Blaze - Automatizando a Interacção em Interfaces Gráficas

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

Abstract

Operating system users are constantly facing situations where repetitive tasks arise and there is no trivial way to automate them. It is true that there are several application launchers currently available, but they are no match to the uniqueness of the repetitive tasks that emerge from everyday usage. Most of the times, to maximize their performance, users struggle to find workarounds, resorting to scripting languages and macro editors. However, these are beyond the common user's knowledge, leading them to accomplish the remainder of the task by hand.

We present Blaze, a new system which is able to automate user's repetitive tasks, not in a single application but operating system-widely. Blaze brings the application launcher concept to a new level, enhancing other application launchers' features and adding the ability to be constantly monitoring users' actions, without interrupting their work. Blaze is able to establish relationships between actions and, resorting to an algorithm, based on common prefix search in a suffix-tree, detect repetition patterns in the user interaction history. These repetitions are used to dynamically generate comprehensive narratives and to fulfill the remaining of the task in the user's place. Furthermore, these automations can be stored in script files, which can later be reproduced, even in different situations from the one in which the repetition was detected.

Keywords

Interfaces adaptativas, programação implícita por exemplo, automatização de tarefas, sensibilidade ao contexto do utilizador.

1. INTRODUÇÃO

Os sistemas operativos, de hoje em dia, já se encontram bastante maduros e mostram-se bastante apelativos, fornecendo imensas ajudas interactivas para resolver os problemas mais comuns dos utilizadores. De facto, a evolução natural das interfaces gráficas de utilizador avança na direcção de sistemas que ofereçam, cada vez mais, pistas visuais e interactivas, para que o utilizador não tenha de memorizar nenhum comando. Contudo, este processo tem conduzido ao desenvolvimento de interfaces que têm vindo a perder o poder expressivo, outrora conferido pela antiga interface de linha de comandos. Apesar de obrigar os utilizares a memorizar conjuntos de comandos válidos, aquelas interfaces forneciam-lhes um controlo mais preciso sobre os ficheiros, concedendo mecanismos para manipular vários ficheiros de cada vez. Para além disso, ao fornecer capacidades de *scripting*, as interfaces de linhas de comandos acabam por suportar a automatização de tarefas repetitivas do utilizador.

Imagine que foi de férias no verão e que tirou cerca de cem fotografias, com a sua câmara digital. Após o regresso, está na altura de as descarregar para o PC. Após completar esta tarefa, repara que todas as fotos descarregadas possuem nomes estranhos, atribuídos automaticamente pela câmara, como por exemplo "SDC10050.JPG", "SDC10051.JPG", "SDC10052.JPG", etc. Desejando ter nomes mais significativos, como "Férias Verão 01.jpg", "Férias Verão 02.jpg", e por aí adiante, torna-se claro que precisa de mudar o nome de todas as fotos. Conforme as suas habilitações, no que toca ao sistema operativo que usa, pode ter à sua disposição várias formas de enfrentar o problema. Se for inexperiente, a sua única opção será mudar o nome de todas as fotos, uma por uma, ou tentar descobrir uma ferramenta apropriada na internet e gastar tanto tempo a tentar aprender a trabalhar com ela, como o que iria gastar a mudar o nome das fotos, uma a uma. Contudo, se for um perito com o sistema operativo e estiver familiarizado com linguagens de programação, poderá, com facilidade, fazer um *script* para completar a tarefa, iterando por todos os ficheiros JPG na respectiva pasta e mudando-lhes o nome. Por conseguinte, precisa de saber programar e ter algum conhecimento para conseguir realizar esta tarefa num período de tempo aceitável.

Como outro exemplo, imagine que precisa de organizar a pasta onde costuma guardar os ficheiros recebidos pela sua aplicação de *Instant Messaging*. Este tipo de pasta costuma ser bastante desorganizado, uma vez que recebe vários ficheiros com todo o tipo de nomes e extensões imagináveis. Você nota que recebeu bastantes relatórios da Maria Antonieta, pois ela costuma colocar sempre o seu nome algures no nome do ficheiro. Nota também que esses relatórios possuem extensões diferentes. Portanto,

você gostaria de mover todos os ficheiros cujo nome possui as porções de texto "Maria" e "Antonieta" para uma subpasta denominada "Maria Antonieta". Ainda assim, para que este problema possa ser resolvido num período aceitável de tempo, seria necessário que você fizesse um *script*. Só que desta vez, o problema é mais complicado, pois requer que sejam executas várias operações de *parsing* de *strings*. Construir um script destes iria consumir tanto tempo quando aquele dispendido a mover os ficheiros directamente para a subpasta, um por um.

Deixando o sistema de ficheiros de lado, existem várias outras situações interessantes onde o utilizador repete as mesmas acções, vezes sem conta. Por exemplo, quando um utilizador insere uma tabela num processador de texto, repetindo sempre a mesma sequencia de botões pressionados, ou até mesmo quando insere porções de texto sequenciais, como "Caso #1:", "Caso #2:", "Caso #3:", etc., onde, claramente, existe um padrão numérico que poderia ser facilmente automatizado. Para resolver o primeiro caso, poderia ser utilizador um gravador de macros, que não requer muito conhecimento acerca do sistema operativo, mas, ainda assim, necessita que se aprenda a trabalhar com ele. Para resolver o último caso, uma linguagem de *scripting* avançada poderia ser usada, como o AutoIt v3[1]. Não obstante, esta solução requer bastante conhecimento acerta de linguagens de programação Basic e da forma como o sistema operativo funciona. Isto está, claramente, fora do alcance da maioria dos utilizadores comuns. A maior parte deles não possui conhecimentos avançados sobre linguagens de programação nem tão pouco percebe como é que o sistema operativo funciona. Apenas sabem utilizá-lo.

Os exemplos acima citados demonstram claramente que, apesar de toda a evolução de que as interfaces dos sistemas operativos têm sido alvo, ainda existem vários problemas significativos, no que toca à automatização de tarefas repetitivas. Este problema força, normalmente, os utilizadores a recorrer às linguagens de *scripting* e às interfaces antigas, como a linha de comandos. Contudo, este tipo de conhecimento está aquém do alcance do utilizador comum de computadores.

Para resolver este problema, desenvolvemos uma aplicação que é capaz de automatizar tarefas repetitivas, no Microsoft Windows. Esta aplicação, denominada Blaze, assumiu a forma dum *application launcher*, implementado e melhorando as características dos seus homólogos.

A interface típica de um *application launcher* é adequada para este serviço, uma vez que fornece uma forma fácil e rápida de lançar uma função específica, com base numa porção de um comando do qual o utilizador se recorda. Desta forma, não só o utilizador não precisa de navegar por dezenas de menus, como também só precisa de se lembrar de parte dos nomes dos comandos.

[1] http://www.autoitscript.com/autoit3/ (Última visita: 01-06-2009)

Apesar de automatizarem tarefas essenciais, como lançar aplicações, fazer cálculos e pesquisar na Web, os *application launchers* comuns não são sensíveis ao contexto do utilizador. A *sensibilidade ao contexto do utilizador* é definida como a capacidade de monitorizar o comportamento do utilizador, enquanto ele utiliza o sistema operativo, e capturar informação valiosa acerca do que ele está a fazer, naquele preciso momento. Como exemplo deste tipo de informação, temos o URL do website ou o caminho do sistema de ficheiros que o utilizador está a visualizar, naquela altura; que ficheiros ele tem actualmente seleccionados; ou a que rede sem fios a que está ligado. Em conformidade, o Blaze apresenta vários mecanismos que facultam o acesso a este tipo de informação. Isto permite que repetições sejam detectadas, sobre a actividade monitorizada do utilizador, e que sejam construídas generalizações acerca de cada acção semelhante do utilizador. Estas generalizações conferem ao Blaze a capacidade de detectar acções sequenciais e, assim, oferecer-se para completar as tarefas repetitivas no lugar do utilizador. Contudo, esta sugestão é passiva e não interrompe o trabalho do utilizador nem tão pouco o distrai.

O Blaze oferece, também, capacidades avançadas de *scripting*, que apresentam duas funções principais: a) permitir que o utilizador guarde as repetições detectadas, para que possam mais tarde ser reproduzidas; b) permitir que o utilizador estenda facilmente a funcionalidade do Blaze, sem que tenha de compilar qualquer código, precisando apenas de um editor de texto comum.

Na próxima secção iremos descrever algumas abordagens, que exploram a automação de tarefas repetitivas, e analisar os *application launchers* actuais mais relevantes. Ao identificarmos as suas lacunas, pudemos construir uma base sólida, sobre a qual o desenvolvimento do Blaze assenta. Iremos também fazer uma breve descrição da arquitectura do Blaze e fornecer uma explicação detalhada forma como ele detecta os padrões de repetições e cria as generalizações. Para terminar, será efectuada uma breve avaliação de desempenho e uma discussão das principais conclusões e possível futuro trabalho.

2. TRABALHO RELACIONADO

Apesar de toda a evolução pela qual as interfaces dos sistemas operativos têm passado, é óbvio que muito poder expressivo tem-se perdido, anteriormente conferido pelas interfaces de linha de comandos. Esta perda criou uma lacuna nas interfaces de hoje em dia, que tem sido preenchida por *application launchers*. A falta de poder expressivo é tão notável que os sistemas operativos mais recentes já incluem os seus próprios sistemas para lançar aplicações, como por exemplo a Spotlight do Mac OS ou a pesquisa do menu Iniciar do Windows Vista.

Actualmente, existem vários *application launchers*, disponíveis para a maioria dos sistemas operativos, e a maior parte deles partilha um conjunto comum de características. Exemplos destas são o lançamento rápido de aplicações, cálculos numéricos, pesquisa na Web, navegação na Web e no sistema de ficheiros, completar auto-

maticamente comandos e caminhos e s navegação por ficheiros recentemente utilizados.

Contudo, existem alguns *application launchers* que se destacam dos demais, por oferecerem alguns mecanismos de automação mais avançados. Estes são alcançados através de meios para a gravação de macros, integração e manipulação do sistema operativo e aprendizagem interactiva de comandos. O ac'tivAid[2] e o Keybreeze[3] são dignos de serem mencionados, uma vez que oferecem meios para automatizar tarefas repetitivas básicas. O primeiro consegue executar *cronjobs* e o segundo oferece um sistema de gravação de macros decente. Apesar de estas funcionalidades poderem ser bastante úteis para utilizadores avançados, a maioria dos utilizadores comuns de computadores pode nem sequer conhecer o conceito de macro e, consequentemente, não estar apta a tirar proveito daquelas funcionalidades. Para além disso, o simples acto de gravar uma macro coloca o utilizador numa posição explícita de programador.

Existem também outros *application launchers* notáveis, como o Dash Command[4] e o Enso[5], devido a sua habilidade de se integrarem com outras aplicações e com o próprio sistema operativo. O Dash Command é bastante completo e integra-se bastante bem com varias aplicações para o Microsoft Windows. Fornece também um óptimo sistema de navegação, permitindo que o utilizador explore directorias e abra, comprima e, até mesmo, veja *previews* de ficheiros. Ademais, o Dash Command apresenta também um sistema de pesquisa na internet rico, integrando-se com vários motores de busca online. Por outro lado, o Enso fornece comandos que permitem ao utilizador minimizar, maximizar e alternar entre janelas. Consequentemente, o Enso apresenta uma boa integração com o Microsoft Windows. A maior desvantagem do Enso é o facto de os seus comandos se tornarem pouco práticos para uso corrente, uma vez que é necessário escrever "open" antes do nome de uma aplicação para a lançar.

É também interessante assinalar que, tanto o Dash Command como o Enso tem alguma sensibilidade ao contexto do utilizador. O primeiro é capaz de detectar que ficheiros e pastas estão actualmente seleccionados pelo utilizador, ou até mesmo que website está ele a ver. O último é capaz de executar operações sobre o texto actualmente seleccionado pelo utilizador, em quase todas as aplicações. Contudo, este mecanismo aparenta ser problemático, pois parece assentar em funções de acessibilidade, o que não constitui um veículo de troca de informação, independente da aplicação [Dix06].

Por fim, temos o Launchy[6]. Ele destaca-se dos restantes por ser simples, rápido, extensível e visualmente apelativo. Isto faz dele o *application launcher* de eleição por milhares de utilizadores.

Apesar destes *application launchers* poderem melhorar consideravelmente o desempenho do utilizador, faltam-lhes capacidades avançadas para automatizar as mais variadas tarefas repetitivas que surgem no dia-a-dia. Contudo, existem algumas abordagens que visam colmatar este problema, e que são dignas de serem referidas.

O SMARTedit [Lau01] [Lau03] consiste de um editor de texto que pode automatizar tarefas repetitivas, recorrendo a mecanismos de aprendizagem máquina baseados na procura em espaços de versões [Mitchell82]. Lau introduz-nos a Álgebra de Espaço de Versões [Lau00], que permite a composição de espaços de versões complexos a partir de outros mais simples. O SMARTedit começa por ter um espaço de versões hierárquico, contendo todos os programas possíveis que ele podssa aprender. Assim que o utilizador começa a gravar as suas acções, algumas funções vão sendo descartadas, pois deixam de ser consistentes com as acções observadas. Desta forma, à medida que o utilizador vai actuando, o espaço de versões vai-se tornando cada vez mais específico, permitindo que o SMARTedit efectue uma escolha, assim que a gravação estiver terminada. A acção mais provável é obtida capturando o estado actual da aplicação e tratando-o como input, executando cada função do espaço de versões sobre ele, e produzindo assim um conjunto de estados de output. Se o resultado for um estado único, então é apresentado ao utilizador com 100% de probabilidade de ser o que ele pretendia fazer. Se existir mais que um estado, então é seleccionado o que tiver a maior probabilidade.

É interessante assinalar que, para construir um espaço hierárquico de versões e compor espaços de versões complexos a partir de outros mais simples, o SMARTedit depende de conhecimento de domínio, tendo em conta a importância de cada conceito do domínio. O maior defeito do SMARTedit é o facto de se basear no conceito de macro, o que implica que o utilizador se envolva directamente no processo de programação.

Existem, contudo, alguns sistemas que colmatam este problema. Exemplos notáveis são o Eager [Cypher93] e o APE [Ruvini00] [Ruvini01]. O Eager é um assistente para o ambiente HyperCard, que é capaz de identificar as primeiras iterações de um ciclo, até uma determinada condição ser verificada. Por outro lado, o APE consiste de um assistente para um ambiente de programação, que é capaz de alargar a funcionalidade do Eager através da detecção de repetições, mesmo que estas não ocorram consecutivamente no histórico de acções. Ambos os sistemas diferem do SMARTedit, principalmente, por não necessitarem que o utilizador inicie a gravação de quaisquer acções. Eles estão constantemente a monitorizar o utilizador, sem interferir com as suas tarefas. Ambos se baseiam no conceito de Programação Implícita por Exemplo [Ruvini04], que consistem em aprender o que automatizar e a fazer uma sugestão, sem causar transtorno ao utilizador. Portanto, dois princípios básicos têm de ser tidos em conta: evitar a metáfora de macro, porque o uti-

[2] http://www.heise.de/ct/activaid/ (Última visita: 01-06-2009)

[3] http://www.keybreeze.com/ (Última visita: 01-06-2009)

[4] http://www.trydash.com/ (Última visita: 01-06-2009)

[5] http://www.humanized.com/ (Última visita: 01-06-2009)

[6] http://www.launchy.net/ (Última visita: 01-06-2009)

lizador não deve ser explicitamente envolvido no processo de programação, e adoptar a metáfora de Assistente, disponibilizando um agente que ofereça assistência sempre que uma repetição for detectada.

Conformemente, o APE é composto por três agentes: o Observador, o Aprendiz e o Assistente. O Observador monitoriza a actividade do utilizador e notifica os outros agentes sempre uma nova acção é detectada. O Aprendiz, por sua vez, detecta tarefas repetitivas e reconhece as situações em que elas ocorreram. Para detectar tarefas repetitivas, o APE utiliza o algoritmo KMR [Karp72]. Para detector as situações em que as repetições ocorreram, dois algoritmos são usados: um baseado em *árvores de decisão*, denominado C4.5 [Quinlan93], e outro baseado na *Eliminação de Candidatos*, denominado IDHYS [Ruvini00]. Por último, o Assistente é responsável por fazer corresponder as acções monitorizadas com os padrões de situação, gerados pelo Aprendiz, e verificar se existe alguma repetição associada. Se existir, então uma sugestão é feita ao utilizador, contudo, sem interromper os seus afazeres.

Apesar de implementar a metáfora de macro, o Creo [Faaborg06] é um sistema merecedor de ser referido. Ele automatiza tarefas repetitivas no Internet Explorer, recorrendo a um detector de dados, denominado Miro, que associa o conteúdo das páginas Web a objectivos de alto nível o utilizador. Por esse motivo, o Creo consegue gravar sequências de operações, realizadas na interface de utilizador que, subsequentemente, são generalizadas de acordo com tipos particulares de informação, detectados pelo Miro. Esta sequência pode, mais tarde, ser reproduzida mesmo que a nova situação não seja exactamente idêntica àquela na qual a gravação foi efectuada.

No que diz respeito à automatização das tarefas do utilizador, o nosso objectivo é idêntico ao dos sistemas acima mencionados, especialmente do APE: adoptar a Programação Implícita por Exemplo para aprender que tarefas devem ser automatizadas e quando deve uma sugestão ser feita, tudo sem perturbar o utilizador. Contudo, ao contrário dos mencionados, o nosso sistema alarga a capacidade de automação, não para uma única aplicação, mas para todo um sistema operativo. Por outro lado, isto conduz-nos a um novo obstáculo: deixamos de poder depender tanto do conhecimento de domínio, uma vez que existem demasiadas aplicações diferentes que podem ser usadas num sistema operativo.

No que toca à faceta de *application launcher*, o Blaze implementa a maior parte das funcionalidades dos seus homólogos, recorrendo à sensibilidade do contexto do utilizador para as elevar a um nível superior.

3. A CRIAÇÃO DE ALGO NOVO

Para colmatar as lacunas identificadas acima, criamos um novo sistema, denominado Blaze. O seu desenvolvimento assentou sobre 6 requisitos principais:

1. Uma interface expressiva e simples. Por isso, escolhemos a interface do *application launcher*, porque realiza os desejos do utilizador com apenas alguns *keystrokes*.
2. Ao escolhermos o *application launcher* como interface, temos elevar as suas funcionalidades para um novo nível.
3. O sistema tem de ser facilmente extensível.
4. O sistema tem estar apto a detectar repetições na experiência do utilizador e, consequentemente, oferecer-se para completar a tarefa no seu lugar.
5. A detecção de repetições e a sugestão não podem distrair o utilizador.
6. As automações sugeridas devem poder ser guardadas, para mais tarde poderem ser reproduzidas.

A interface do Blaze está ilustrada na Figura 1.

Figura 1 – A Interface do Blaze

3.1 Melhorar a Receita

A interface típica de um *application launcher* foi a nossa escolha porque ela oferece o poder expressivo conferido pela antiga linha de comandos, sendo mais apelativa visualmente e integrando-se melhor com os novos sistema operativo e com as novas interfaces gráficas.

O Blaze oferece a maior parte das funcionalidades dos seus homólogos, como por exemplo, o lançamento rápido de aplicações, navegação na Web e no sistema de ficheiros, completa automaticamente comandos e caminhos, efectua cálculos numéricos, pesquisa na Web, integração com a linha de comandos, e por ai adiante. Adicionalmente, oferece funcionalidades mais avançadas, como inserção rápida de texto em outras aplicações e a execução de operações sobre o texto seleccionado pelo utilizador. É também capaz de indexar os conteúdos dos ficheiros, como texto e *tags* ID3, no caso de ficheiros de música. As melhorias mais notáveis foram efectuadas em três áreas distintas: predição textual, sensibilidade ao contexto do utilizador e extensibilidade.

A predição textual é, provavelmente, a chave para o sucesso de um *application launcher*. Ela é responsável por equiparar o texto introduzido pelo utilizador com os comandos conhecidos pelo sistema. Contudo, os *application launchers* actuais não admitem que o utilizador dê erros ortográficos. Por exemplo, se o utilizador escrever "tanderbird" em vez de thunderbird, um *application launcher* típico não perceberá que o utilizar poderá querer lançar o Mozilla Thunderbird.

O Blaze supera este problema implementando um algoritmo rápido de predição textual, que recorre à métrica da distância de Levenshtein [Levenshtein66] para tolerar erros ortográficos.

O Blaze oferece maior sensibilidade ao contexto do utilizador, estando apto a recolher informação valiosa acerca

do que o utilizador está a fazer num preciso momento. Um exemplo desta informação é o caminho ou URL de uma pasta ou website que o utilizador esteja a visualizar, o caminho dos ficheiros ou pastas seleccionados pelo utilizador, o texto actualmente seleccionado ou até mesmo o SSID da rede sem fios a que ele está ligado. Por exemplo, o utilizador pode seleccionar um excerto de texto, de um artigo interessante que esteja a ler no seu navegador favorito, abrir o Blaze e escrever:

```
email "john.doe@provider.com" "check this out" !this
```

Daí, o seu cliente de *email* favorito abrir-se-á, com um email criado para o endereço "john.doe@provider.com", com "check this out" como assunto e o texto actualmente seleccionado como corpo. Como o leitor pode notar, "!this" é um comando contextual, que representa o texto actualmente seleccionado pelo utilizador, e que pode ser facilmente combinado com outros comandos do Blaze. Vários comandos contextuais estão à disposição do utilizador, como o "!this", "!clipboard", "!here", "!url" ou "!selected".

Em termos de extensibilidade, o Blaze disponibiliza duas formas de alargar o seu leque de funcionalidades: através de bibliotecas DLL, escritas em C#, ou através de *scripts* de texto, escritos em IronPython. A primeira forma permite que o utilizador adicione novos conjuntos de comandos e novas regras de indexação de ficheiros. A última, adiciona um novo comando por script, permitindo que seja feito o usufruto da poderosa .NET Framework, sem que seja necessário compilar uma linha de código.

3.2 Adicionar o Ingrediente Especial

A inovação por detrás do Blaze reside na capacidade de monitorizar todas as acções do utilizador, ao nível do sistema operativo. Para conseguir isto, recorremos a várias funções da API do Windows e da .NET, bem como a *system wide hooks* de rato e teclado. Um exemplo do tipo de acções que podem ser capturadas pelo Blaze é descrito na Figura 2.

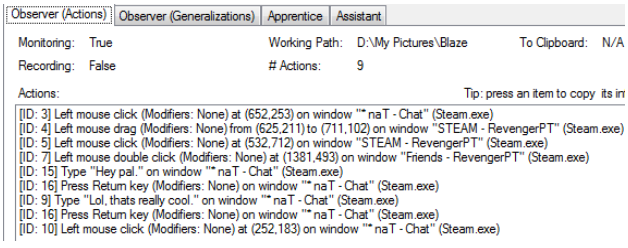

Figura 2 – Acções Monitorizadas na Janela de Debug

Como o APE, o Blaze possui três agentes: um Observador, um Aprendiz e um Assistente. O Observador utiliza os meios descritos acima para recolher informação sobre utilizador e a converter para objectos de domínio, denominados *Acções do Utilizador*. Estas acções podem ser combinadas entre elas para compor novas acções de alto nível. O Aprendiz, por sua vez, analisa a sequência de acções do utilizador, capturada pelo Observador, e detecta padrões de repetição. Para alcançar isto, é utilizado um algoritmo de *data-mining*. Deste modo, o Assistente verifica a existência de repetições detectadas pelo Aprendiz, e constrói um conjunto de sequências, adequadas para completar a tarefa repetitiva do utilizador. Estas sequências são sugeridas ao utilizador, de uma forma não intrusiva, mostrando um botão na interface no qual o utilizador pode clicar. Ao fazê-lo, a janela do Assistente do Blaze surge, mostrando uma ou várias narrativas que descrevem cada sugestão. Daí, o utilizador pode ordenar que a tarefa seja completada por si, pode gravar a automatização num script, que possa mais tarde ser reproduzido, e pode, até, modificar uma das sugestões, o que o conduz a um editor avançado de sequências de acções. O Assistente do Blaze está representado na Figura 3.

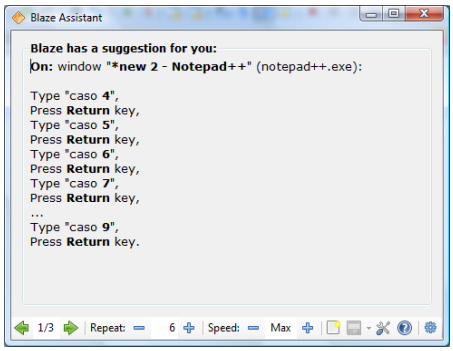

Figura 3 – Assistente do Blaze

4. POR DETRÁS DOS PANOS

Construir um sistema capaz de detectar repetições e de as automatizar, num sistema operativo, é bastante diferente de fazer o mesmo para uma única aplicação. Enquanto no último caso podemos ter ao nosso dispor informação sobre o domínio, acerca de cada acção que o utilizador possa desempenhar, no primeiro, nem sempre sabemos o que o utilizador pode vir a fazer. Por exemplo, o Blaze pode detectar que o utilizador está a pressionar o botão esquerdo do rato no Notepad++, mas não sabe se o utilizador está a mudar de separador ou a seleccionar texto ou, até mesmo, a ir ao menu de opções. Há demasiadas aplicações diferentes, suportadas pelos sistemas operativos actuais, que são desenvolvidas por diferentes pessoas e com diferentes regras. Isto torna difícil identificar cada acção que o utilizador possa fazer em cada uma delas.

4.1 O Observador

O problema que acabámos de mencionar consiste no maior desafio que o Observador tem de enfrentar: converter informação redundante acerca da actividade do utilizador, fornecida por diferentes fontes de monitorização, em informação fidedigna que possa, subsequentemente, ser processada pelos outros agentes.

O Observador tem três fontes de monitorização: 1) *system wide hooks* de teclado e rato, que levantam eventos sempre que o utilizador actuar sobre aqueles periféricos; 2) o observador do sistema de ficheiros, que desperta eventos sempre que o utilizador efectuar uma acção sobre determinados ficheiros ou directorias; e 3) o módulo de sensibilidade ao contexto do utilizador, que complementa a informação das outras fontes, com dados adicionais acerca da aplicação na qual os eventos ocorreram. Contudo,

estas diferentes fontes fornecem dados com diferentes níveis de granularidade. Por exemplo, quando um utilizador pressiona uma tecla, duas acções são geradas: uma acção para quando a tecla desce e outra para quando a tecla sobe. Por outro lado, quando é mudado o nome a um ficheiro, apenas uma acção é gerada. Não é aceitável deixar uma acção tão importante, como a de mudar o nome a um ficheiro, ser representada por um único objecto de domínio, enquanto uma mais vulgar, como a de pressionar uma tecla, é representada por dois objectos de domínio. Para colmatar esta lacuna, nos classificamos as acções em duas categorias: *acções de baixo nível* e *acções de alto nível*. As acções de baixo nível podem ser combinadas entre elas, para gerar outras de alto nível. Por exemplo, uma acção que represente a *descida* de uma tecla pode ser fundida com outra que represente a *subida* da mesma tecla, para gerar uma única acção que represente o *pressionar* da tecla. Ambas as acções de descida e subida são de baixo nível, enquanto a de pressionar é de alto nível. Não obstante, as acções de alto nível também podem ser combinadas entre elas, para gerar outras, também de alto nível. Por exemplo, uma acção de pressionar a tecla "O" pode ser fundida com outra de pressionar a tecla "K" para gerar uma acção de *redacção*, sendo "ok" o texto associado.

Sempre que o utilizador efectua uma acção, uma nova de baixo nível é gerada pelo Observador e adicionada ao histórico de acções. Mediante esta adição, é verificado se a nova acção e a última do histórico podem ser combinadas. Caso possam, então as duas são fundidas, gerando uma nova acção de alto nível. Daí, o mesmo procedimento é levado a cabo, ciclicamente, para verificar se a nova acção de alto nível pode ser combinada com outra anterior, parando apenas quando já não houverem combinações possíveis. Contudo, devido às nossas regras restritivas de fusão entre acções, sabemos à partida que não ocorrerão mais de três fusões consecutivas, partindo das mesmas duas acções.

Duas acções podem ser fundidas apenas se forem *similares*. Duas acções dizem-se similares se, e só se, for possível criar uma *generalização* entre as duas. Uma generalização consiste de uma expressão e um conjunto de funções, que descrevem o *relacionamento* que se estabelece entre duas acções. Um relacionamento tanto pode ser constante como sequencial. Cada acção do utilizador é identificada por um *id*, que não é único. Duas acções podem partilhar o mesmo id se, e só se, elas forem similares. Consequentemente, a cada id está associado um conjunto de generalizações.

Cada vez que duas acções se vão fundir, uma generalização é criada. Se não existirem generalizações anteriores associadas ao id da acção mais antiga, então a nova generalização é validada, e à nova acção é atribuído o id da acção antiga, resultando na fusão bem sucedida das duas acções. Caso já existam, então as novas generalizações e as antigas têm de ser fundidas. Se a fusão das generalizações for bem sucedida, então também a das acções o será.

Para clarificar o mecanismo por detrás da generalização, o seguinte exemplo deve ser tido em conta:

```
Observed Actions:

[ID: 13] Type "Hello 1"
[ID: 14] Press Return.
[ID: 13] Type "Hello 2"
[ID: 14] Press Return.

Generalizations:

[ID: 13] Hello §1, §1(n+1) = §1(n) + 1,
§1(n) = 2;
[ID: 14] repeat;
```

Como pode ser constatado, a primeira e terceira acções foram generalizadas, bem como a segunda e a quarta. No primeiro caso, a generalização resultante contém a expressão "`Hello §1`" e a função "`§1(n+1) = §1(n) + 1, §1(n) = 2`" onde "`n`" representa o número da iteração actual. Esta generalização traduz-se numa relação sequencial numérica entre as duas acções, e permite ao Blaze deduzir as próximas iterações. O leitor pode facilmente reparar, com este exemplo, que as generalizações são chave para inferir as iterações futuras do utilizador. No segundo caso, a generalização resultante apenas apresenta a expressão "`repeat`", porque "repetição" é a única relação que se pode estabelecer entre duas acções que representem o pressionar da mesma tecla.

A título de exemplo, apenas uma generalização foi mostrada por id. Contudo, no sistema real, várias generalizações podem ser geradas. As generalizações são criadas ordenadamente, de acordo com um critério preditivo. Por exemplo, perante duas mudanças de nome de ficheiro, uma generalização que traduza uma sequência numérica, tanto entre os nomes antigos, como entre os nomes novos dos ficheiros, terá uma maior probabilidade de ser válida do que uma que mostre que ambos os nomes antigos dos ficheiros apenas apresentam um conjunto comum de segmentos de texto. Este último caso apresenta uma maior probabilidade de ser puro acaso.

As generalizações são construídas com base no algoritmo da Diferença[7] [Myers86], na validação de expressões regulares e em operações aritméticas e de *strings*. Estas operações são aplicadas sobre dados específicos a cada tipo de acção do utilizador. Pegando no exemplo anterior, a generalização das acções 1 e 3 resulta da aplicação do algoritmo da Diferença e de uma procura por inteiros, usando uma expressão regular, sobre os campos de texto das duas acções de redacção. Daí, várias operações aritméticas e de *strings* foram aplicadas, com o intuito de gerar a expressão e as respectivas funções.

4.2 O Aprendiz

O Aprendiz é responsável pela tarefa de identificar sequências repetidas de acções, no historial de acções do utilizador, que por sua vez é mantido pelo Observador.

Dado que o nosso sistema não é intrusivo, não pode simplesmente interromper o utilizador para obter aprovação sobre que exemplos são positivos ou negativos. Portanto,

[7] http://www.mathertel.de/Diff/ (Última visita: 05-06-2009)

precisámos de desenvolver uma abordagem alternativa para identificar que exemplos são positivos, de uma tarefa adequada para ser automatizada. Como foi mencionado por Mahmud [Mahmud05] no seu relatório, uma boa forma de compensar a falta de exemplos negativos é tendo ocorrências repetidas dos exemplos positivos. Isto é, deveras, exactamente o que precisamos. As acções que o utilizador repete várias vezes têm um enorme potencial para serem integrantes de uma tarefa repetitiva, digna de ser automatizada. Por este motivo, precisámos de uma forma rápida de detectar sequências repetitivas no historial de acções do utilizador.

Assumindo que todas as acções têm um identificador e que o histórico tem um tamanho finito, 20 no caso da última versão, sabemos então que nunca teremos mais de 20 identificadores diferentes. Desta forma, o problema pode ser tratado com um de procura em *strings*, uma vez que cada identificador pode ser encarado como uma letra de um alfabeto.

O problema de procura descrito acima pode ser visto como o problema de achar a sub cadeia de caracteres repetida mais longa de uma *string*, ou também denominado, *Longest Repeated Substring problem* (LRS). Queremos assim, encontrar a sequência de identificadores mais longa, sem sobreposição, que ocorra pelo menos três vezes no histórico de acções. Daí, a melhor solução para resolver este problema consiste em construir uma árvore de sufixos[8] [Stephen94] e procurar todos os nós internos mais profundos. Todo o arco que conduz até um desses nós representa um prefixo comum de sufixos da *string* de entrada. Por isso, procuramos todos os prefixos comuns de sufixos, escolhendo o mais longo como sendo a sequência repetida de ids mais longa, sem sobreposição. Esta abordagem permite-nos detectar mais repetições e oferecer uma gama mais ampla de opções ao utilizador. Apesar de o KMR ser adequado para esta tarefa, a abordagem da árvore de sufixos é mais eficiente, pois consegue resolver o problema numa quantidade linear de tempo e espaço.

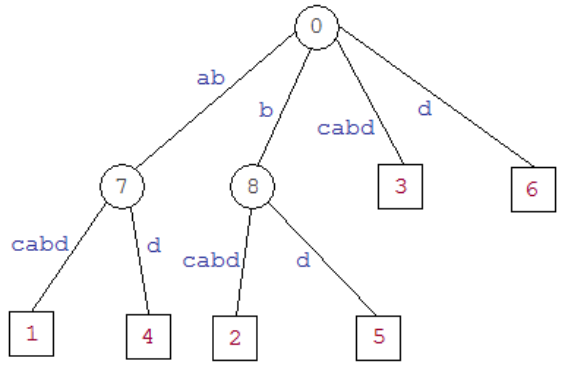

Figura 4 – Árvore de sufixos que representa "abcabd"

A Figura 4 retrata um exemplo de uma árvore de sufixos, que representa a *string* "abcabd". Cada arco está rotulado com um prefixo e cada caminho desde a raiz até a um nó folha corresponde a um possível sufixo da *string*. Note que os quadrados representam nós folha enquanto os círculos representam nós internos.

Imagine que a, b, c e d são identificadores distintos de acções do utilizador e que o histórico de acções é composto pela sequência "abcabd", como no exemplo acima. Para encontrar a *substring* repetida mais longa, sem sobreposição, uma pesquisa em profundidade tem de ser efectuada, indo tão fundo quanto o nó interno mais fundo, em cada caminho. Cada arco que conduz a um desses nós representa o prefixo comum de sufixos mais fundo, no respectivo caminho, e cada um desses prefixos é um candidato a ser a sequência repetida mais longa, sem sobreposição. Logo, é trivial perceber que tanto "ab" como "b" são prefixos comuns, pois o primeiro é descrito pelo arco que conduz ao nó 7, que é o nó interno mais fundo no seu caminho, e o segundo é descrito pelo arco que conduz ao nó interno 8, que é também o mais fundo no seu caminho. Conformemente, sendo tanto "ab" e "b" adequados, "ab" é escolhido como melhor opção, pois é o mais longo. Este algoritmo prova, assim, ser ideal para encontrar a sequência repetida mais longa, sem sobreposição, de identificadores de acções. Ademais, permite que o Blaze detecte as repetições mesmo que estas não ocorram consecutivamente no historial de acções.

O Aprendiz, por sua vez, mantém uma lista de repetições, ordenada por ordem decrescente de comprimento e data-que, juntamente com a informação fornecida pelo Observador, permitem ao Assistente gerar narrativas consistentes, descrevendo automatizações para as tarefas repetitivas.

4.3 O Assistente

Agora que já temos um agente que sabe o que automatizar, precisamos que um que saiba quando sugerir. A sugestão tem de ser feita de forma não intrusiva, pois o utilizador não deve ser interrompido do que estiver a fazer. Por isso, a responsabilidade desta tarefa recai sobre o Assistente, mostrando um ícone no canto superior direito da interface, no qual o utilizador pode simplesmente carregar se quiser ver as sugestões, ou ignorar se não quiser ser incomodado.

O ícone é mostrado sempre que existe, pelo menos, uma sugestão detectada pelo Aprendiz. Clicar no ícone irá conduzir o utilizador para uma nova janela onde, no máximo, 4 sugestões serão mostradas. Cada sugestão é descrita por uma narrativa que corresponde a uma repetição detectada pelo Aprendiz, que foi subsequentemente *processada* pelo Assistente. As narrativas são geradas com base nas generalizações mais prováveis de cada acção que componha a repetição em causa.

Tenha em conta a seguinte repetição, detectada pelo Aprendiz:

```
Observed Actions:

[ID: 3] Type "Hello 1" on window "new 2 -
Notepad++" (notepad++.exe)
```

[8] http://www.allisons.org/ll/AlgDS/Tree/Suffix/ (Última visita: 16-06-2009)

```
[ID: 17] Press Return key (Modifiers:
None) on window "*new 2 - Notepad++"
(notepad++.exe)
[ID: 3] Type "Hello 2" on window "new 2 -
Notepad++" (notepad++.exe)
[ID: 17] Press Return key (Modifiers:
None) on window "*new 2 - Notepad++"
(notepad++.exe)
[ID: 3] Type "Hello 3" on window "new 2 -
Notepad++" (notepad++.exe)
[ID: 17] Press Return key (Modifiers:
None) on window "*new 2 - Notepad++"
(notepad++.exe)

Generalizations:

[ID: 3] Hello §1, §1(n+1) = §1(n) + 1,
§1(n) = 3; Hello §1, §1(n+1) = §1(n) + 1,
§1(n) = '3';
[ID: 17] repeat;

Repetitions:

[3, 17]
```

Ao ser detectada, esta repetição é *processada* pelo Assistente. Uma repetição diz-se processada quando o Assistente escolha a generalização mais provável de cada acção e prediz as próximas iterações do utilizador. Como as generalizações já se encontram ordenadas, aquando associadas a um identificador específico, o Assistente apenas precisa de escolher a primeira da lista. Consequentemente, no exemplo acima citado, o Assistente escolhe a generalização mais provável para as acções 3 e 17. Apesar da acção 17 ter apenas uma generalização, a acção 3 tem duas distintas. A primeira, mais provável, encara o segmento de texto alterado da *string* como sendo um inteiro, enquanto a segunda encara o mesmo segmento como sendo um caracter. Por conseguinte, o Assistente recorre à expressão e às funções de cada generalização para gerar os valores seguintes, conseguindo assim construir a seguinte a narrativa descrita na Figura 5.

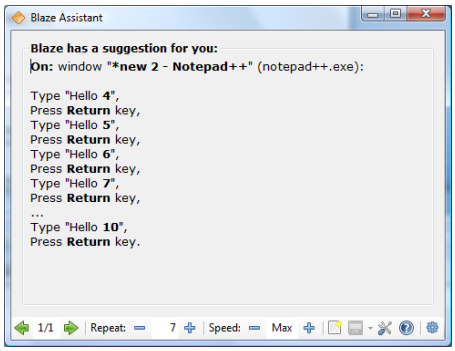

Figura 5 – Exemplo de uma Narrativa

Contudo, existe outro grande problema com o qual Assistente tem de lidar: Quantas iterações deverão ser executadas, para um dada repetição? Como não podemos depender muito no conhecimento de domínio, não temos uma forma trivial de responder a esta pergunta. Para a maioria dos casos, como o descrito no exemplo acima, deixamos o utilizador especificar a quantidade de iterações, através da interface do Assistente. Com o objectivo de elucidar o utilizador, sempre que o número de iterações é alterado por ele, a narrativa é também actualizada, mostrando o efeito da alteração.

Em casos mais complexos, como aqueles em que são efectuadas operações sobre o sistema ficheiros, recorremos à informação proveniente do contexto do utilizador para identificar que ficheiros ou pastas estão a ser manipulados e quantas mais operações são precisas efectuar para completar a tarefa repetitiva. Imagine que recebe os ficheiros do seu cliente de mensagens instantâneas na pasta "C\IM", e que os ficheiros nela contidos contêm no nome a porção de texto "mary". Você gostaria de organizar esta pasta e enviar todos esses ficheiros para uma subpasta. Para isto, bastaria mover pelo menos 3 ficheiros que respeitassem a regra e o Blaze irá inferir o resto. Neste caso, a generalização mais apropriada para uma sugestão é a seguinte:

```
Observed Actions:

[ID: 15] File 'XPTO notes from Mary
Jane.txt' moved from folder 'C:\IM' to
folder C:\IM\Mary Jane.
[ID: 15] File 'Evil Mary.jpg' moved from
folder 'C:\IM' to folder 'C:\IM\Mary
Jane'.
[ID: 15] File 'report mary 09.doc' moved
from folder 'C:\IM' to folder 'C:\IM\Mary
Jane'.

Best Generalization:

[ID: 15] §1, §1(n) = C:\IM\*(.*) containg
"mary".
```

Esta generalização indica que todos os ficheiros na pasta "C:\IM", que contenham o segmento de texto "mary" no nome, são apropriados para serem movidos para a pasta "C:\IM\Mary Jane", independentemente da sua extensão. Consequentemente, o Assistente só precisar de pedir ao módulo de contexto do utilizador para reunir o nome de todos os ficheiros contidos na pasta "C\IM", e verificar quais deles contêm aqueles segmentos de texto. Os que satisfizerem este condição são, então, adicionados a uma lista. Torna-se evidente que apenas uma iteração deverá ser executada sobre cada elemento da lista, deixando de ser necessário perguntar ao utilizador pelo número de iterações que devem ser efectuadas.

É também interessante assinalar que o Assistente permite que o utilizar modifique cada sugestão, no caso da narrativa mostrada não descrever correctamente a tarefa que ele estava a desempenhar. O Assistente mostra, juntamente com cada sugestão, um botão de "Modificar", que ao ser pressionado, leva o utilizador para uma nova janela. Nesta, o utilizador pode alterar as acções que devem ser executadas, com base nas restantes generalizações que não foram anteriormente escolhidas como mais prováveis.

4.4 Juntando as peças todas

Apesar de termos três agentes, o Observador é o único que está constantemente a monitorizar o utilizador. Sempre que o utilizador efectua uma acção, um evento é levantado, o que leva o Observador a gerar o objecto de domínio de baixo nível correspondente. Este agente tenta, então, fundir a nova acção com as já registadas, atribuin-

do-lhe um novo identificador e um conjunto de generalizações. Seguidamente, o Observador avisa o Aprendiz de que uma acção foi efectuada pelo utilizador. Daí, o Aprendiz constrói uma árvore de sufixos, com base no histórico de acções do Observador, e percorre-a, identificando as sub sequências repetidas mais longas de acções do utilizador. Seguidamente, o Aprendiz notifica o Assistente que, por sua vez, processa as repetições identificadas e constrói as respectivas sugestões.

Em suma, todas as operações, incluindo as respeitantes à validação periódica de acções, são apenas executadas quando um evento é levantado, o que nos permite poupar bastante tempo de processamento e, consequentemente, ter um algoritmo rápido.

5. AVALIAÇÃO DE DESEMPENHO

Levámos a cabo alguns testes de performance, que demonstram que a capacidade que o Blaze tem de monitorizar as acções do utilizador não afecta significativamente o desempenho da máquina, não prejudicando também, por isso, a experiência do utilizador.

O procedimento de teste foi o seguinte: durante 15 minutos o Blaze foi exposto ao que consideramos ser uma utilização típica por um utilizador de *application launchers* regular. O teste foi constituído do lançamento de algumas aplicações, efectuação de cálculos, e ordenação de algumas linhas de texto no bloco de notas. Além disso, o formulário de opções foi também acedido 4 vezes, em que numa delas o utilizador adicionou uma nova directoria para indexação.

Como todos os application launchers, o Blaze também indexa as aplicações do utilizador e, por isso, precisa de manter um índice com essas entradas. Ao total, o índice foi construído 4 vezes. O Assistente do Blaze foi também utilizado uma vez, para automatizar uma tarefa repetitiva de inserção de texto no bloco de notas.

Os testes efectuados cobrem tanto a utilização do CPU como a quantidade de vezes que cada método relevante do Blaze foi executado. Por método relevante consideramos aqueles que são propícios a consumirem bastante tempo de processamento, quer por serem executados muitas vezes ou por utilizarem rotinas dispendiosas.

A Figura 6 mostra que a maior parte da utilização do CPU é inerente a métodos de exibição de formulários, como é o caso do Open Settings e do Show Blaze. 9.1% e 7.2% são considerados baixos níveis de utilização de CPU, tendo em conta que estes métodos só são executados por ordem do utilizador. O terceiro método mais pesado é o Predicting Text. Este é executado sempre que o utilizador introduz texto no Blaze. Ainda assim, tendo em conta que este método só e executado a pedido do utilizador, 2.9% é um consumo muito reduzido de poder computacional. O utilizador pode facilmente constatar, com base na Figure 7, que o referido método é executado 199 vezes, logo os 2.9% de utilização de CPU são divididos por todas as chamadas.

A maioria dos outros métodos são executados em segundo plano e, por isso, têm maior risco de degradar a performance do computador. Contudo, a informação combinada dos dois gráficos permite-nos verifica que a utilização do CPU desses métodos é mínima, considerando a quantidade elevada de vezes que eles são executados. Portanto, não há uma combinação possível destes métodos que possa afectar, de forma considerável, a performance da máquina.

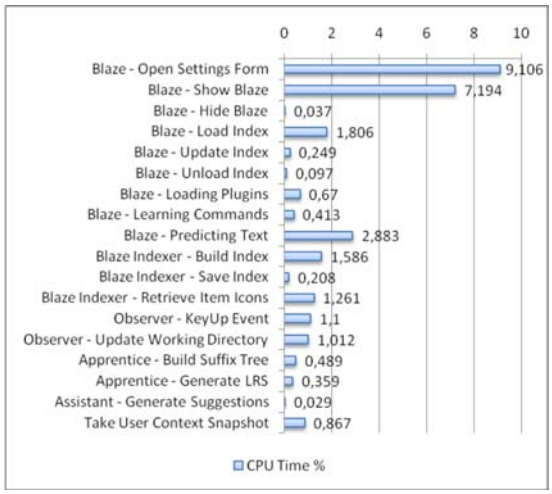

Figura 6 – Utilização de Tempo do CPU em Percentagem

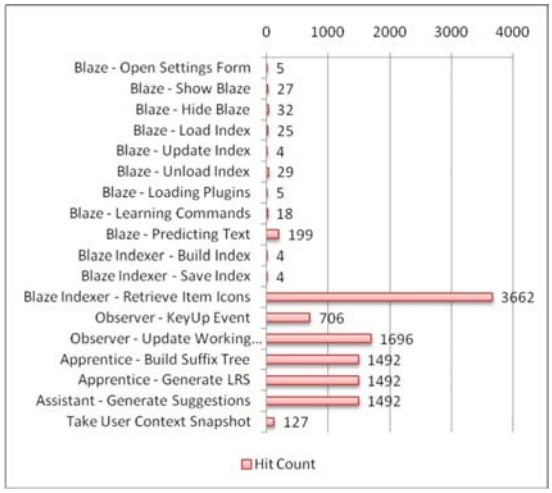

Figure 7 – Número de Chamadas a Cada Método

As operações mais pesadas que correm em plano de fundo são, sem dúvida, as relacionadas com a indexação. Apesar de elas não serem propícias a tornar o sistema operativo lento, elas podem afectar o próprio desempenho do Blaze, enquanto o utilizador o estiver a utilizar directamente. Por esta razão, confinámos as funções de indexação a um executável único, que é periodicamente chamado pelo Blaze. Para além disso, enquanto um novo índice é construído, o Blaze mantém uma cópia do antigo. Desta forma, a indexação não afecta a performance do Blaze e o utilizador tem sempre os itens indexados disponíveis.

Relativamente ao consumo de memória, tendo em conta o *Private Working Set*, o Blaze consome cerca de 33 megabytes. Apesar desta quantidade poder parecer excessiva, quando comparada com o consumo de outros *application launchers* simples, como o Launchy, que consome

cerca de 5 MB, temos de considerar que o Blaze está constantemente a monitorizar o utilizador e que, por isso, precisa de guardar muita informação. Por outro lado, quando comparado com outras aplicações, como o Mozilla Firefox, que consome cerca de 105 MB com apenas 7 separadores abertos, encaramos os 33 MB médios do Blaze como sendo bastante razoáveis.

6. CONCLUSÕES

Os utilizadores dos sistemas operativos, de hoje em dia, mostram a necessidade de uma maior poder expressivo. Este poder, anteriormente conferido pelas interfaces de linha de comandos, tem-se perdido com a evolução das interfaces gráficas. Para preencher esta lacuna, um novo tipo de aplicações surgiu, denominado *application launchers*. Os *application launchers* actuais podem simplificar bastante a vida do utilizador, mas não são capazes de detectar quando este está a efectuar uma tarefa repetitiva nem tão pouco de a completar por si.

Existem algumas abordagens, que recorrem à Programação Implícita por Exemplo, para aprender que tarefas devem ser automatizadas e quando deve uma sugestão ser feita ao utilizador. Apesar destas abordagens se basearem em sistemas que apenas automatizam as tarefas repetitivas do utilizador numa única aplicação, nos desenvolvemos um novo sistema, denominado Blaze, que é capaz de automatizar a experiencia do utilizador em todo o sistema operativo.

A nossa solução consiste de um *application launcher*, que eleva as potencialidades dos seus homólogos a um novo nível, e que oferece capacidades avançadas para capturar o contexto do utilizador. Consequentemente, o Blaze é capaz de monitorizar as acções do utilizador, por todo o sistema operativo, e detectar determinados tipos de relacionamentos entre elas, denominados generalizações. Daí, um algoritmo baseado em árvores de sufixos é utilizado para identificar a sequência repetida mais longa de acções, sem sobreposição, a partir do histórico de acções. Estas repetições, combinadas com as generalizações associadas a cada acção, permitem que o Blaze infira as próximas acções do utilizador e possa formular sugestões.

É interessante denotar que o utilizador nunca é interrompido da sua actividade, durante todo o processo de identificação de tarefas repetitivas, e por isso, ele não participa activamente no processo de programação. Além disso, os nossos testes mostram que o Blaze consegue monitorizar o utilizador utilizando uma quantidade de recursos computacionais razoável, provando ter uma performance aceitável.

No futuro, iremos ampliar as capacidades de captura de contexto do Blaze e aumentar a quantidade de padrões de repetições que podem ser detectados. Iremos também efectuar testes com utilizadores, o que irá mostrar o quão adequados os padrões detectados são perante situações do dia-a-dia.

7. REFERÊNCIAS

[Cypher93] Cypher, Allen, Watch What I Do: Programming by Demonstration, *MIT Press*, 1993

[Dix06] Dix, Alan, et al. Intelligent context-sensitive interactions on desktop and the web, *CAI '06: Proceedings of the international workshop in conjunction with AVI 2006 on Context in advanced interfaces*, *ACM*, 2006, 23-27

[Faaborg06] Faaborg, Alexander, Lieberman, Henry, A goal-oriented web browser, *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, *ACM*, 2006, 751-760

[Karp72] 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*, *ACM*, 1972, 125-136

[Lau00] 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*, 2003, 527-534

[Lau01] Lau, Tessa, et al., Learning Repetitive Text-Editing Procedures with SMARTedit, In H. Lieberman, *Your Wish is My Command.* Morgan Kaufmann, 2001, 209-225

[Lau03] Lau, Tessa, et al. Programming by demonstration using version space algebra, *Machine Learning*, 53 (1-2), October 2003, 111-156

[Levenshtein66] Levenshtein, Vladimir, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, *Soviet Physics - Doklady*, February 1966, 707-710

[Mahmud05] Mahmud, Jalal, A Survey of Inductive Learning, *Research Proficiency Exam Report*, Stony Brook University, CS Department, December 2005, 12

[Mitchell82] Mitchell, Tom M., et al. Generalization as Search, *Artif. Intell.*, 18 (2), 1982, 203-226

[Myers86] Myers, Eugene W., An O (ND) difference algorithm and its variations, *Algorithmica Vol.1 No.2*, 1986, 251-266

[Quinlan93] Quinlan, J. Ross, C4.5: programs for machine learning, *Morgan Kaufmann Publishers Inc.*, 1993

[Ruvini00] 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*, *ACM*, 2000, 229-232

[Ruvini01] Ruvini, Jean-David, Dony, Christophe, Learning Users' Habits to Automate Repetitive Tasks, In H. Lieberman, *Your Wish is My Command.* Morgan Kaufmann, 2001, 271-295

[Ruvini04] Ruvini, Jean-David, The Challenges of Implicit Programming by Example, *IUI04,* Madeira, Portugal, 2004

[Stephen94] Stephen, Graham A., String Searching Algorithms, *National Academy Press*, October 1994, 191-201