Zugriffe auf Webseiten mit Screen Scrapern automatisieren

Datenrüssel

Manchmal ist es zu umständlich, sich durch die Hierarchie einer Website zu wühlen, nur um eine bestimmte Information zu finden. Das Perl-Modul WWW::Mechanize::Shell hilft so genannte Screen Scraper zu schreiben, die wie Browser agieren und den Zugriff auf Webseiten automatisieren. Michael Schilli

In Kalifornien darf man sich das Nummernschild fürs Auto gegen eine Gebühr selbst aussuchen. Aus nahe liegenden Gründen führt mein 13 Jahre alter Acura Integra das Kennzeichen „PERL MAN“ (Abbildung 1). Die Kraftfahrzeugbehörde gibt sich fortschrittlich und bietet eine Webseite an, die prüft, ob das gewünschte Kennzeichen verfügbar ist – allerdings geht das vielen Anwendern zu langsam. Dieser Artikel zeigt, wie man die Abarbeitung von Webanfragen in einem Perl-Skript nachbildet.

Dazu bieten sich so genannte Screen Scraper an. Diese Programme bereiten bestimmte Daten für die Darstellung am Bildschirm auf, sie formatieren zum Beispiel HTML-Seiten für die Shell um.

Das passende Perl-Modul

Für solche Screen-Scraper-Programme hat Andy Lester das Modul WWW::Mechanize entwickelt. Neuerdings steht mit WWW::Mechanize::Shell von Max Maischein sogar eine Shell zur Verfügung, mit der Programmierer eine Browser-Session sehr schnell in ein Skript umwandeln. Die Beschreibung der Session erfolgt allein auf logischer Ebene, zum Beispiel: Folge einem Link auf der aktuellen Seite, der den Text ’xxx’ enthält. Das Skript funktioniert also auch dann, wenn sich das Format der Webseite ändert.

Über die CPAN-Shell lassen sich die Module WWW::Mechanize und WWW::Mechanize::Shell komfortabel installieren. Sie hängen ihrerseits von weiteren Modulen ab, deren Installation dank der Shell-Automatik ebenfalls einfach ist. Der Sreen Scraper benötigt zusätzlich das Modul IO::Socket::SSL, um auch mit HTTPS-Seiten zurechtzukommen.

Die Browser-Emulator-Shell startet nach folgendem Perl-Aufruf:

```
perl -MWWW::Mechanize::Shell -eshell
```

Dem Benutzer bietet sich nun ein Prompt. Um die Seite der kalifornischen Kraftfahrzeugstelle „Department of Motor Vehicles“ zu laden, tippt er einfach »get http://www.dmv.ca.gov« ein, die Shell antwortet daraufhin mit »Retrieving http://www.dmv.ca.gov(200)« um zu signalisieren, dass das Skript die Startseite erfolgreich geladen hat (HTTP-Code 200).

Die Browserdarstellung der Seite in Abbildung 2 zeigt den gesuchten Link zu »Personalized Plates«. Um herauszufinden welche Links die Shell erkannt hat, genügt das Kommando »links«, das alle verfügbaren Verweise anzeigt:

```
>links
...
[14] Vehicle Industry & Commercial Permits
[15] Personalized Plates
[16] Disabled Placards
...
```

Nun könnte der Benutzer einfach dem Link Nummer 15 mit »open 15« folgen, aber das wäre nur begrenzt gültig, da

Abbildung 1: Der erstaunliche PERL MAN: Schon 13 Jahre alt, doch er kann es einfach nicht lassen, die Sportwagen an den Ampeln San Franciscos zu provozieren.

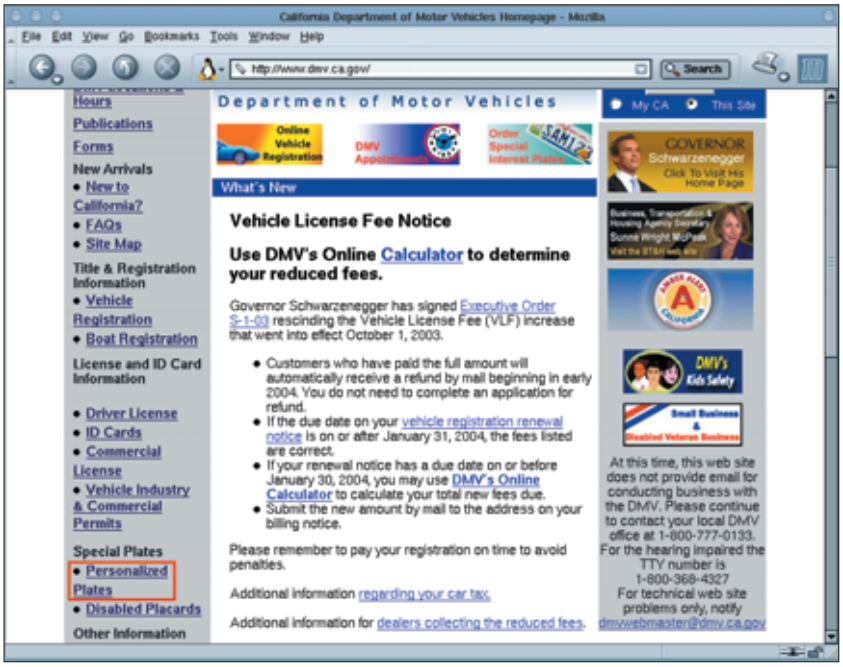

Abbildung 2: Die Eingangsseite des kalifornischen Department of Motor Vehicles. Der hervorgehobene Link führt zu einer Seite, auf der Kalifornier sich personalisierte Kfz-Kennzeichen erstellen lassen.

sich Anzahl und Reihenfolge der Links auf einer regelmäßig gewarteten Seite stetig ändern. Der Vorgang sollte daher allgemein gültiger ablaufen, damit das Skript auch dann noch funktionsfähig ist, wenn die Behörde Verweise auf der Seite einfügt oder entfernt.

Links suchen per Regex

Folgender Befehl sucht nach einem Link, dessen Textbeschreibung das Wort »Personalized« enthält, und simuliert dann einen Klick darauf:

```
>open /Personalized/
```

Passen auf den angegebenen regulären Ausdruck mehrere Links, stellt die Mechanize-Shell ein Auswahlmenü dar. Im vorliegenden Fall passt nur ein einziger, was die Shell folgendermaßen quittiert:

```
Found 15
(200)
```

Die nächste Seite enthält den Link »order Special Interest and Personalized license plates«. Die Suche nach diesem Text mit »open "/order Special Interest/"« führt zu einer Seite, auf der sich mehrere HTML-Formulare befinden. Bei dem vorigen »open«-Befehl ist noch zu beachten, dass der reguläre Ausdruck in Anführungszeichen stehen muss, weil er Leerzeichen enthält. Die Formulare lässt sich der Benutzer mit dem Kommando »forms« anzeigen:

```
>forms
...
Form [2]
POST https://vrir.dmv.ca.gov/ipp/ ?
PerLicensePlateServlet [personalized]
  page=Select                    (hidden)
  Submit2=Order Personalized     (submit)
...
```

Das zweite Formular, mit der Bezeichnung »personalized«, ist das hier relevante. Die Befehlsfolge »form "personalized"« und »submit« wählt es aus.

In einem weiteren Formular legt der Benutzer mit Drop-down-Menüs und Radio-Buttons Fahrzeugtyp, Nummernschildmuster und Ähnliches fest. Für so komplizierte Formulare eignet sich der Befehl »fillout«, der die Werte für die Formularfelder interaktiv vom Benutzer entgegennimmt. Dazu dient ein Dialog wie in Abbildung 3. Der Radio-Button »kidpic« bleibt mit einem Druck auf die [Enter]-Taste deaktiviert. Ein abschließender »submit«-Befehl veranlasst die Shell dazu, die eingestellten Werte an den Server zu schicken.

Auf der darauf folgenden Seite wählt der Anwender dann den Wunschtext seines Nummernschilds aus. Wie die Browser-Darstellung in Abbildung 4 zeigt, erfolgt dies wiederum über ein HTML-Formular, das die Buchstaben des Nummernschilds einzeln abfragt. Auch hier liefert ein »fillout« den nötigen Dialog und ein »submit« schickt die Daten.

Ra ru rick, Barbatrick!

Wie in der Barbapapa-Serie kommt nun der Clou: Um aus dieser Session ein an die privaten Erfordernisse angepasstes Skript zu generieren, genügt der Befehl »script« in der Shell – schon spuckt die treue Seele ein Perl-Skript aus, das genau das reproduziert, was man bisher in Handarbeit eingegeben hat.

Listing 1 ist auf diese Weise entstanden. Es ist jedoch an ein paar Stellen angepasst: So gibt der Anwender nun das zu untersuchende Nummernschild auf der Kommandozeile mit »dmv *Text*« an. Zeile 12 bricht ab, falls kein Parameter angegeben ist. Der Konstruktor des »WWW::Mechanize«-Objekts betreibt den Browser-Simulator mit gesetzter »autocheck«-Option in einem Modus, der das Programm sofort abbricht, falls ein Fehler auftritt, er also beispielsweise eine Webseite nicht findet.

Zeile 19 dirigiert den Simulator zur Eingangsseite. Die »follow()«-Methode in Zeile 20 springt wegen des angegebenen regulären Ausdrucks auf einen Link, der den Text »Personalized« enthält. Zeile 21 führt einen weiteren, Regex-gesteuerten Sprung durch. Zeile 23 wählt das Formular mit dem Bezeichner »personalized«

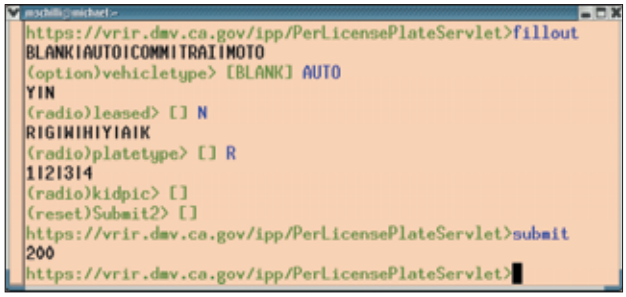

Abbildung 3: Die Shell des Moduls WWW::Mechanize::Shell verhält sich wie ein Webbrowser. Hier hat der Befehl »fillout« einen Dialog gestartet, in dem der Benutzer HTML-Formulare ausfüllt.

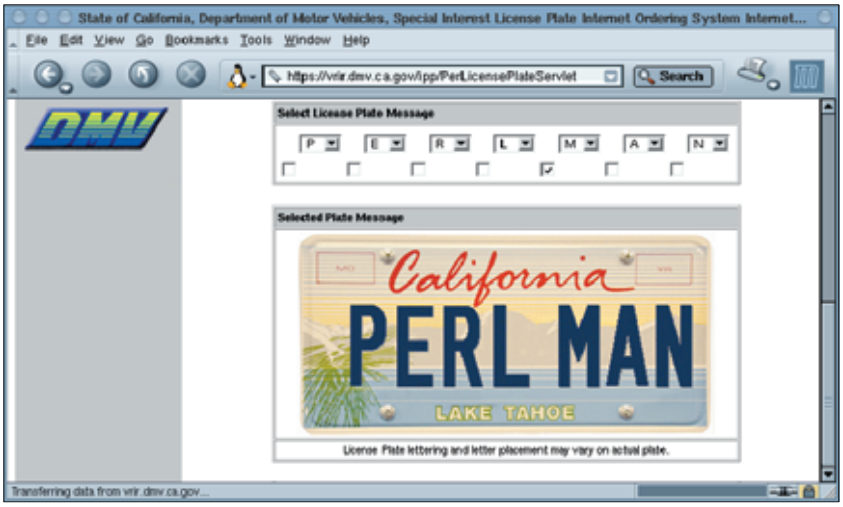

Abbildung 4: Wer sein Auto in Kalifornien gemeldet hat, kann sich sein künftiges Nummernschild auf der Website des Department of Motor Vehicles selbst zusammenstellen, inklusive des Schriftzugs.

aus und die darauf folgende »submit()«-Methode bedient den Submit-Knopf des ansonsten leeren Formulars.

Das in Zeile 17 erzeugte »WWW::Mechanize::FormFiller«-Objekt übernimmt das Ausfüllen des nun erscheinenden Formulars. Die »add_filler()«-Methoden ab Zeile 26 spezifizieren jeweils den Namen eines Felds, das Eingabeverfahren und den Wert. Für hartkodierte Werte findet »Fixed« Anwendung, während »Interactive« den Form-Filler-Code dazu bringt, den Benutzer zur Laufzeit nach dem entsprechenden Wert zu fragen.

Listing 1: »dmv«

```
01 #!/usr/bin/perl
02 ##########################################
03 # dmv -- Automate checking CA plates
04 # Mike Schilli, 2003 (m@perlmeister.com)
05 ##########################################
06 use strict;
07 use warnings;
08
09 use WWW::Mechanize;
10 use WWW::Mechanize::FormFiller;
11
12 die "usage: $0 XXXXXXX" unless
13                           defined $ARGV[0];
14 $ARGV[0] =~ s/\s+//g;
15 my $agent = WWW::Mechanize->new(
16               autocheck => 1);
17 my $fi = WWW::Mechanize::FormFiller->new();
18
19 $agent->get('http://www.dmv.ca.gov');
20 $agent->follow(qr(Personalized));
21 $agent->follow(
22      qr(order Special Interest));
23 $agent->form("personalized");
24 $agent->submit();
25
26 $fi->add_filler('vehicletype' =>
27                 Fixed => 'AUTO' );
28 $fi->add_filler('leased' =>
29                 Fixed => 'N' );
30 $fi->add_filler('platetype' =>
31                 Fixed => 'R' );
32 $fi->add_filler('kidpic' =>
33                 Fixed => '' );
34 $fi->add_filler('Submit2' =>
35                 Fixed => '' );
36 $fi->fill_form($agent->current_form);
37 $agent->submit();
38
39 for(0..6) {
40     $fi->add_filler("LicPltCharAry$_" =>
41          Fixed =>
42          $_ > length $ARGV[0] ?
43          "" : substr($ARGV[0], $_, 1));
44 }
45
46 for(0..6) {
47     $fi->add_filler("HalfSpace$_" =>
48                     Fixed => '');
49 }
50
51 $fi->add_filler('Submit2' => Fixed => '');
52 $fi->fill_form($agent->current_form);
53
54 $agent->submit();
55
56 if($agent->content() =~ /not available/) {
57     print "$ARGV[0]: not available\n";
58 } elsif($agent->content() =~
59                   /Complete Order Form/) {
60     print "$ARGV[0]: available\n";
61 } else {
62     print "Unexpected response",
63           $agent->content(), "\n";
64 }
```

Wenn der Form Filler weiß, woher er alle Werte für die Formularvariablen bekommt, kann's losgehen: Die »fill_form«-Methode startet mit dem gegenwärtig bearbeiteten »HTML::Form«-Objekt des »WWW::Mechanize«-Agenten den Ausfüllvorgang.

Das Skript steuert die Eingangsseite des DMV an, wühlt sich geduldig durch alle Links und Formulare und trägt schließlich in der For-Schleife ab Zeile 39 die angegebene Zeichenkette in die Auswahlboxen ein. Enthält das gewünschte Nummernschild weniger als sieben Buchstaben, setzt die Logik ab Zeile 42 einfach leere Felder ein. Der Code in Zeile 54 schickt die Daten per SSL an den Server und der Agent nimmt die Ergebnisseite als HTML entgegen.

Es kann nur einen geben

Steht dort etwas wie »not available«, ist die Kombination entweder schon vergeben oder wegen anstößigen Inhalts nicht verfügbar und das Skript gibt eine entsprechende Kurzmeldung aus. Wurde das Kennzeichen genehmigt, erscheint ein Bestellformular, welches das Skript anhand der Zeichenkette »Complete Order Form« erkennt und daraufhin die Meldung »*XXX*: available« ausspuckt. Die Ausgabe »PERLMAN: not available« bestätigt, dass PERL MAN nun leider schon vergeben und wohl auf lange Sicht nicht verfügbar ist. *(mwe)* ■

Infos

[1] Listings zu diesem Artikel: [ftp://ftp.linux-magazin.de/pub/listings/magazin/2004/03/Perl]

[2] Website des kalifornischen Department of Motorvehicles für die Auswahl von personalisierten Kennzeichen: [http://www.dmv.ca.gov]

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].