/usr/bin/perl
002 ##########################################
003 # gtkticker
004 # Mike Schilli, 2004 (m@perlmeister.com)
005 ##########################################
006 use warnings;
007 use strict;
008
009 my $YHOO_URL = "http://quote.yahoo.com/d?".
010                "f=sl1c1&s=";
011 my $RCFILE   = "$ENV{HOME}/.gtkticker";
012 my @LABELS       = ();
013 my $UPD_INTERVAL = 60;
014 my @SYMBOLS;
015
016 use Gtk;
017 use POE qw(Component::Client::HTTP);
018 use HTTP::Request;
019 use Log::Log4perl qw(:easy);
020 use Data::Dumper;
021
022 Log::Log4perl->easy_init($DEBUG);
023
024     # Read config file
025 open FILE, "<$RCFILE" or
026     die "Cannot open $RCFILE";
027 while(<FILE>) {
028     next if /^\s*#/;
029     push @SYMBOLS, $_ for /(\S+)/g;
030 }
031 close FILE;
032
033 POE::Session->create(
034   inline_states => {
035       _start         => \&start,
036       _stop    => sub { INFO "Shutdown" },
037       yhoo_response  => \&resp_handler,
038       wake_up        => \&wake_up_handler,
039   }
040 );
041
042 my $STATUS;
043
044 $poe_kernel->post("ticker", "wake_up");
045 $poe_kernel->run();
046
047 ##########################################
048 sub start {
049 ##########################################
050
051   DEBUG "Starting up";
052
053   $poe_kernel->alias_set( 'ticker' );
054
055   my_gtk_init();
056
057   $STATUS->set("Starting up");
058
059   POE::Component::Client::HTTP->spawn(
060     Agent     => 'gtkticker/0.01',
061     Alias     => 'useragent',
062     Timeout   => 60,
063   );
064 }
065
066 ##########################################
067 sub upd_quotes {
068 ##########################################
069
070     my $request = HTTP::Request->new(
071         GET => $YHOO_URL .
072                join ",", @SYMBOLS);
073
074     $STATUS->set("Fetching quotes");
075
076     $poe_kernel->post('useragent',
077       'request', 'yhoo_response',
078       $request);
079 }
080
081 ##########################################
082 sub my_gtk_init {
083 ##########################################
084
085     my $w = Gtk::Window->new();
086     $w->set_default_size(150,200);
087
088         # Create Menu
089     my $accel = Gtk::AccelGroup->new();
090     $accel->attach($w);
091     my $factory = Gtk::ItemFactory->new(
092        'Gtk::MenuBar', "<main>", $accel);
093
094     $factory->create_items(
095       { path =>  '/_File',
096         type =>  '<Branch>',
097       },
098       { path        =>  '/_File/_Quit',
```

hängt ein typisches Pull-down-Menü, das aus einem Menübalken mit einem Eintrag »File« besteht. Dessen ausziehbares Menü enthält nur den Eintrag »Quit«, der die Applikation über eine Callback-Routine mit »Gtk->exit(0)« beendet. Dafür, dass der Benutzer auch mit der Tastenkombination [Ctrl]+[Q] das Programm verlassen kann, sorgt ein »Gtk::AccelGroup«-Objekt, das die Menüsteuerung mit so genannten Accelerators bestückt.

Menüs aus der Fabrik

Das Menü wird mit Hilfe der »Gtk::ItemFactory« aufgebaut, die zuerst einen Menübalken vom Typ »Gtk::MenuBar« erzeugt. Die Menü-Einträge und ihnen untergeordnete ausklappbare Menüs entstehen per »create_items()«.
Der »path«-Parameter gibt dabei die Lage des Menüpunkts an – so spezifiziert »/_File/_Quit« den Eintrag »Quit« unter dem »File«-Eintrag im Menübalken. Die Underlines »_« sorgen dafür, dass »Gtk« das folgende Zeichen unterstreicht und der Anwender mit den entsprechenden Tasten (etwa [Alt]+[F]) im Menü navigieren kann. Der »callback«-Parameter setzt eine Funktion, die »Gtk« anspringt, falls der Benutzer den Eintrag mit der Maus anwählt oder die über den »accelerator«-Parameter definierte Tastenkombination drückt. Die Callback-Funktion ist hier als anonyme Subroutine ausgelegt: Sie trägt keinen Namen und ist nur an Ort und Stelle zu benutzen.

Layout-Manager

Um Widgets geometrisch anzuordnen, kommen zwei verschiedene Verfahren zum Einsatz: »Gtk::VBox« und »Gtk::Table«. Das Container-Element »Gtk::VBox« richtet in ihm enthaltene Widgets vertikal aus. Seine »pack_start()«-Methode platziert dabei die Elemente vom oberen Rand nach unten, während »pack_end()« seine Widgets von unten nach oben stapelt. Der Aufruf

```
$vb->pack_start(Menübalken, Expand, Fill, ➥
Padding);
```

packt den Menübalken oben in die VBox. Gtkticker holt den Balken per »$factory->get_widget('<main>')« in Zeile 108 über seinen Namenseintrag aus der Factory.
Der *Expand*-Parameter gibt an, ob die Fläche, in der das Widget schwimmt, sich vergrößert, falls der Bediener das Hauptfenster mit der Maus aufzieht. Falls ja, gibt *Fill* an, ob das Widget selbst sich ausdehnt – auf diese Weise können kleine Druckknöpfe riesengroß werden. *Padding* spezifiziert schließlich die Anzahl der Pixel, die das Widget mindestens vertikal zu seinen Nachbarn hält.

Listing 2 (Fortsetzung): Gtkticker

```
099         accelerator =>  '<control>Q',
100         callback    =>
101                   [sub { Gtk->exit(0) }],
102       });
103
104     my $vb   = Gtk::VBox->new(0, 0);
105     my $upd  = Gtk::Button->new(
106                               'Update');
107
108     $vb->pack_start($factory->get_widget(
109                     '<main>'), 0, 0, 0);
110
111         # Button at bottom
112     $vb->pack_end($upd,  0, 0, 0);
113
114         # Status line on top of buttons
115     $STATUS = Gtk::Label->new();
116     $STATUS->set_alignment(0.5, 0.5);
117     $vb->pack_end($STATUS, 0, 0, 0);
118
119     my $table = Gtk::Table->new(
120                      scalar @SYMBOLS, 3);
121     $vb->pack_start($table, 1, 1, 0);
122
123     for my $row (0..@SYMBOLS-1) {
124
125       for my $col (0..2) {
126
127         my $label = Gtk::Label->new();
128         $label->set_alignment(0.0, 0.5);
129         push @{$LABELS[$row]}, $label;
130
131         $table->attach_defaults($label,
132                 $col, $col+1, $row, $row+1);
133       }
134
135     }
136
137     $w->add($vb);
138
139         # Destroying window
140     $w->signal_connect('destroy',
141         sub {Gtk->exit(0)});
142
143         # Pressing update button
144     $upd->signal_connect('clicked',
145         sub { DEBUG "Sending wake_up";
146               $poe_kernel->post(
147                 'ticker', 'wake_up')} );
148     $w->show_all();
149 }
150
151 ##########################################
152 sub resp_handler {
153 ##########################################
154     my ($req, $resp) =
155             map { $_->[0] } @_[ARG0, ARG1];
156
157     if($resp->is_error()) {
158         ERROR $resp->message();
159         $STATUS->set($resp->message());
160         return 1;
161     }
162
163     DEBUG "Response: ", $resp->content();
164
165     my $count = 0;
166
167     for(split /\n/, $resp->content()) {
168         my($symbol, $price, $change) =
169                               split /,/, $_;
170         chop $change;
171         $change = "" if $change =~ /^0/;
172         $symbol =~ s/"//g;
173         $LABELS[$count][0]->set($symbol);
174         $LABELS[$count][1]->set($price);
175         $LABELS[$count][2]->set($change);
176         $count++;
177     }
178
179     $STATUS->set("");
180
181     1;
182 }
183
184 ##########################################
185 sub wake_up_handler {
186 ##########################################
187     DEBUG("waking up");
188
189         # Initiate update
190     upd_quotes();
191
192         # Re-enable timer
193     $poe_kernel->delay('wake_up',
194                     $UPD_INTERVAL);
195 }
```

Statusmeldungen zeigt Gtkticker in einem unauffälligen »Gtk::Label«-Widget direkt über dem »Update«-Knopf. Die »set_alignment()«-Methode erreicht mit

```
$STATUS->set_alignment(0.5, 0.5);
```

dass der Text horizontal und vertikal zentriert wird. Wer experimentieren will: Ein Wert »0.0« wäre links-, »1.0« wäre rechtsbündig.
Das Container-Element »Gtk::Table« hingegen gibt Perl-Programmierern ein Werkzeug in die Hand, um andere Widgets bequem in Tabellenform zu arrangieren: Die »attach_defaults()«-Methode erwartet das anzuordnende Widget und je zwei Spalten- und zwei Reihenkoordinaten, zwischen denen das Widget liegen soll. Zum Beispiel legt

```
$table->attach_defaults($label, 0, 1, 1, 2);
```

fest, dass das mit »$label« referenzierte »Gtk::Label«-Objekt in der ersten Reihe („zwischen 0 und 1“) und in der zweiten Spalte („zwischen 1 und 2“) der Tabelle »$table« aufgehängt sein soll.

And Action!

Widgets vom Typ »Gtk::Button« kann man Aktionen zuordnen, die »Gtk« ausführt, sobald der Benutzer den vereinbarten Knopf drückt. Die in Zeile 144 aufgerufene Methode »signal_connect()« legt fest, dass »Gtk« einen »wake_up«-Event an den POE-Kernel schickt, falls der Benutzer auf den »Update«-Knopf klickt. Auch das Hauptfenster verknüpft eine Aktion – im Ereignis löst der Benutzer das Schließen des Fensters aus, wenn er auf das »X« rechts oben klickt.

```
$w->signal_connect('destroy',
    sub {Gtk->exit(0)});
```

räumt die »Gtk«-Session auf und bricht das Programm ab. Sind alle Widgets definiert, werden sie von der »show_all()«-Methode des Hauptfensters (Zeile 148) auf den Bildschirm befördert.

Der Kernel schlägt zurück

Im Zustand »yhoo_response« springt der POE-Kernel zur ab Zeile 152 gelisteten Funktion »resp_handler«. Per Definition legt »POE::Component::Client::HTTP« dabei ein Request- und ein Response-Paket in »ARG0« und »ARG1« ab. POE nutzt ja diese seltsam anmutende Parameterübergabe, nachdem es neue numerische Konstanten wie »KERNEL«, »HEAP«, »ARG0«, »ARG1« eingeführt hat. Der POE-Autor erwartet, dass der Programmierer sie nutzt, um das Funktionsparameter-Array »@_« zu indizieren: »$_[KERNEL]« zum Beispiel gibt so immer das Kernelobjekt zurück.
Die erwähnten Request- und Response-Pakete sind Referenzen auf Arrays, deren erste Elemente die »HTTP::Request«- beziehungsweise »HTTP::Response«-Objekte enthalten. Also extrahiert der »map«-Befehl sie in den Zeilen 154 und 155 nach »$req« und »$resp«.
Tritt ein HTTP-Fehler auf, erzeugt Zeile 159 eine entsprechende Meldung im Status-Widget und kehrt sofort zurück. Andernfalls wird das globale zweidimensionale Array der Label-Widgets aufgefrischt, die für jede zu überwachende Aktie das Börsensymbol, den aktuellen Kurs und die prozentuale Veränderung anzeigen (null Prozent wird als Sonderfall einfach unterdrückt).

Periodisch verzögerter Alarm

Ein »wake_up«-Event im POE-Kernel löst die ab Zeile 185 definierte Routine »wakeup_handler()« aus. Sie ruft die ab Zeile 67 implementierte Funktion »upd_quotes()« auf, die ein »HTTP::Request«-Objekt definiert und es per Event an die Komponente »POE::Component::Client::HTTP« schickt. Als Zielzustand für den »ticker« gibt sie »yhoo_response« an. Nachdem diese Vorarbeit erledigt ist, setzt »wakeup_handler()« mit der »delay()«-Methode des Kernels einen Weckruf. Der löst nach der in »$UPD_INTERVAL« definierten Sekundenzahl (60) einen »wake_up«-Event der »ticker«-Session aus. Ab jetzt frischt der Ticker alle 60 Sekunden seine Aktiendaten auf, egal ob der Benutzer den »Update«-Knopf drückt oder nicht.

Installation

Die erforderlichen POE-Module POE sowie POE::Component::Client::HTTP installiert man am besten mit einer CPAN-Shell. Falls das Modul POE::Component::Client::DNS ebenfalls installiert ist, werden sogar DNS-Anfragen asynchron bearbeitet, sonst verursacht das eingesetzte »gethostbyname()« gelegentlich eine kleine Verzögerung.
»Gtk« vom CPAN zieht diverse Abhängigkeiten herein – und bereitete jedenfalls dem System des Autors dieses Beitrags einige Probleme. Aber

```
touch ./Gtk/build/perl-gtk-ref.pod
perl Makefile.PL --without-guessing
```

im Distributionsverzeichnis mit nachgeschobenem »make install« brach alle Widerstände. Und wie sonst auch lässt sich die Geschwätzigkeit der Logs des Skripts auf dem Terminal mit »Log::Log4perl« (ebenfalls vom CPAN) und in Zeile 22 einstellen.
Es ist faszinierend, wie glatt sich die Oberfläche bedienen lässt. Selbst wenn man während eines automatischen Auffrischvorgangs über ein langsames Netzwerk im Menü rumfuhrwerkt, kommt die Applikation nicht ins Schleudern. Eine ideale Technologie für alle grafischen Client-Applikationen. *(jk)* ■

Infos

[1] Listings zu diesem Artikel: [ftp://www.linux-magazin.de/pub/listings/magazin/2003/04/Perl]
[2] POE: [http://poe.perl.org]
[3] Jeffrey Goff, „A Beginner's Introduction to POE“, 2001: [http://www.perl.com/pub/a/2001/01/poe.html]
[4] Matt Sergeant, „Programming POE“, Vortrag auf TPC 2002: [http://axkit.org/docs/presentations/tpc2002]
[5] Gtkperl: [http://gtkperl.org]
[6] Gtkperl-Tutorial: [http://personal.riverusers.com/~swilhelm/gtkperl-tutorial/]
[7] Eric Harlow, „Developing Linux Applications with GTK+ and GDK“: New Riders, 1999, ISBN 0735700214

Der Autor

Michael Schilli arbeitet als Software-Engineer für AOL/Netscape in Mountain View, Kalifornien. Er hat „Goto Perl 5“ (deutsch) und „Perl Power“ (englisch) für Addison-Wesley geschrieben und ist unter [mschilli@perlmeister.com] zu erreichen. Seine Homepage ist [http://perlmeister.com].