Renée Bäcker

Perl für Vortragende

Der 13. Deutsche Perl-Workshop ist gerade vorbei. An drei Tagen wurden Vorträge gehalten und viel diskutiert. So wie auf dem Deutschen Perl-Workshop gibt es noch viele andere Gelegenheiten, bei denen man Vorträge halten kann. In den meisten Fällen sind es "statische" Vorträge, das meine ich die Vorträge, bei denen es nur Folien gibt.

Diese Vorträge sind auch in den meisten Fällen vollkommen in Ordnung. Oft ist es aber für den Zuschauer interessanter, wenn noch etwas gezeigt wird - ein Stück Code, eine Vorführung einer Anwendung oder etwas Ähnliches.

So eine Dynamik in seine Vorträge zu bekommen, kann aber für den Vortragenden "gefährlich" sein. Gerade ungeübte Redner sind häufig sehr nervös und machen dadurch Fehler. Wenn sich die Fehler häufen, assoziieren die Zuhörer diese Fehler mit dem Code/der Anwendung. Oder als Vortragender vergisst man, was man zeigen wollte. Hinterher ärgert man sich, dass die wichtigsten Features der Anwendung nicht gezeigt wurden.

Aber es liegt ja nicht immer am Vortragenden selbst wenn bei solchen Live-Demos etwas schief geht. Auf der YAPC::Europe 2007 habe ich einen Lightning Talk gehalten, der voll automatisiert war. Ich wollte zeigen, wie man mit Perl automatisiert Vorträge erstellen kann und auch Online-Spiele erfolgreich meistern kann. Kurz vor der Session kam ein anderer Teilnehmer zu mir, um seinen Lightning Talk mit meinem Laptop zu präsentieren. Dazu fehlte ein Plugin im Firefox, also noch schnell installiert.

Zum Glück habe ich dann meinen Talk nochmal schnell zur Probe laufen lassen und dann der GAU - das Online-Spielen funktionierte nicht mehr. Das Plugin hat eine neue Toolbar eingeblendet, so dass die Positionen von Elementen des Spiels an einer anderen Position zu finden waren. Man sieht, dass durch kleine Änderungen am System, der ganze Vortrag scheitern kann.

Wenn etwas aus der laufenden Entwicklung gezeigt werden soll, dann kann eine kurze Änderung am Code, der fehlerhaft z.B. im SVN eingecheckt wurde, vieles kaputtmachen und die Live-Demo platzen lassen.

Probleme vermeiden

Die Probleme bei solchen Vorträgen lassen sich zum Teil ganz einfach vermeiden: Kurz vor dem Vortrag sollten keine Änderungen mehr am System vorgenommen werden, bei Vorführungen von aktuellen Entwicklungen, sollte man sich merken, bei welcher Version alles funktioniert hat.

Man kann sich aufschreiben was man machen will - was aber ganz klar nicht so "cool" aussieht - und das oberste Gebot ist natürlich "üben, üben, üben". Dazu eignen sich Perlmonger-Treffen, die eigenen Kollegen oder auch ein kleinerer Workshop wie den Frankfurter Perl-Community Workshop.

Eine Person, die extrem gut bei Vorträgen ist, ist Damian Conway. Er kennt seine Folien fast auswendig, er ist witzig und hat extrem gute Ideen. Auch wenn er selbst immer wieder sagt, dass man Live-Demos nicht machen sollte, zeigt er immer wieder Code.

Mit einem Modul von Damian Conway kann man solche Live-Demos mit Code auch Vortragssicher machen: `IO::Prompt`

IO::Prompt to the rescue

Es ist schon toll, wenn man live auf der Bühne programmiert und es funktioniert alles. Nur wenn man es wirklich live macht, vertippt man sich wahrscheinlich. Deshalb kann man so tun als würde man live programmieren, dabei ist alles schon vorbereitet.

Das Programm an sich ist ganz simpel:

```
use IO::Prompt;

system( 'clear' );
print 'perl@ubuntu:~/foo$ ';

while (prompt '') {
}

__DATA__
__PROMPT__
```

Das `system()` wird nur benutzt um den Programmaufruf zu verstecken. Und danach wird ein Prompt ausgegeben. So merkt der Zuhörer nicht, dass alles nur ein Fake ist. Das Entscheidende ist dann die `while`-Schleife. `prompt` ist eine Funktion von `IO::Prompt`. Dieser Funktion kann man mehrere Parameter übergeben, die praktisch für den eigentlichen Einsatz des Moduls ist - Benutzerinteraktion. Für die Fake-Demo muss ein leerer Prompt übergeben werden.

Im `__DATA__`-Bereich gibt es dann wieder einen extra Bereich, den `IO::Prompt` als Quelle für die Ausgaben nutzt. Dieser wird mit `__PROMPT__` gekennzeichnet. Ein komplettes Beispiel sieht somit folgendermaßen aus:

```
use IO::Prompt;

system( 'clear' );
print 'perl@ubuntu:~/foo$ ';

while (prompt '') {
}

__DATA__
__PROMPT__
cat > test.pl
#!/usr/bin/perl

use strict;
use warnings;
print "Hallo Welt!";
```

Wenn das Programm gestartet wird, wird mit jedem Tastaturanschlag ein Zeichen aus dem Prompt-Bereich ausgegeben. Nur am Zeilenende muss man zweimal die Return-Taste drücken, damit es weiter geht. Aber Vorsicht! Wenn man mitten in der Zeile (also z.B. nach "use") die Return-Taste drückt, wird der Rest der Zeile ganz schnell ausgegeben. Da könnte es passieren, dass der Fake auffällt.

Das Programm aus dem Prompt-Bereich sollte man schon als Skript gespeichert haben (hier: test.pl), da `IO::Prompt` nur die Ausgabe macht und nicht den Befehl ausführt.

Nachdem die letzten Zeichen aus dem Fake-Programm ausgegeben wurden, kann man mit Ctrl+D das Programm beenden und man ist wieder ganz normal in der Shell unterwegs. Jetzt kann man test.pl ausführen und keiner merkt, dass das Programmieren der Anwendung schon vorher erledigt wurde.

Aber auch wenn man jetzt die Vertipper nicht im Programm haben kann, sollte man den Vortrag üben, damit man keine Überraschungen erlebt was denn als nächstes kommt.

Leider kann man das Modul nicht mit ActivePerl oder StrawberryPerl nutzen, da es auf `IO::Tty` aufbaut. Wer es unter Windows dennoch nutzen will, muss Cygwin nehmen. Unter Linux funktioniert das Modul einwandfrei.

GUIs automatisiert bedienen

Aber nicht jede Live-Demo beinhaltet Programmierung; viel häufiger werden es Anwendungen mit GUIs sein. Diese Anwendungen automatisiert zu bedienen ist an sich kein Problem - auch hier liefert CPAN alle notwendigen Tools. Für Windows gibt es `Win32::GuiTest` und für X11 gibt es `X11::GUITest`. Für letzteres gibt es sogar ein Modul, mit dem man die Interaktionen aufnehmen kann um sie anschließend wieder abspielen zu können. Das Modul `X11::GUITest::record` wurde in der Ausgabe "Sommer 2009" von `$foo` vorgestellt.

Die entstehenden Programme sind aber nicht besonders hübsch. Ein Beispiel:

```
#!/usr/bin/perl

use strict;
use warnings;

use X11::GUITest qw/
  StartApp
  WaitWindowViewable
  SendKeys
/;

# Start application
StartApp('padre');

# Wait for application window to come up
# and become viewable.
my ($GEditWinId) =
    WaitWindowViewable('Padre');
if (!$GEditWinId) {
  die("Couldn't find window in time!");
 }

sleep 11;

# session <ctrl+alt+o>

SendKeys( "^(%(o))" );
sleep 3;
SendKeys( "\n" );
```

Ich habe hier einige `sleep`s eingebaut, weil das Programm in einer VM läuft, die nicht gerade die schnellste ist. Damit aber immer die notwendigen Informationen da sind, ist ein Warten unabdingbar.

Nachdem die Zielanwendung (hier: padre) gestartet ist, kann mit der automatischen Bedienung angefangen werden. In diesem Beispielprogramm werden nur ein paar Tastatureingaben simuliert. `SendKeys` wird dabei ein String übergeben, der die Tastatureingaben darstellt. Hier muss man sich erst an die Darstellungsweise gewöhnen: `^(...)` ist die Ctrl-Taste, `%(...)` die Alt-Taste und für weitere "Spezialtasten" wie die Pfeiltasten gibt es zusätzliche Syntax. In dem String können auch ganze Abfolgen von Tasten enthalten sein:

```
^(g){PAUSE 1000}5\n{END}{LEF 13}
```

Das sendet Ctrl+g, macht dann eine Sekunde Pause, sendet dann eine 5 und einen Zeilenumbruch. Schickt die "Ende"-Taste und geht dann 13 Positionen nach links.

Neben dem Senden von Tastatureingaben kann man auch die Maus steuern. Dazu gibt es die Funktionen `MoveMouseAbs` und `ClickMouseButton`. Das Problem bei der Verwendung der Mausfunktionen ist, dass die Position nur absolut angegeben werden kann. Das heißt, immer wenn an der GUI etwas geändert wird oder sich am System etwas ändert, müssen die Skripte überprüft werden.

Auf jeden Fall muss die Automatisierung von GUIs gut geplant werden.

Zu den offenen Punkten gehört sowohl bei `IO::Prompt` als auch `X11::GUITest`, wie man es am geschicktesten umsetzt, dass man wirklich noch auf Rückfragen aus der Zuhörerschaft eingehen kann. Gerade bei der Automatisierung von GUIs mit `X11::GUITest` kommt man schwer aus dem ursprünglich geplanten Ablauf raus, um evtl. nochmal auf eine andere Einstellung zurückzugehen.

Ich selbst habe es bisher so gelöst, dass ich Fragen erst am Ende des Vortrags akzeptiert habe. Es wäre natürlich schöner, wenn man den Ablauf unterbrechen kann und dann einfach wieder am Ausstiegspunkt einsteigen kann.

Diese zwei Beispiele zeigen, dass man durchaus "Live-Demos" in Vorträgen zeigen kann und man einige Unsicherheiten ausschließen kann.