Batchboard: Unix Interaction for Everyone

Michael Smith, Computer Science
Faculty Mentor: John David N. Dionisio, PhD, Assistant Professor
Department of Electrical Engineering & Computer Science
Loyola Marymount University, Los Angeles

Project Questions
The Unix operating system, along with its associated set of applications, tools, and utilities, is one of the most mature software packages in use today. In its many incarnations both past and present, Unix enjoys wide use in universities, businesses, and networks.

However, Unix is virtually unknown in personal computing. Even in the cases of Linux and Mac OS X, both genuine Unix platforms that have less technical or advanced users, Unix is a background or foundation technology, neither seen nor heard, responsible primarily for a computer’s internal housekeeping and management but not for direct interactive use.

Thus, Unix programs exist below the radar of most users, despite their potential usefulness and relative maturity in an age when newer software can be unreliable and error-prone. Because their stream-oriented, modular paradigm does not fit the current window-menu-pointer conventions of personal computers, the general population never experiences these programs’ reliability, stability, maturity, flexibility, and power.

Is this technical device unavailable? Is it possible to design an interaction model that enables average personal computer users to see, explore, and utilize traditional Unix programs? The Batchboard project seeks to answer that question.

Background
A computer’s operating system is the suite of software services, utilities, and applications that performs its fundamental activities. Microsoft Windows is the best known and most widely used operating system, but it is by no means the most stable nor most mature one.

This distinction belongs to Unix, an operating system that was initially developed in the 1960s and has been continuously refined and improved since that time.

Unix is distinctive for many reasons beyond its maturity and stability. A broad spectrum of programs whose functions greatly exceed the core tasks of an operating system exists for Unix. In addition, these programs share a common paradigm that allows them to work together extremely well, without a priori knowledge of each other’s functionality. One can generally assume that Unix application will interoperate with other applications. If it does not, then such a program is said to violate Unix conventions.

Eric S. Raymond, one of the key proponents of the open source software movement, analyzes Unix’s design philosophy and culture in his book *The Art of Unix Programming*. Raymond concludes that Unix software, on both philosophical and technical levels, stands as the most effective, flexible, and reliable software model to date.

Despite such qualities, this “booboo of unchanged basics,” in Raymond’s words, remains unappreciated and undervalued by today’s typical computer user. One of its key strengths — its emphasis on providing “mechanism, not policy” — also results in its relative inaccessibility to non-technical users.

Similar Programs
The challenge of making Unix philosophy “friendlier” has caught the interest of many scientists and has produced a wide variety of projects over the years.

Spilk by Henry et al., proposed to convert the text-based Unix shell — the traditional method for invoking Unix functions — into a graphical interface. Bong built upon Spilk with Elif, which added a visual scripting environment, Uscape, for specifying chains of commands to be executed in sequence. Subsequently, V TEC (visual Unix-filter components) by Spinellis de-emphasized an interactive graphical shell and focused on a visual programming environment.

While these projects all propose a visual interface to Unix components, they are intended for programmers, not end users. Other projects do target the end user, although they emphasize different tools from the Unix software base. The MedaFile project focuses only on video-oriented utilities. Apple’s Automator is more general, but is focused less on Unix and more on the scriptable functions in Mac Internet and “digital lifestyle” products. Finally, Batchboard selected Unix tools to Automator. However, it requires custom programming for each individual Unix tool.

Enter Batchboard
Batchboard adds a new element to the category of visual Unix interfaces it seeks to represent available Unix tools in a database. The database is then read by a graphical user interface package that generates a display based on the properties stored in the database.

The database is initially built by the Batchboard Author program. For now, it is created manually, but a future element of Batchboard is an automated element that can scan the programs on a computer look for their instructions (called “man pages” in Unix terms), then build the database on its own.

The end-user Batchboard Action client then reads this database and allows the user to choose which Unix programs to invoke. The Batchboard Action also allows the user to connect these programs together, so that they work like Unix “pipes” on a text-based shell.

Batchboard Implementation
It turned out that Automator serves as a very viable platform for the Batchboard front end. This front end is implemented as an Automator Action, and can be built within the Automator environment. In addition, integration with Automaton has the potential of allowing Batchboard to interact with the other non-Unix actions available in Automator, further expanding its usefulness.

Batchboard Status Quo
Design and conceptualization of Batchboard has been completed, and an implementation is in its very early stages. The software is hosted on SourceForge at:

http://sourceforge.net/projects/batchboard

References


Jens Makholm, Unix and Tk, and Unix Tk, Key 4 is a Visual Shell, January 2002.

Jarkko Tarkka, Dylan’s Logging: Marcon’s Environment, September 1999.


References for Batchboard: