Vision Statement for a Visual Chart Interpreter

Introduction

It has been my experience over the years that many computer users are quite happy to use arbitrarily complex GUI programs (see Blender for example), but are very nervous about using a command line program.

The UNIX desktop environments, from CDE on, have focussed on running GUI programs. The result is that the amazingly rich set of command line UNIX/Linux programs is now almost invisible from the desktop.

It is difficult for desktop users to use and combine these programs to automate their work. Many of us have seen office workers manually using a computer to perform the same tasks every day. It is often the case that these tasks can be automated with some simple shell scripting, but there is rarely the opportunity for experienced programmers to assist with this sort of work. The users have the motivation and probably the opportunity, but are often unaware that scripting is even a possibility.

The aim of this project is provide a set of tools that will give a novice user the ability to build and run scripts that can be used to automate some of their routine tasks.

In addition to actually constructing a script there are some other problems that must be overcome:

- There are literally thousands of programs available on a typical Linux computer. There must be some way for the novice user to select the programs required for their problem.
- Each program has its own set of parameters, and there is often very little that is common or standard in the selection of the identifiers for these parameters. Help in setting these parameters must be part of the solution.
- It must be possible to install the scripts into the desktop environment so that they can be used like any other program.

UNIX Command Line Programs

The UNIX command line programs were mostly designed with scripting in mind. The philosophy behind UNIX is to use small specialised programs that perform a single function, and to combine these into scripts to perform more complex, business specific, tasks.

The programs are usually designed so that the output of one (lines of text) can be easily used as the input of the next. It is common to find several programs combined into one of these pipelines in a typical script. For example to find the number of files in a directory one would do:

```
ls | wc -l
```

where the ls command (short for list) produces a list of the files in the current directory. The output of this command is then passed to the wc command which, when given the -l option, outputs the number of lines of input it was given.

Commands can also be run in either foreground or background mode. A program running in the foreground causes the script to wait until it completes. A program running in the background is started and then control returns immediately to the script.

The main issues facing the novice user is that there are thousands of commands available, with names that are frequently so abbreviated as to be meaningless, and these commands take various

options and parameters which are also difficult to remember. Even experienced script programmers usually need to review the parameters of the commands that they are using in their scripts.

Run Time Script Usage

A script for use in a graphical desktop is necessarily a little different from the usual command line script. For example, it needs to have a visual presence so that files can be passed to it from the file browser using drag-and-drop.

The interpreter for the script thus needs to have a menu, with at least an exit option. An option to change the current working directory would seem likely to be useful, as would an option to open a file browser. It would have other options which trigger the user's scripts. The program would also need to have a place where files can dropped.

Other user interactions with the script would be handled by programs like zenity run from within the script.

The editor would be responsible for installing the script into the standard desktop menu system and the usual association mechanism would execute the script using the interpreter when it was selected.

Command Wrappers

The lack of uniformity in the UNIX commands forces us to introduce some way of associating the command with its documentation, and providing a way of documenting the parameters both for human users and the editor program itself. An XML file, for each command, could provide this association.

The wrapper would contain the name of the command, a short description, the reference to the on-line documentation (and the type of that documentation), the symbol to use for the command within the graphical editor, and the details of each parameter.

The system would come with a collection of wrappers for common commands. Experienced users could provide new wrappers in response to user's needs.

Searching

The user needs to be able to locate suitable commands for use in their scripts. The editor needs to provide an indexed search capability over the short descriptions in the wrappers, as well as over the on-line documentation.

The editor might also usefully provide a simple cataloguing system where a hierarchy of classes of programs could be defined and the commands classified into one or more of these classes. The user could then search by selecting classes and getting a list of those commands that match. The search could be refined by allowing the union or intersection operation to be performed on the list of commands.

Building Scripts

The user should be able to build multiple “event scripts” with each one triggered by an event in the run-time interpreter. There would be two standard events – “start” and “drop”. Other events could be defined and added into the menu tree.

The script would be assembled by placing symbols into a diagram that has the appearance of a flow chart. This is a simple structure that is easily understood.

There would be some standard symbols representing conditional and loop constructs. A section of a chart could be “grouped” to form the equivalent of a function which could be re-used. Lines connecting the symbols would indicate the sequence in which the commands were to be executed and other lines would indicate the pipeline connection data flow between programs (and files). Other symbols would represent variables which could be assigned values during the execution of the script. It will also be possible to flag a group of commands to be run as a background task.

Test and Installation

The editor program will need to allow the user to test their script and monitor its behaviour. Hence it will need to be able to call the interpreter, and the interpreter will need a mode that allows its state to be monitored.

Once the user is satisfied with their script they will need to be able to install it into their desktop menu system.

Summary

A system comprising a visual editor and a graphical interpreter application is proposed. It aims to make the Linux command line programs readily accessible to desktop users and to allow those users to automate some of their routine tasks.