Automating Repetitive Tasks in User Interaction

Gabriel Barata, Tiago Guerreiro, Daniel Gonçalves
Dep. Engª. Informática, IST
Av. Rovisco Pais, 1000 Lisboa
gabriel.barata@ist.utl.pt, tjvg@vimmi.inesc-id.pt ,daniel.goncalves@inesc-id.pt

Abstract

Computer users constantly face situations where repetitive tasks emerge and there is no easy way to automate them. Although there are several application launchers currently available, they lack of automating power to face the uniqueness of the repetitive tasks that arise from everyday usage. Many times, users have to resort to scripting languages and macro recorders to perform these tasks. However, these means are too farfetched for a common user.

We propose an approach capable of monitoring user activity, learn which tasks are recurrent and suggest an automation capable of completing the repetitive task. All this is done without disturbing the user or requiring his intervention. Preliminary user tests show that our solution can help users perform at least 169% faster for simple tasks.

1. Introduction

Nowadays, computer users are constantly facing recurrent tasks arising from the everyday usage. Most of these tasks tend to be unique, making them hard to automate. Application Launchers, such as Launchy, Enso or Dash Command, are very popular nowadays, but they are only useful for automating well known tasks, such as launching applications and calculations. Often, users have to resort to scripting languages and macro recorders in order to automate recurrent tasks. However, this is out of reach of the regular user and not applicable to many cases.

A few approaches try to address these issues. SMARTedit [4,5] is a text editor which resorts to a machine learning algorithm to automate repetitive tasks. This algorithm, *Version Space Algebra* [3], allows the composition of more complex version spaces from simpler ones. However, SMARTedit adopts the macro recorder concept, requiring the user to start and stop the recording of the repetitive task.

The Adaptive Programming Environment [7,8], hereafter referred as APE, automates repetitive tasks on a programming environment. APE avoids the macro concept by adopting the *Implicit Programming by Example* technique [6], which consists of monitoring the user's activity and detect recurrent tasks without requiring his intervention. APE is constantly monitoring the user and uses the Karp-Miller-Rosenberg (KMR) algorithm [1] to detect repetitive patterns of actions and, therefore, infer which tasks are recurrent. However, APE only operates over a single application and is not able to detect conditional and variable loops.

In order to bridge these gaps, we developed a new system for Microsoft Windows, named Blaze, which is able to automate repetitive tasks without requiring any user intervention. Blaze operates over the whole operating system, covering any recurrent task the user may perform in the file-system or any application. To attain this, Blaze also adopts the Implicit Programming by Example technique, using a data mining algorithm to identify repetitive tasks.

2. The Blaze System

As many recurrent tasks on modern operating systems are related to launching the same applications over and over again, Blaze takes the form of an application launcher, offering the user the same expressive power once granted by command line interfaces.

Unlike other application launchers, Blaze features an advanced text prediction algorithm which allows it to tolerate typos. The user input is separated into text tokens and a string contention algorithm and the Levenshtein distance [2] are used to match them with the indexed items. If the user mistypes something, Blaze can still understand it (Figure 1).

Figure 1 – Blaze tolerating typos.

In order to automate recurrent tasks, Blaze presents three automation agents: the Observer, which is responsible for monitoring user activity; the Apprentice, whose duty is to detect recurrent tasks; and the Assistant, which is responsible for composing automations capable of completing the recurrent task and, non-intrusively, notify the user.

The Observer is capable of monitoring every *user action* regarding the file-system, the mouse and the keyboard. To each action is associated contextual information, related to the application in which it occurred. User actions may be combined in order to produce more complex user actions. For instance, if the user presses sequentially the keys "H", "E", "L", "L" and "O", the respective *key press* actions are compressed to a single *type* "hello" action. Moreover, to each action is associated an id and, to each id, is associated a set of *generalizations*. A generalization describes a possible relationship between two or more actions. For example, a generalization describing the relationship between "type «hello 1»" and "type «hello 2»" could be "type «hello *f(n)*»" in which *f(n)* describes the numeric sequence {1, 2, 3, …}.

The Apprentice is the one responsible for identifying which tasks are recurrent or not. As every action has an id and there are no more than 20 actions in memory at the same time, each id can be treated as a letter from an alphabet and, therefore, finding a repetitive pattern can be treated as a string search problem, more precisely, the Longest Repeated Substring (LRS) problem. The best way to find the longest non overlapping substring of ids is to build a suffix-tree [9] and find all of its deepest internal nodes.

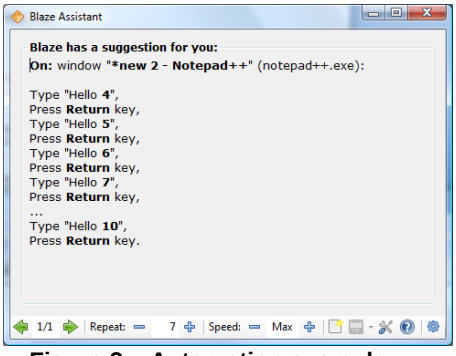

Figure 2 – Automation example.

The edge leading to one of these nodes represents a common prefix of suffixes of the input string. This allows us to keep track of all common prefixes of suffixes and pick the longest one as the longest non overlapping repeated substring. Although KMR would also be suitable for this task, the suffix-tree approach is more efficient, as it solves the problem in linear time and space.

Every time a repetition is detected, the Assistant uses the list of longest non overlapping substrings of ids and the list of generalizations to produce one or more sets of actions capable of automating the recurrent task. Moreover, in order to not disturb the user, it notifies him by lighting up the system tray icon and by displaying a button in the main interface. The user can choose to accept the suggestion or just ignore it. Figure 2 depicts the suggestion composed by Blaze after the user typed "Hello 1", "Hello 2" and "Hello 3".

3. Tests

In order to validate our approach we carried out both user and performance tests. Twenty users were asked to perform a set of 7 simple but typical tasks, and the time and number of errors was recorded. Most of the users had ages between 18 and 25 and used a computer daily, although only 50% of them were familiarized with *application launchers*. As shown in Chart 1, tests revealed that Blaze, on average, allows users to perform repetitive tasks 2.98 times faster. For more complicated tasks, involving the repetition of over larger sets of items, the gain would be even larger. Performance tests demonstrated that Blaze does not consume too much computational resources, presenting low CPU time usage and an average of 36 Megabytes of used memory, barely noticeable in a modern system.

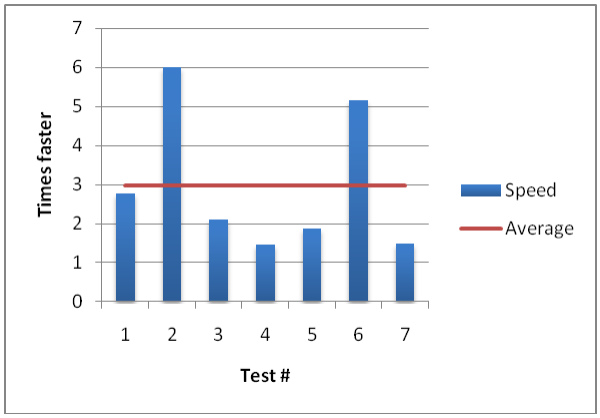

Chart 1 – User tests results.

4. References

[1] Karp, Richard M., et al. Rapid identification of repeated patterns in strings, trees and arrays. STOC '72: Proceedings of the fourth annual ACM symposium on Theory of computing, pp. 125-136. Denver, Colorado, USA, ACM, 1972

[2] Levenshtein, Vladimir. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics - Doklady, 10, pp. 707-710, February 1966

[3] Lau, Tessa, et al. Version Space Algebra and its Application to Programming by Demonstration. ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning, pp. 527-534. Stanford, California, USA, Morgan Kaufmann Publishers Inc., 2000

[4] Lau, Tessa, et al. Learning Repetitive Text-Editing Procedures with SMARTedit. In H. Lieberman, Your Wish is My Command, pp. 209-225. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2001

[5] Lau, Tessa, et al. Programming by demonstration using version space algebra. Machine Learning, 53 (1-2): pp. 111-156. Hingham, MA, Kluwer Academic Publishers, October 2003

[6] Ruvini, Jean-David. The Challenges of Implicit Programming by Example. IUI ’04, Madeira, Portugal, 2004

[7] Ruvini, Jean-David, Dony, Christophe. APE: learning user's habits to automate repetitive tasks. IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces, pp. 229-232. New Orleans, Louisiana, USA, ACM, 2000

[8] Ruvini, Jean-David, Dony, Christophe. Learning Users' Habits to Automate Repetitive Tasks, In H. Lieberman, Your Wish is My Command, pp. 271-295. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2001

[9] Stephen, Graham A. String Searching Algorithms, pp. 191-201. World Scientific, October 1994