CGI::Auto – Automatic Web-Service Creation

Davide Sousa, Alberto Simões, and José João Almeida

Departamento de Informática
Universidade do Minho
`kevorkyan@gmail.com,{ambs,jj}@di.uminho.pt`

Abstract. The creation of a CGI or a WebService as an interface for a command line tool is not as unusual as it may seem. It is extremely usual and useful.
There are applications developed as command line tools that can be useful for different purposes, and different kind of users. Some of these users might not be able to run these tools directly. For instance, it is not easy to install a bunch of Perl modules to have a small tool working. For these situations, it is easier to make the tool available in the Web or as a WebService.
The problem with making the tool available in these fashions, is that programmers tend to rewrite the tools to incorporate the CGI or XML specific layers.
We defend that these CGI or WebService interfaces should use the already available command line tool, without any change. This interface should be able to read a simple textual specification of how the command line tool works, and buid the CGI or XML specific layers automatically.
The `CGI::Auto` module aims this purpose: to encapsulate command line tools in a CGI layer based on a textual specification, transforming the command line tool in a web application.

1 Introduction

When solving a problem in a quick way, programmers tend to write small command line tools. These commands (or scrips) can do a lot of interesting things, but their dissemination is not easy. People who want to run them have to install (and probably compile) the command.

The process of making these tools available to other users can involve two approaches:

– to prepare a standalone package, that can be downloaded over the Internet for local installation;
– to prepare a WebService [1] or an interactive Web Application for final-user usage.

While the first approach is preferred for most cases, it is not the easier way for all kind of users. If the tool was developed using, for instance, a scripting language, the final-user will need to install the scripting language interpreter to run it. If the programmer wants to distribute a binary it is needed to compile the tool for each architecture and operating system. Other option might be the distribution of the source-code. That is handy if the final-user is also a programmer, or else, she will be lost when compiling the tool.

When the command line tool is to be used by a non technical user, most programmers tend to choose the second approach to release their tools. To prepare a Web Application is quite easier, as the tool will run in the server computer, and thus, no cross compiling is necessary.

The problem with this approach is that while it is not difficult to develop a user-friendly web interface, it involves the modification of the original tool, adding a Web layer.

We might argue that this process is performed just once. But it is not true. When adding a new feature to the command line tool we will want to reflect it in the Web application. In the same way, when changing the Web application, we will want to reflect the changes in the command line tool.

We defend that the best approach is to develop a Web layer that uses the command line tool directly. The programmer can change the command line tool, and the new features will be available for the final-user automatically, with no need to re-design the Web Application.

For this purpose we developed a Perl module, named `CGI::Auto`. This module provides means to write a CGI[1] [3] application that behaves accordingly with a textual specification of how the command line tool is executed. `CGI::Auto` will not re-implement command line tools: it will run them directly on the host operating system shell.

This module uses a textual specification that describes how to run a specific command (or generically any pipeline of commands), and generates the Web application on the fly. The textual specification just describes the accepted flags and option for the command line tool, together with some strings to make the Web application more user-friendly.

`CGI::Auto` will be introduced, showing how to build a Web interface for two well known Unix command line tools: word count (`wc`) and pattern searching (`grep`).

2 `CGI::Auto` Approach

Before developing `CGI::Auto`, we needed to study how command line tools work. The main requirement for `CGI::Auto` is that it should be general enough to produce interfaces for all kind of command line tools. Command line tools can interact with the user in many different ways [2]:

– accept input through standard input;
– print errors to a standard error file handle;
– print results to the standard output;
– create files;

[1] Future work intends to incorporate a WebService layer as option.

– read files;
– accept optional flags or required ones;
– accept optional or required arguments;
– can be compound using a pipeline;

All these kind of operations need to be treated by `CGI::Auto`, and need to be defined in some way in the specification file, in a simple an concise description.

The `CGI::Auto` CGI have just one command: the invocation of the `CGI::Auto` main method with the textual specification of how the Web Application should interact with the command line tool.

The CGI will act as an abstraction layer between the command line tool and the Web Application. It will show a simple form to the user, where the command line tool available options are shown as common HTML widgets: radio buttons, combo boxes, text fields and text areas. When the user fills in the form and submits it, a command line pattern is filled with the user information and run on the host operating system. The output will then be presented to the user.

2.1 Example 1 — Word Count

A command line tool works with a number of specific parameters, and usually operates on one or more files. As a first example, consider the word count command (`wc`) which, among other things, allows the user to count the number of lines and the number of words in a given file. Its main usage is:

```
wc -l -w file
```

Basically, it accepts two switches that specify what we want to count (lines or words) and a filename. The two switches are shown to the user as HTML radio button, where the user can select one, both or none.

Regarding the file to process, we need to make it available in the host operating system. We can supply it to the Web application using three methods: to upload the file using a common CGI upload field, to copy and paste the content of the file into a text area, or to specify an URL where the file should be downloaded.

The specification to run this command under `CGI::Auto` is as simple as:

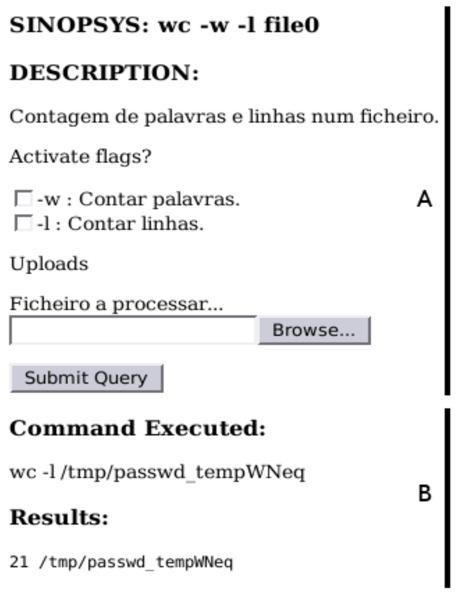
```
my %wc= (
  description => "Counting lines and words in a file.",
  command => "wc [%words%] [%lines%] [%file%]",
  args => {
    words => {type => "flag",value => "-w",name => "Count words"},
    lines => {type => "flag",value => "-l",name => "Count lines"},
    file  => {type => "upload",name => "File to process..."}
        }
);
```

The first argument on this specification is a string that describes the web application. Follows the command line pattern with placeholders, delimited by `[%`

and %]. These placeholders are then explained, one by one, in the arguments list.

These arguments can have different types. On this example we have flags (for command line switches), and upload fields (for supplying files to the command line). Flags have a value (the value that is replaced in the command line pattern if the user select it), and a name, that is shown to the user in the Web application. File uploads just have a name, with a simple description of what will be done with that file.

Figure 1 shows the CGI created with this configuration. The area **A** corresponds to the user interface with the command line tool. After activating the desired parameters and uploading the file to process, the command result is presented (area **B**).



SINOPSYS: wc -w -l file0

DESCRIPTION:

Contagem de palavras e linhas num ficheiro.

Activate flags?

☐ -w : Contar palavras.
☐ -l : Contar linhas.

Uploads

Ficheiro a processar...

Command Executed:

wc -l /tmp/passwd_tempWNeq

Results:

```
21 /tmp/passwd_tempWNeq
```

Fig. 1. Word-Count (wc) with a web interface.

This interface is not the best for all kind of operations, but is fully configurable by the `CGI::Auto` specification.

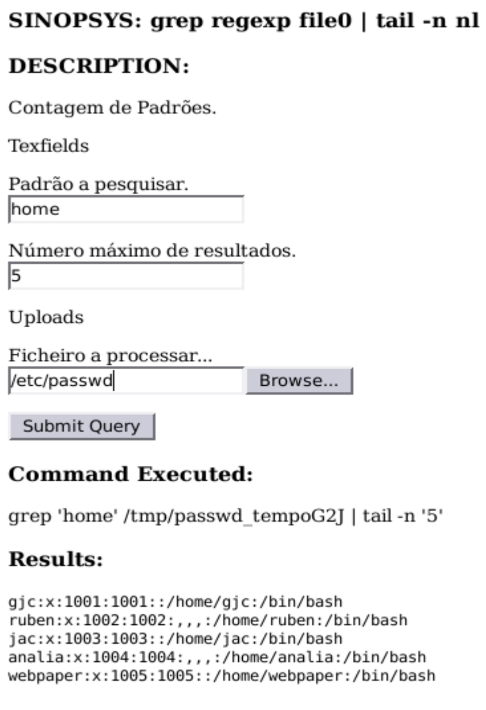
2.2 Example 2 — Pattern Search

The second example includes a third type of parameter: textual expressions to be replaced directly in the command line pattern.

To demonstrate this kind of parameter we will build an interface to the `grep` unix tool. Also, to explain how we can supply a pipeline of tools, we will tail the result of the grep, showing just the final n results (being n a value specified by the user).

```
my %grep= (
  command => "grep [%regexp%] [%file%] | tail -n [%nl%]",
  description => "Pattern Search.",
  args => {
    regexp => {type => "textfield",name => "Pattern to Search"},
    nl     => {type => "textfield",name => "Maximum number of results"},
    file   => {type => "upload"    ,name => "File to process.."}
  }
);
```

The specification is straightforward. The command is just another command line pattern. It is written as if it was executed directly on your shell. Regarding arguments, we have the third type: text fields. These arguments are fields where the user can write anything, and their content will be replaced in the command line pattern. Figure 2 shows the Web interface generated for this description.



SINOPSYS: grep regexp file0 | tail -n nl

DESCRIPTION:

Contagem de Padrões.

Texfields

Padrão a pesquisar.
home

Número máximo de resultados.
5

Uploads

Ficheiro a processar...
/etc/passwd

Command Executed:

grep 'home' /tmp/passwd_tempoG2J | tail -n '5'

Results:

```
gjc:x:1001:1001::/home/gjc:/bin/bash
ruben:x:1002:1002:,,,:/home/ruben:/bin/bash
jac:x:1003:1003::/home/jac:/bin/bash
analia:x:1004:1004:,,,:/home/analia:/bin/bash
webpaper:x:1005:1005::/home/webpaper:/bin/bash
```

Fig. 2. Pattern Search (grep) with a web interface.

3 Conclusion and Future work

While this work is still in progress, it shown that the approach is viable. It is quite easy to specify any command line behavior using a command line pattern and a set of arguments description.

There are a lot of new features that are being currently implemented. They include more flexibility in the user interface (for instance, to generate mime-type rich documents, like PDF), incremental pipeline execution, and batch jobs. This last feature is very important as some tools will make the web-server time-out before a result is shown. For these kind of tools we are developing a deferred approach. The user will fill in the form and supply an email. Later, when the job is finished, the user will receive an email with the job results (or for some jobs with big results, with a specific URL where the result can be downloaded).

At the moment we know that the interpolation of user input directly in the command line patter is a big source of exploits. At the moment we want to prepare the basic module, and later include a security layer to filter and validate user input.

Currently, the integration with WebServices is not yet a reality. While its implementation is not complicated, it requires the development of a WebService client. Without this client we can not validate it. With web applications this kind of validation is quite easier.

References

1. Carlos Jorge Lopes and José Carlos Ramalho. *Web Services — Aplicações Distribuídas sobre Protocolos Internet.* FCA, 2005.
2. Eric Steven Raymond. *The Art Of Unix Programming.* Addison-Wesley, 2003.
3. L.D. Stein. *Official guide to programming with CGI. pm.* Wiley New York, 1998.