Departamento de Ciências e Tecnologias da Informação

Redefinição de Interfaces Gráficas

Rita Carolina Castelo Gama Maia

Dissertação submetida como requisito parcial para obtenção do grau de
Mestre em Engenharia Informática

Orientador(a):
Professor José Luís Silva,
ISCTE-IUL

Co-Orientador(a):
Professor João Carlos Silva,
Instituto Politécnico do
Cávado e do Ave

Julho, 2019

Agradecimentos

Quero agradecer em primeiro lugar às pessoas mais importantes da minha vida, a minha família, por todo o apoio e força que me deram ao longo de todo este percurso.

De seguida quero agradecer ao meu namorado também por todo o apoio que me deu e por nunca me fazer desistir de completar esta etapa, por muitos obstáculos que se colocaram pelo caminho.

E por último, mas não menos importante, quero agradecer ao meu orientador, José Luís Silva, por todo o apoio e auxílio que me prestou ao longo da realização deste trabalho e por toda a paciência para que teve comigo.

Resumo

Hoje em dia existem vários sistemas de computação interativa que ainda possuem Interfaces Gráficas do Utilizador (GUIs) inadequadas em termos de usabilidade e experiência para o utilizador. Inúmeras melhorias foram feitas no desenvolvimento de novas GUIs, no entanto, pouco foi feito para melhorar as já existentes. Isso pode ser explicado pelo fato de que a maioria dos sistemas de computação interativa, não fornecer acesso ao código-fonte, restringindo aos proprietários a introdução de melhorias nos mesmos.

Esta dissertação apresenta o desenvolvimento de uma ferramenta capaz de redefinir semi-automaticamente (sem acesso ao código fonte) interfaces gráficas, através de algoritmos de visão computacional, tornado-a mais apelativa e com maior usabilidade. A ferramenta é capaz de reconhecer um subconjunto de elementos da interface, que tenham determinadas características.

A avaliação da redefinição da nova GUI comparativamente à antiga usando a ferramenta irá ser descrita. E os resultados finais demonstram estatisticamente que a nova GUI reduz significativamente o número de erros cometidos, tem uma melhor taxa de sucesso na conclusão das tarefas e ainda uma melhor satisfação do utilizador, que foi comprovado através da realização de um questionário sobre a ferramenta.

Palavras-Chave: Interface Gráfica do Utilizador; OpenCV; Usabilidade; Processamento de Imagem; Elementos da Interface; Redefinição.

Abstract

Nowadays there are several interactive computer systems that still have inappropriate Graphical User Interfaces (GUIs) in terms of usability and experience for the user. Countless improvements were made in the development for new GUIs, however, not so much was done to improve the existing ones. This can be explained by the fact that mostly interactive computer systems, do not provide access to source code, restricting to the owners to the introduction of enhacements of them.

This dissertation presents the development of a tool capable of semi-automatically redefining (without access to the source code) graphic interfaces, through computational vision algorithms, making it more appealing and more usable. The tool is able to recognize a sub-set of interface elements that have certain characteristics.

The evaluation of the redefinition of the new GUI compared to the old one using the tool will be described. And the final results statistically demonstrate that the new GUI significantly reduces the number of errors committed, has a better success rate in completing tasks, and even better user satisfaction, which has been proven through a questionnaire about the tool.

Keywords: Graphical User Interface; OpenCV; Usability; Image Processing; Widgets; Redefinition

Índice

Índice de Tabelas

Índice de Figuras

Lista de Abreviaturas e Siglas

BOVW - Bag of Visual Words in a Nutshell

DSR – Design Science Research

GUI - Interfaces Gráficas do Utilizador

GUIRT - Graphical User Interface Redefinition Tool

IDE – Ambiente de Desenvolvimento Integrado

ISI - Interactive Systems Integration Tool

MSER - Maximally Stable Extremal Regions

SCI - Sistemas de Computação Interativa

SIFT - Scale-invariant Feature Transform

SVM - Support Vector Machine

TIC - Tecnologias da informação e comunicação

Capítulo 1 – Introdução

1.1. Enquadramento do tema

As interfaces gráficas do utilizador (GUIs) foram ao longo do tempo sujeitas a melhorias significativas. Todos esses progressos são o resultado de vários avanços, p.ex. melhores abordagens de desenvolvimento, isto é, centradas no utilizador, e novas técnicas de avaliação [1]. A área do *design* também fez avanços importantes, onde foram desenvolvidos princípios e padrões [2][3][4]. Através destes avanços e melhorias, é possível proporcionar ao utilizador maior usabilidade e uma experiência mais agradável. No entanto, apesar de todo o trabalho que tem sido desenvolvido ao longo dos anos, ainda existem sistemas de computação interativa (SCI), que têm GUIs inapropriadas. Este problema pode ser explicado por duas principais razões: i) os programadores não aplicam os mais recentes avanços nas suas interfaces; ii) os clientes não pretendem adquirir uma versão mais atualizada ou não foi desenvolvida uma nova versão.

Este trabalho foca-se nos casos em que não foi desenvolvida uma versão atualizada do *software*. E portanto, foi desenvolvida uma solução que possibilita a melhoria das GUIs existentes sem ter acesso ao seu código fonte. Atualmente muitas empresas ainda usam SCI antigos com GUIs inadequadas. Falamos de SCI que foram desenvolvidos sem ter em conta a usabilidade nem a preocupação com a experiência do utilizador na sua interação com a GUI.

Esta dissertação aborda este desafio apresentando a ferramenta GUIRT (*Graphical User Interface Redefinition Tool*), onde as GUIs são redefinidas de forma semi-automática sem acesso ao código fonte e com o objetivo de melhorar a sua usabilidade e experiência do utilizador. A aplicabilidade da ferramenta é ilustrada com um exemplo em que uma GUI antiga é redefinida e aprimorada.

1.2. Motivação

Apesar dos recentes avanços em termos de usabilidade a utilização de sistemas interativos computacionais continua a levantar desafios aos seus utilizadores [5][6]. Esses desafios são ainda mais exigentes quando são considerados vários tipos de utilizador. Isto é, utilizadores que tenham necessecidades especiais e onde a interface tenha que ser adaptada a essas necessidades. Como por exemplo, adaptar a interface com cores específicas, ou ter os *widgets* num certo tamanho.

A nova GUI redefinida pode ser adaptada a certas especificidades/limitações dos utilizadores (sendo benéfico para eles) sem que isso exija alterações nas aplicações originais. A solução desenvolvida nesta dissertação permite redefinir GUIs existentes sem necessidade de acesso ao seu código fonte, dando também a possibilidade de esta ser mais apelativa, facilitando a realização de tarefas durante o dia-a-dia de trabalho.

Por fim, uma das motivações principais no desenvolvimento deste projeto foi o fato de não existir (que seja do nosso conhecimento apesar da extensa pesquisa), nenhuma ferramenta com este tipo de comportamento.

1.3. Questão e objetivos de investigação

Nesta secção é apresentada a questão que foi levantada e quais os objetivos principais para a elaboração deste trabalho.

A principal questão que se levanta nesta investigação é:

- É possível desenvolver uma ferramenta que permite introduzir melhorias em termos de usabilidade em interfaces existentes sem ter acesso ao seu código fonte?

Para além da questão de investigação também é importante definir os objetivos para a elaboração deste projeto. Posto isto, as principais metas são:

- Desenvolver uma ferramenta para redefinição de interfaces gráficas via visão por computador (sem acesso ao código fonte);

- Aplicar a ferramenta num caso de estudo explorando as capacidades de redefinição da interface;

- Avaliar com utilizadores as interfaces gráficas produzidas para quantificar o impacto da ferramenta.

1.4. Abordagem metodológica

A metodologia adotada para o desenvolvimento deste trabalho tem o nome de Design Science Research (DSR) proposta por Peffers et al., (2007).

Esta metodologia é orientada para a criação de artefactos na área dos sistemas de informação e é composta por seis atividades: identificação do problema, definição dos objetivos para a solução, desenho e desenvolvimento, demonstração, avaliação e comunicação.

De acordo com a análise e escrita neste relatório as seis atividades são definidas da seguinte forma:

- Identificação do problema: encontram-se apresentados na secção da Introdução que engloba toda a introdução e enquadramento do tema, a motivação e o problema;
- Definição dos objetivos para a solução: os principais objetivos são numa primeira fase realizar toda a parte da redefinição das interfaces através de métodos de visão computacional, para conseguir identificar e extrair os *widgets* relevantes. Tudo isto com o intuito de se criar interfaces mais simples. De seguida, aplicar o segundo objetivo que é a criação das interfaces, com os *widgets* extraídos, fazendo com que estas fiquem mais apelativas e simples de usar. Numa fase subsequente realizar um conjunto de avaliações com utilizadores, aplicando num caso de estudo;
- Desenho e desenvolvimento: Utilização de ferramentas para o processamento de imagem, ou seja, através de métodos de visão computacional. Ter-se-á em consideração o conceito de usabilidade e de certas métricas para potenciar a interface gráfica;

- Demonstração: Realização de um conjunto de avaliações, aplicados a um caso de estudo, para serem executados por utilizadores;
- Avaliação: Verificar que de acordo com o resultado das avaliações e da realização de um questionário de satisfação, se realmente existe comentários positivos por parte dos utilizadores na utilização desta nova ferramenta;
- Comunicação: A importância deste trabalho será comunicada com este documento e uma publicação científica.

1.5. Estrutura e organização da dissertação

O presente documento está organizado em cinco capítulos.

O primeiro capítulo introduziu o tema da investigação, motivação e objetivos da mesma.

O segundo capítulo reflete o enquadramento teórico e sobre trabalhos relacionados.

O terceiro capítulo apresenta a arquitectura da ferramenta e a abordagem técnica utilizada na implementação da mesma, ilustrando as várias etapas seguidas e expondo as razões para as decisões tomadas.

O quarto capítulo apresenta a avaliação realizada e respetivos resultados obtidos.

No quinto e último capítulo apresentam-se as conclusões deste trabalho bem como as recomendações, limitações e trabalhos futuros.

Capítulo 2 – Revisão da Literatura

2.1. Background

2.1.1. Aspeto visual das GUIs

As pessoas preferem aquilo que é mais bonito, mais agradável de se ver. Para além disso, no que toca ao campo das tecnologias de informação, mais nomeadamente ao aspeto visual das interfaces gráficas, quanto mais atrativas forem, melhor. Portanto, é importante ter o cuidado de trabalhar nos detalhes dos aspetos visuais, fazendo com que o *design* se destaque de tudo o resto [7].

Hoje em dia, cada vez mais existe uma maior interação entre a pessoa e o computador, e portanto é necessário que a sua utilização seja satisfatória. Aqui entra o termo de usabilidade que está relacionado com a qualidade e satisfação em utilizar determinado *software*.

Uma das maneiras de garantir elevado nível de usabilidade, consiste na avaliação da usabilidade da interface, na qual pode ser realizada através de testes empíricos que são utilizados na participação dos utilizadores para a recolha de dados, onde estes serão analisados posteriormente. Por conseguinte, tudo isto também garante a qualidade do *software* através da sua interface gráfica, que corresponde à parte do sistema que o utilizador mantém contato ao utilizá-lo [8].

De acordo com o ciclo de vida de uma tecnologia apresentado por Norman [9], existe um ponto de transição onde a tecnologia satisfaz as necessidades básicas, mas os utilizadores hoje em dia, preferem comodidade e eficiência. Posto isto, todas as interfaces associadas a uma tecnologia devem ir sendo adaptadas ao longo do tempo de acordo com as expetativas do utilizador. Sendo que estas nunca são iguais e estáticas, logo é importante que haja ajustes ao longo do tempo.

Os desafios para alcançar a usabilidade universal estão associados à lacuna no conhecimento dos utilizadores e na diversidade tecnológica [9]. Segundo Habieb-Mammar, H. et al. [10], usabilidade universal consiste no “grau em que um serviço interativo é utilizável, ao mesmo tempo em que acomoda a possível diversidade de

utilizadores que acedem à aplicação a partir de uma variedade de contextos técnicos, organizacionais, sociais, físicos e culturais."

Existem várias abordagens desenvolvidas na construção das interfaces. Uma delas é criar soluções que sejam universais, enquanto que a outra abordagem engloba interfaces mais específicas e que vão de encontro a utilizadores particulares. Segundo Huh et al. [9] um dos maiores desafios é encontrar o equilíbrio entre os utilizadores e obter lucro para os *designers*. Tudo isto também está relacionado com o fato de os utilizadores extremos, não serem um número elevado e devido a isso serem um pouco ignorados. Ou seja, um *designer* que idealize uma interface que tenha apenas em conta utilizadores com necessidades especiais, acaba por não ir de encontro às exigências da maioria. Por outro lado, se implementar um *design* que tenha em conta situações específicas e não só, acaba por abranger um conjunto mais vasto de utilizadores.

De acordo com o que foi descrito e com o uso da nova ferramenta, a redefinição da nova interface terá em conta não só o utilizador comum, mas também utilizadores com condições especiais, tornando-se assim uma ferramenta com diversidade e abrangência. Para além disso tem como foco a atratividade e a usabilidade.

2.1.2. Processamento de Imagem

O processamento de imagem é um método que realiza algumas operações numa imagem, para obter uma imagem aprimorada ou para extrair informação útil da mesma. É um tipo de processamento de sinal no qual recebe como entrada uma imagem e como saída pode devolver também uma imagem ou então características associadas a essa imagem [11].

O processo para a implementação das novas GUIs redefinidas tem como base pegar nas imagens que representam as GUIs antigas e recolher os *widgets* dessas interfaces. Esses elementos das interfaces gráficas são extraídos através de métodos utilizados para o processamento de imagem.

Existe um vasto conjunto de ferramentas e bibliotecas para manipulação de imagens e vídeos, que se encontram disponíveis para desenvolver aplicações nesta área do

processamento de imagem. Uma delas é o OpenCV[1]. Desde a sua introdução em 1999, tem sido largamente adotada como principal ferramenta de desenvolvimento pela comunidade de investigadores e programadores no tópico de visão por computador [12].

Uma das técnicas utilizadas para detetar objetos numa imagem é por via de identificação de contornos. Contornos representam as formas encontradas numa imagem. Quando os contornos de um objeto são detetados, é possível, por exemplo, determinar o número de objetos numa imagem, classificar as formas e ainda o tamanho dos mesmos [12].

O processo de identificação do contorno tem como entrada uma imagem binária, onde vão ser aplicadas técnicas de binarização e/ ou de deteção de bordas [12]. Binarização consiste numa técnica de segmentação de imagem, geralmente usada para criar imagens binárias [13]. Segundo Shapiro, L. et al. [12], "binarização é uma técnica que converte uma imagem que possua uma escala de cinza para uma imagem binária, em que os pixéis são divididos em dois grupos, dependendo de um valor definido como o valor limite. Os pixéis que têm um valor maior do que o valor limite recebem o valor de 1, enquanto que os outros recebem o valor 0". Por fim, continuando o processo de identificação de contornos é necessário detetar uma sequência ordenada de pixéis que se encontram na fronteira, a partir dos quais é possível extrair a forma geral do padrão [12]. Na Figura 1 encontra-se ilustrado um exemplo de uma imagem onde foi aplicada a técnica de binzarização.

[1] https://opencv.org/

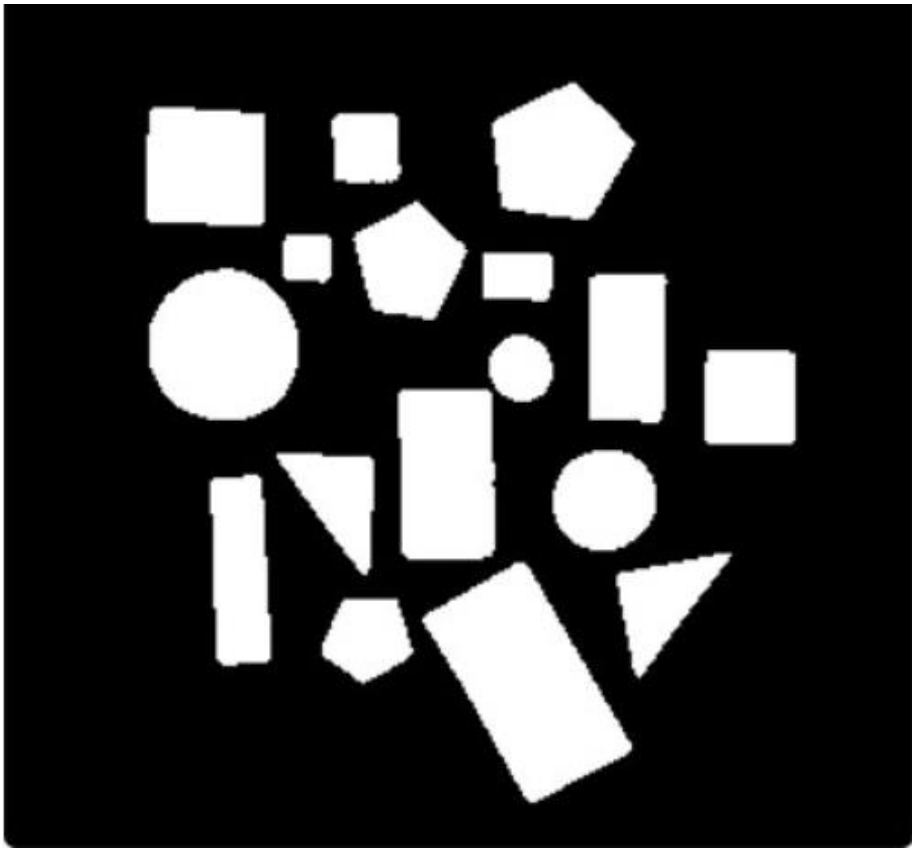

Figura 1 - Exemplo de uma imagem onde foi aplicada a técnica de thresholding [14]

Posto isto, foi feito um pequeno estudo comparativo entre possíveis ferramentas a utilizar para o processamento de imagem. Na Tabela 1, podemos observar um estudo comparativo com algumas ferramentas possíveis de utilizar.

Ferramentas	Métodos na deteção de regiões	Aspetos relevantes	Linguagens
OpenCV	MSER, findContours	- Possui módulos específicos para processamento de imagens em tempo real - Grande velocidade de processamento - Disponível para aplicações móveis - Grande comunidade *online*	C, C++, Python
CVIPTools	Não possui o método MSER nem o findContours	- Contém duas poderosas ferramentas de desenvolvimento que permitem o processamento em massa e a análise e desenvolvimento de algoritmos automáticos	C
VlFeat	MSER, não possui o findContours	- Escrito em C para eficiência e compatibilidade com interfaces em MATLAB para facilidade de uso - Melhor detetor em termos de repetibilidade e bom comprometimento da velocidade	C
Matlab Computer Vision Toolbox	MSER, não possui o findContours	- MATLAB é uma linguagem interpretada e afeta negativamente o desempenho. Este é muito importante na visão por computador, especialmente se se estiver a fazer processamento de vídeo em tempo real - Grande comunidade online	MATLAB

Tabela 1 - Estudo comparativo entre várias ferramentas

O OpenCV foi projetado para uma boa eficiência computacional e com um grande foco em aplicações em tempo real. Escrito em C / C ++ otimizado, a biblioteca pode aproveitar o processamento multi-core. Adotado em todo o mundo, o OpenCV "possui mais de 47 mil pessoas de comunidade de utilizadores e o número estimado de *downloads* excede os 18 milhões. A biblioteca é amplamente utilizada em empresas, grupos de investigação e órgãos governamentais" [15].

O CVIPtools é um *kit* de ferramentas de software para visão computacional e processamento de imagens, desenvolvido no Laboratório de Visão de Computador e Processamento de Imagem da Southern Illinois University em Edwardsville. Os instrumentos CVIPtools podem ler muitos formatos de imagem, incluindo TIFF, PNG, GIF, JPEG, BMP. Esta ferramenta suporta funções padrão de processamento de imagem, como compressão de imagem, restauração de imagem, operações lógicas e

aritméticas entre imagens, manipulação de contraste, nitidez de imagem, transformação de frequência, deteção de bordas, segmentação e transformações geométricas [16].

A biblioteca de código aberto VLFeat implementa algoritmos de visão de computador, especializados em compreensão de imagens e extração e correspondência de recursos locais. Algoritmos incluem Fisher Vector, VLAD, SIFT, MSER, k-means, k-means hierárquicos, superpixels SLIC, superpixels rápidos, treino via SVM em larga escala e muitos outros [17].

O Computer Vision System Toolbox fornece algoritmos, funções e aplicativos para projetar e simular sistemas de processamento de vídeo e visão de computador. É possível executar a deteção, extração e correspondência de recursos, bem como a deteção e rastreamento de objetos [18].

De acordo com a Tabela 1 e depois de uma análise, foi decidido que se ia utilizar a ferramenta OpenCV. Com esta ferramenta é possível fazer o reconhecimento de objetos em tempo real. Possui também uma vasta biblioteca de funções, entre elas, o *findContours* e MSER (*Maximally stable extremal regions*), para a deteção de contornos em imagens. Também é possível aplicar esta ferramenta em aplicações móveis, que poderia vir a ser um extra no desenvolvimento deste trabalho. E também o fato de possuir uma grande comunidade *online*, ajudou na decisão final.
Depois da escolha da ferramenta a utilizar, foram pensados dois tipos de abordagens (a utilização do método *findContours* e o uso de um classificado de aprendizagem automática), que irão ser mencionadas de seguida, no tema que remete para o processamento da imagem para a redefinição das novas GUIs.

2.1.3. FindContours

O OpenCV tem uma função específica para identificar contornos numa imagem: *cv2.findContours*. Tem como propósito extrair informação da imagem de origem, mas antes de ser aplicado, é necessário utilizar a técnica de binarização, através do método *cv2.threshold*, e só depois encontrar os diferentes contornos na imagem. Para uma melhor precisão, é necessário alterar a imagem para binária, lidando apenas com duas cores: preto e branco [19]. Utiliza a técnica de aproximação do contorno [12], que consiste em reduzir o número de pontos numa curva, através de um conjunto reduzido

de pontos. Esta técnica é conhecida como o algoritmo Ramer-Douglas-Peuckeral. A aproximação do algoritmo baseia-se no pressuposto de que uma curva pode ser aproximada por uma série de segmentos curtos de linha. Isso leva a um resultado de uma curva aproximada, que consiste num subconjunto de pontos que foram definidos pela curva original.

Este método tem como *input* três parâmetros: a imagem a ser processada em modo binário, o modo de recuperação do contorno e o método de aproximação do contorno. O output é: a imagem, os contornos e a hierarquia. Em termos de programação, os contornos são representados numa lista que contem todos os contornos da imagem [12].

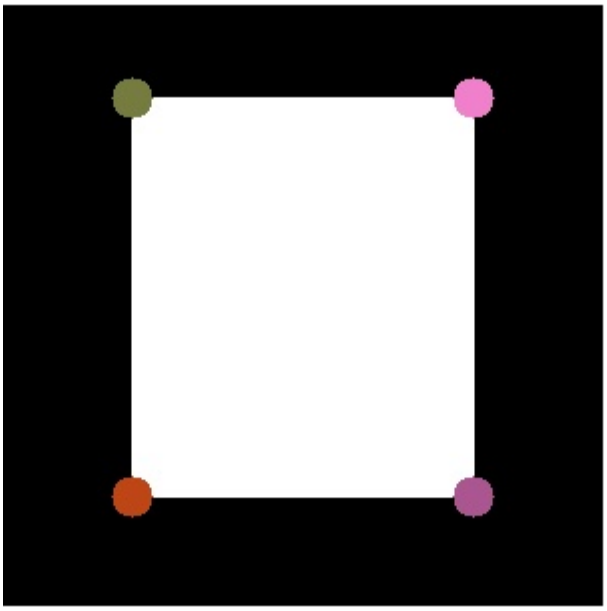

Figura 2 - Pontos de contorno para um retângulo [19]

De acordo com a Figura 2, os pontos são selecionados de tal forma que os contornos podem ser desenhados como uma linha reta unindo esses pontos. Mas apenas os pontos finais são armazenados, isto é, os que se encontram nos vértices do quadrado [19]. Para melhor visualização do objeto que foi encontrado numa imagem, como se pode observar pela Figura 3, existe a função *cv2.drawContours*, onde é possível desenhar a forma do contorno, desde que estejam definidos os pontos de contorne do objeto em questão. A cor em questão é verde mas a escolha da mesma fica ao encargo do utilizador que esteja a usar esta técnica.

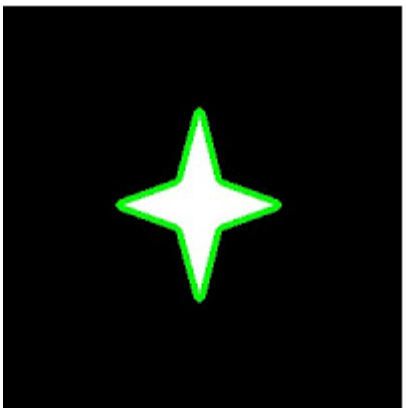

Figura 3 - Contornos desenhados [19]

2.1.4. Computação Orientada por Imagens

Quando se fala no domínio das tecnologias de comunicação da informação (TIC), tem-se logo a perceção que atinge um impacto inegável nos dias de hoje. E quando se pensa em TIC's mais acessíveis, sabe-se que estas podem vir a beneficiar um grupo alargado de pessoas com problemas de saúde ou que tenham algum tipo de incapacidade.

Existe um novo paradigma de computação, computação orientada por imagens, que recentemente foi introduzido no contexto geral de programação e no teste da interface do utilizador [20].

Segundo George Kourousias e Silvio Bonfiglio [20], computação orientada por imagens consiste num paradigma de computação que tem como núcleo a abstração de dados da informação visual num domínio espacial bidimensional. Esta informação visual é o que pretende ser visível num dispositivo gráfico de saída. Normalmente o dispositivo gráfico de saída é o monitor do computador e a informação visual é uma imagem num todo ou partes da mesma. Ao contrário do que foi descrito, os computadores tendem a trocar blocos de informações com base nas abstrações de dados de informações numéricas e textuais. Por exemplo, quando se faz uma pesquisa na internet, num motor de pesquisa, sobre cadeira de rodas, requer que escrevamos a informação via texto ou palavras-chave sobre o artigo (p.ex.: cadeira de rodas). Como alternativa, pesquisas recentes resultaram em serviços orientados por imagens que permitem que a pesquisa seja realizada com uma fotografia como entrada. Podemos observar esse mesmo exemplo na Figura 4 onde se visualiza dois motores de pesquisa, com o intuito de em

vez de se escrever o texto da pesquisa que se pretende efetuar, coloca-se uma imagem sobre essa mesma pesquisa.

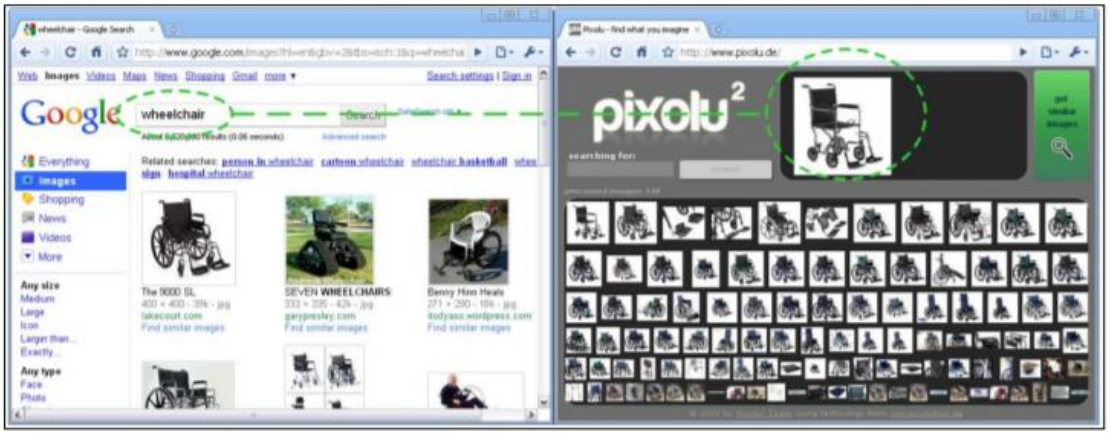

Figura 4 - Serviço de pesquisa de imagens que utiliza palavras-chave e outro que utiliza imagens representativas em vez de palavras-chave [20]

O Sikuli [20] é um exemplo representativo da computação orientada por imagens, tanto para o utilizador final quanto para o programador. O Sikuli Script é um módulo de *script* que permite que os programadores usem *screenshots* de elementos da interface para controlá-los programaticamente sem qualquer API. Ou seja, através de screenshots sobre elementos de uma interface e a utilização de eventos sobre o teclado ou o rato é possível automatizar uma determinada ação numa GUI. O script consegue reconhecer os elementos da interface através de técnicas de processamento de imagem. O Sikuli Script incorpora o *Python* como linguagem de *script* e um ambiente de edição projetado adequadamente para escrever *scripts* de automação baseados em *screenshots*. Do ponto de vista técnico, é uma biblioteca Jython e Java que automatiza a interação da GUI usando padrões de imagem para direcionar eventos de teclado e rato [20]. Na Figura 5 é possível observar um exemplo de um *script* no Sikuli.

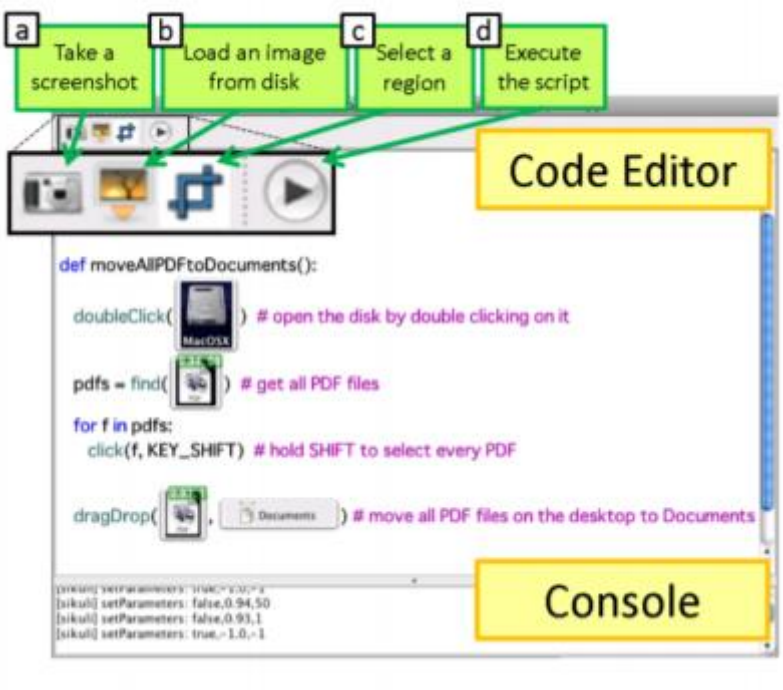

Figura 5 - Editor para escrever scripts sikuli em python [21]

Para além do que já foi descrito, existe também uma abordagem para testes de interfaces que utiliza a técnica de visão por computador, através do Sikuli *Test*. Com esta ferramenta é possível ajudar os utilizadores que realizam a avaliação das interfaces a automatizar as suas tarefas. Permite também que estes escrevam *scripts* visuais usando imagens, isto é, *screenshots* sobre os widgets da interface que devem ser testados e qual o *feedback* visual que deve ser observado [22].

O Sikuli foi a ferramenta escolhida para a automatização dos passos a executar na GUI da aplicação original enquanto se utiliza a nova GUI redefinida, pois é *opensource*, possui uma vasta comunidade *online* e para além disso foi sugerido pelo orientador.

2.2. Redefinição de Interfaces

2.2.1. Prefab

No decorrer do estudo e pesquisa de ferramentas que ajudassem no processamento de imagem foi encontrado um *toolkit* com o nome de Prefab. O Prefab possui métodos

baseados em pixéis que permitem a modificação de interfaces existentes sem acesso ao seu código [23]. Estes métodos retém a informação do espectro contida numa imagem onde cada píxel é processado. É utilizado para classificação de objetos numa imagem [38]. Nesta *toolkit* as interfaces são hierarquias onde se aplica engenharia inversa trabalhando iterativamente a partir de pixéis em bruto para uma interpretação detalhada [23].

A Figura 6 ilustra um mecanismo básico que funciona da seguinte maneira:

- Um bitmap de janela de origem é capturado;
- A imagem de origem é interpretada;
- A interface modificada é apresentada numa nova janela (janela de destino);
- A entrada na janela de destino é mapeada de volta para a de origem;
- Por fim gera uma nova saída que é capturada e usada para atualizar a janela de destino.

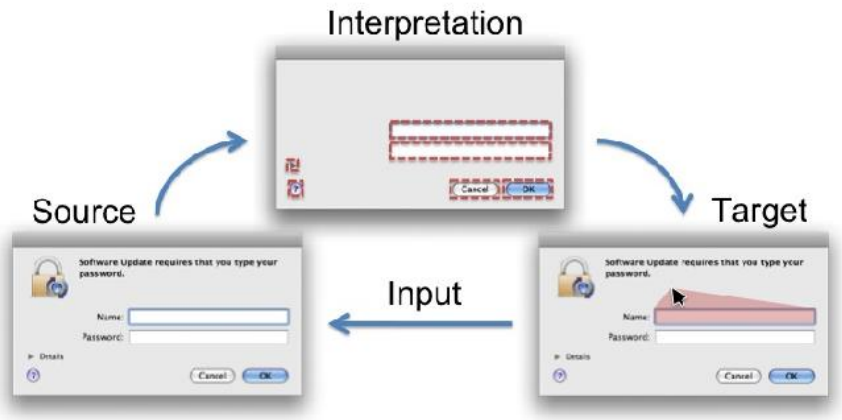

Figura 6 - Mecanismo que o Prefab utiliza para gerar novas interfaces [24]

O Prefab é o primeiro sistema que consegue combinar métodos baseados em píxeis com um redireccionamento de entrada e saída, permitindo que este modifique uma interface independentemente da forma como foi implementada, isto é, não tendo acesso ao código fonte [24].

A Figura 7 ilustra um exemplo de um protótipo, que consiste num modelo de oito partes que parametriza um modelo abstrato de uma borda de oito partes com partes

correspondentes a um botão Microsoft Windows 7 *Steel*. Este reconhece todos os botões do Microsoft Windows 7 *Steel*, independentemente de seu conteúdo interno. Protótipos que atribuem diferentes pixéis às suas partes podem reconhecer diferentes estilos de botões ou diferentes *widgets* que pintam uma borda. O Prefab é implementado como uma biblioteca de protótipos com métodos para aplicar esses protótipos para identificar *widgets* [24].

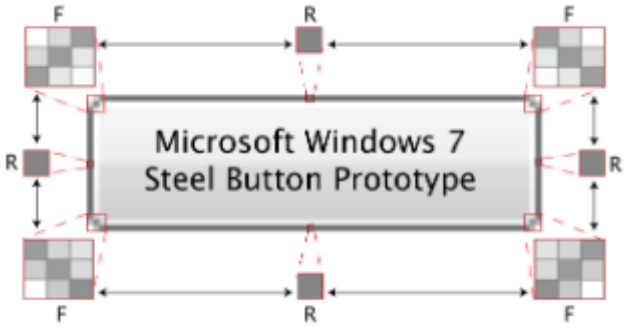

Figura 7 – Exemplo de um botão da Microsoft onde é aplicado o protótipo do Prefab [24]

Depois de toda uma pesquisa e investigação sobre esta ferramenta, chegou-se à conclusão de que poderia ser uma mais-valia na realização deste trabalho e portanto descobriu-se que o código encontrava-se disponível no *github*. Foi realizado o *download* da ferramenta para se perceber se funcionava ou não. Depois de testada verificou-se que a mesma não estava a funcionar, deixando então de se tornar numa opção viável.

2.2.2. Classificação de objetos de uma GUI

No trabalho de Dubrovina, A et al. [25], foi elaborado um estudo no qual foi feito o reconhecimento e classificação de elementos de uma GUI a partir do método de classificação: SVM (*Support Vector Machine*). Este tópico é importante para o ramo da automação de *software*, no qual é baseado em imagens, tendo como objetivo imitar a interação entre o humano e o computador.

Foram utilizados diferentes tipos de objetos provenientes de um conjunto de dados tratado. Observando a Figura 8, todos os *screenshots* foram tirados de aplicações

Windows. Para além disso os objetos possuem uma variedade de tamanhos, cores, textos ou imagens dentro dos objetos.

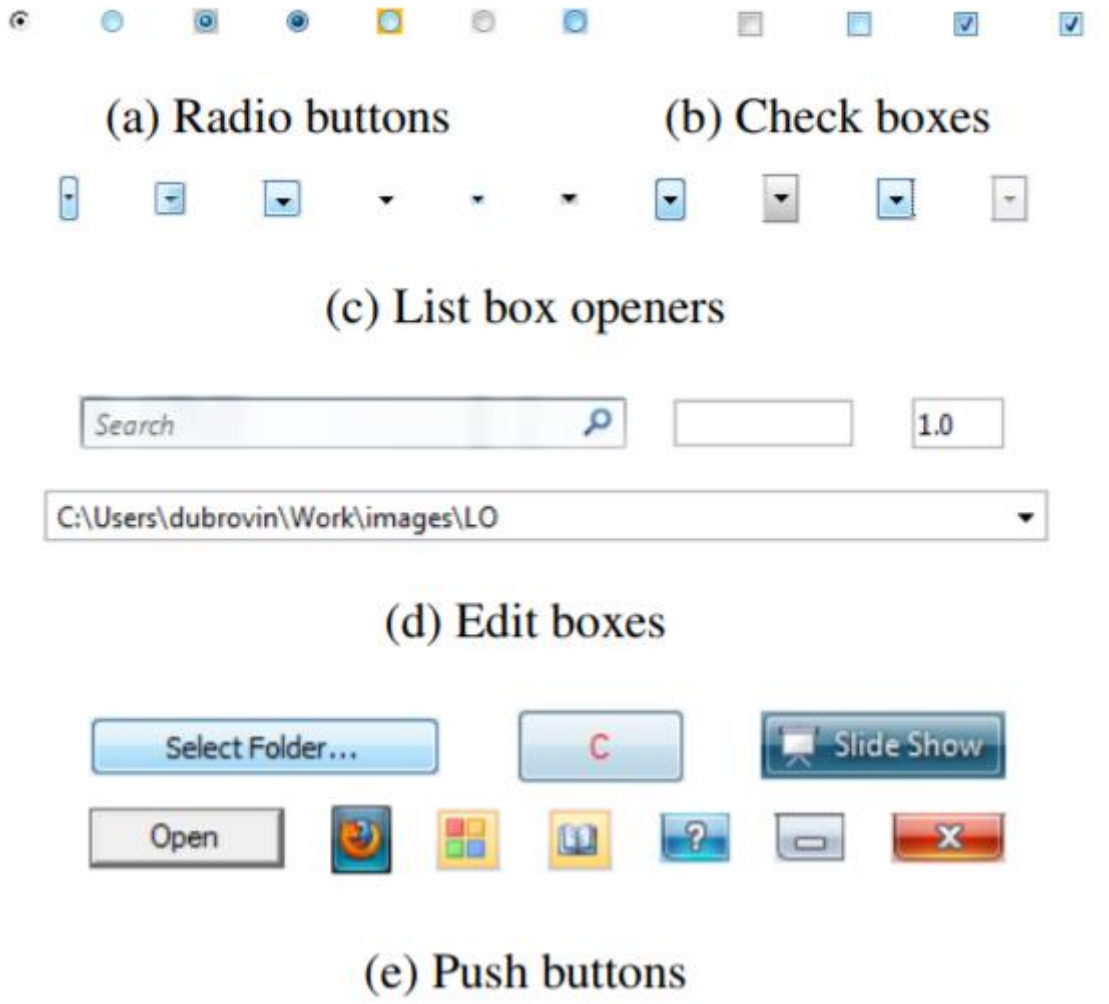

Figura 8 – Exemplos de diferentes objetos de várias interfaces relativos ao dataset usado [25]

Foi proposto um novo tipo de descritor de imagem desenvolvido especificamente para realizar a classificação de um objeto GUI. O descritor é baseado na versão 1D da Transformada de Fourier-Mellin, isto é, dada uma imagem em tons de cinza, são calculadas as derivadas das suas projeções nos eixos *x* e *y*. Para além disso é incorporada a informação sobre os gradientes da imagem do objeto e a percentagem da cor branca do mesmo [25].

Para a realização da avaliação do descritor foi criado um conjunto de dados de 258 objetos segmentados pertencentes a cinco classes de objeto GUI: *check box*, *edit box*, *list box opener*, *push button* e *radio button*. Foi utilizado este conjunto de dados tanto

para treinar o classificador do Kernel SVM, e também para executar o procedimento de validação cruzada, para encontrar os parâmetros SVM ideais.

Para avaliar a qualidade da classificação com o descritor proposto, foi dividido aleatoriamente o conjunto de dados de treino e teste. Foram utilizados conjuntos de treino de diferentes tamanhos - 40% - 90% do tamanho total do conjunto de dados. Foram apresentadas 10 experiências para cada um desses tamanhos de treino. Depois foi calculado o número médio de objetos classificados corretamente de todas as cinco classes, normalizados pelo tamanho do conjunto de teste, em função do tamanho do conjunto de treino. Na Figura 9 abaixo encontram-se os resultados [25].

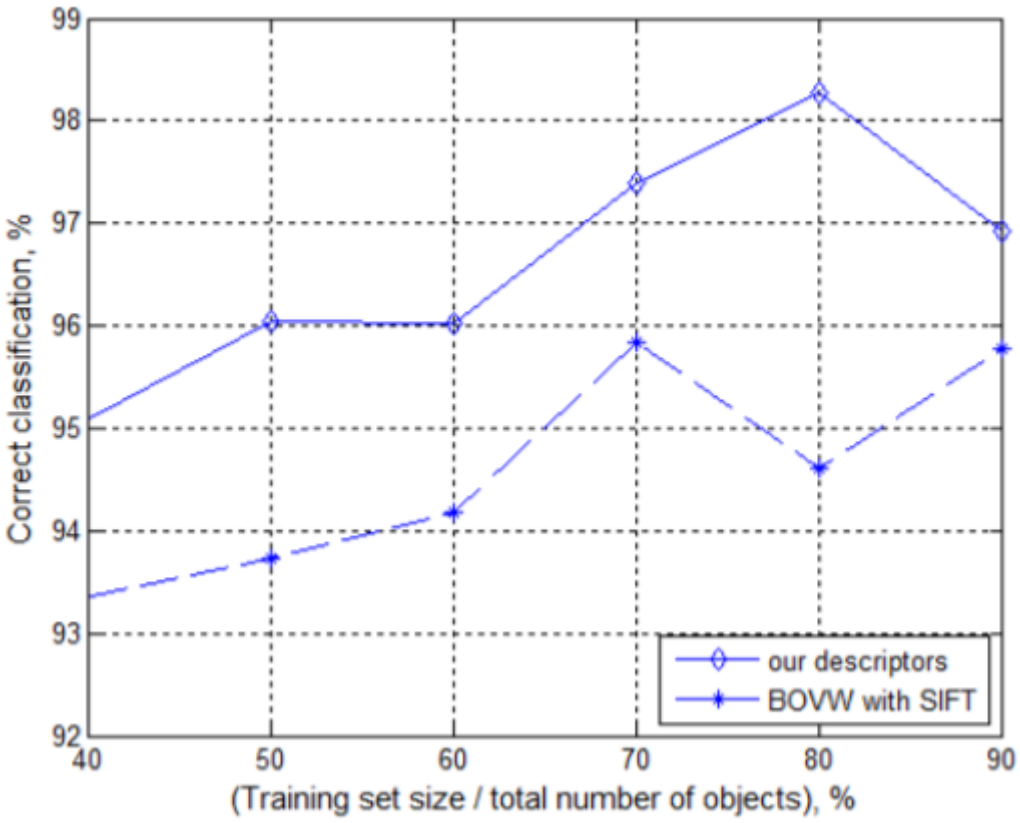

Figura 9 - Grafico com os resultados [25]

A Figura 9 ilustra um gráfico onde foi feita uma comparação de resultados entre os descritores SIFT (*Scale-invariant Feature Transform*) - *Bag Of Visual Words (BOVW)* e os descritores mencionados anteriormente [25]. E como se pode observar, embora com a utilização do método SIFT, os resultados tenham sido bons, o descritor proposto teve ainda resultados melhores na classificação dos objetos de GUIs.

Depois de uma análise a este estudo e como existia a hipótese de se adotar esta abordagem para o desenvolvimento deste trabalho, foi então contactada uma das autoras deste trabalho para saber se era possível ter acesso ao *dataset* utilizado. A mesma informou que já não possuía consigo o conjunto de dados, não sendo possível obtê-lo. Posto isto e depois de muita pesquisa, sem sucesso, de possíveis *datasets* com este tipo de dados, esta abordagem acabou por ficar sem efeito.

2.2.3. Supple

Existe também um outro trabalho desenvolvido, designado de Supple que é capaz de gerar automaticamente novas interfaces. Quando é necessário processar uma interface de um certo dispositivo, este sistema tem como objetivo procurar a execução que vai de encontro às restrições de um certo dispositivo e ao mesmo tempo, minimizando o esforço estimado numa ação na interface por parte do utilizador. O Supple calcula não só o *layout* ideal, mas também escolhe os *widgets* a serem usados na renderização da nova interface. Um renderizador de uma interface adaptável requer três entradas: uma especificação funcional da interface, um modelo do dispositivo e por fim um modelo do utilizador, ou seja, os passos que este faz para realizar uma tarefa [26].

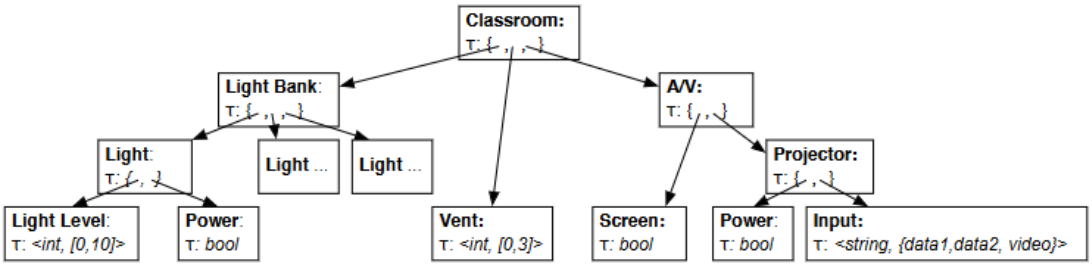

Figura 10 - Representação em árvore da especificação da interface funcional de um controlador de aparelho de sala de aula [26]

A Figura 10 ilustra a especificação funcional de uma interface para um controlador de aparelho de uma sala de aula. A interface é definida por um conjunto de elementos e restrições que são especificadas pelo *designer*. Foram refletidos certos aspetos dos recursos e quais os requisitos da interação com um dispositivo, fornecendo à Supple bibliotecas de *widgets* apropriadas. Por exemplo, se o modo principal de interação com a interface renderizada for tocar, apenas *widgets* suficientemente grandes serão usados e

serão manipulados com o dedo. Por fim, é necessário adaptar as interfaces consoante o tipo de utilizador. Foi optado usar um rastreamento como fonte de informação sobre a forma como a interface é usada pelo utilizador. Esse rastreamento é definido por um conjunto de caminhos, ou seja, as sequências de passos onde os elementos foram manipulados pelo mesmo [26].

Relativamente ao algoritmo implementado, este tem como objetivo encontrar a solução ótima. Este algoritmo consiste na propagação de restrição completa em cada etapa de pesquisa. A propagação de restrição garante que depois de cada variável ser atribuída, os potenciais *widgets* considerados para variáveis não assignadas são consistentes com todas as restrições. Isso permite detetar mais rapidamente os caminhos que não produzirão uma solução ótima. Por exemplo, supõe-se que um widget do tipo *slider* (grande) que tenha sido escolhido para a velocidade de ventilação. A propagação de restrição pode perceber imediatamente que não havia como ajustar um *widget* à intensidade de luz. Além disso, permite fazer estimativas mais precisas do custo final da interface completa [26].

De seguida é possível observar pela Figura 11 e Figura 12, dois exemplos onde foram aplicados o Supple. Na Figura 11 encontram-se ilustrados dois exemplos de uma interface da sala de aula, onde na imagem do lado esquerdo está presente um dispositivo baseado em ponteiros e no lado direito baseado num painel de toque. Na Figura 12 é possível observar os vários *screenshots* de uma interface da sala de aula com os passos para manipular o nível de brilho das luzes.

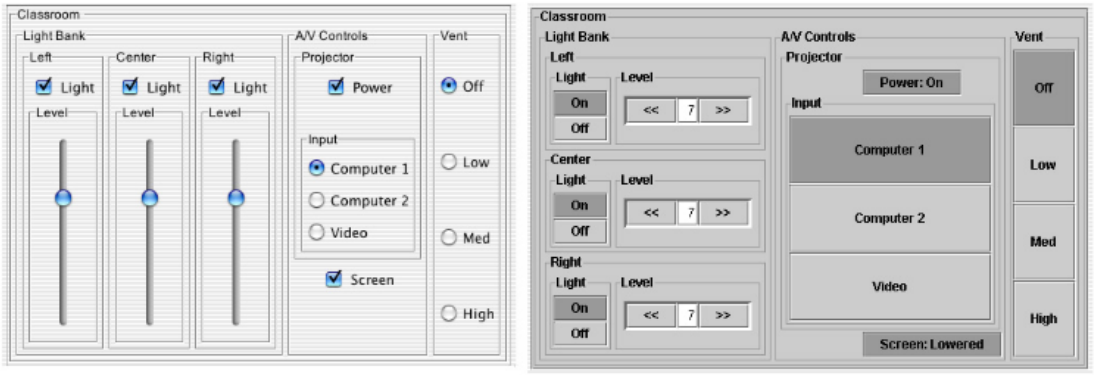

Figura 11 - A interface da sala de aula processada para dois dispositivos com o mesmo tamanho: (a) um dispositivo baseado em ponteiro (b) um dispositivo de painel de toque [26]

Figura 12 - A interface da sala de aula processada por um simulador de telefone WAP (Sony Ericsson T68i) [26]

2.2.4. Proxies de Interação

Existe atualmente, também uma estratégia no que toca a alterações de interfaces em tempo real. Estamos a falar da utilização da ferramenta proxies de interação, capaz de em tempo real adicionar melhorias a interfaces para aplicações móveis, tendo em conta a acessibilidade. Como se pode observar pela Figura 13, esta funcionalidade está inserida entre a interface original de uma aplicação e a interface de manifesto. Isto é, a interface que o utilizador usa para interagir com a aplicação (exemplo: interface quando tem o *talkback* do android, para permitir ler o que está na mesma). Todo este processo é implementado sem se ter acesso ao código fonte [27].

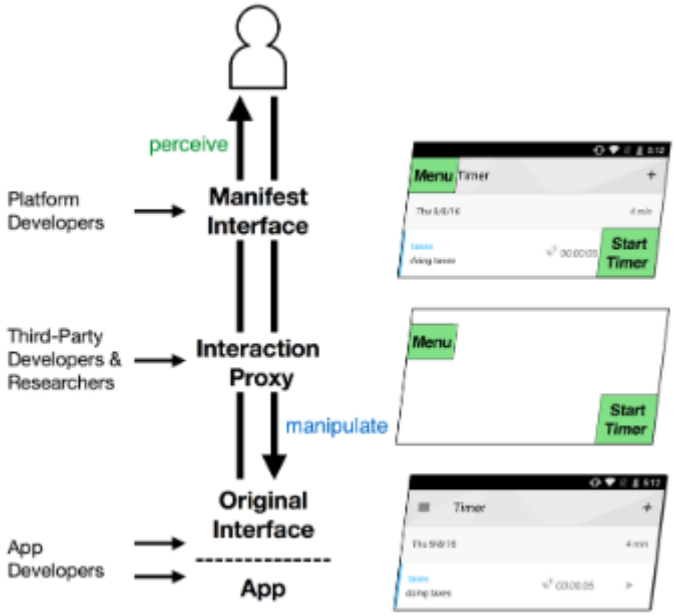

Figura 13 - Ilustração da inserção de um proxie de interação para melhorar a acessibilidade [27]

A técnica utilizada na modificação da interface é através da adição de *floating* Windows, de acordo com as informações disponíveis através de APIs de acessibilidade das plataformas padrão [28]. *Floating* Windows corresponde à camada que fica sobreposta sobre a interface original, ficando visível no ecrã. Na Figura 14 é possível visualizar um exemplo de implementação usando o conceito de *floating* window para criar uma sobreposição completa, sendo exibida como uma sobreposição translúcida, capturando todos os elementos. Utiliza a introspeção do conteúdo, isto é, a informação sobre cada elemento da interface (p.ex. conteúdo, tamanho, estado, etc), para encontrar as fronteiras dos elementos que devem estar visíveis, depois através de automatização, para a fase de manipulação dos elementos. Por fim utiliza um evento para as partes auditivas que permite com que haja uma resposta rápida do *proxy*, numa mudança da interface, como por exemplo, quando a interface avança entre cada etapa [27].

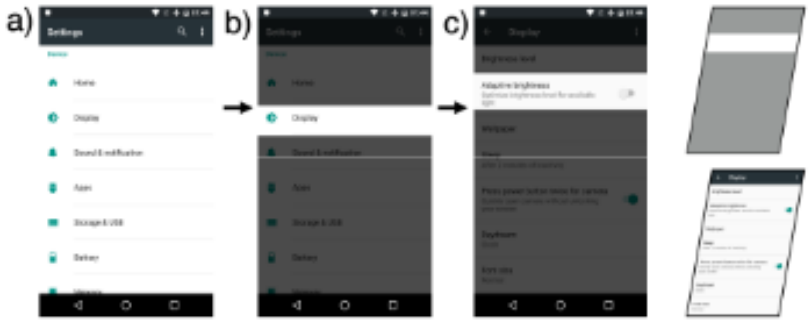

Figura 14 - Exemplo de um tutorial de uma aplicação através da ferramenta proxie de interação [27]

2.2.5. Comparação dos trabalhos

Para uma melhor compreensão sobre como se posiciona o presente trabalho, perante todos os trabalhos relacionados mencionados acima, é possível observar através da Tabela 2 uma visão geral em termos comparativos como é que as várias ferramentas se comportam.

Ferramentas	Redefinição da GUI	Máquina Virtual	Avaliação Utilizadores
Prefab	Sim	Não	Sim
Classificação de objetos com Redes Neuronais *	Sim	Não	Não
Supple	Sim	Não	Sim
Proxies de Interação	Sim	Não	Sim
ISI	Não	Sim	Sim

Tabela 2 - Tabela comparativa entre os vários trabalhos (Não se enquadra como uma ferramenta mas sim como um caso de estudo comparativo entre a utilização do classificador SVM e um método usado por visão por computador (SIFT))*

No que diz respeito ao Prefab, esta ferramenta consegue redefinir os *widgets* numa GUI através da funcionalidade engenharia inversa baseada em pixéis. Fazendo a identificação e interpretação dos vários widgets da GUI, esta consegue implementar soluções para melhorar a interface (p.ex. técnicas de *target-aware pointing*, ou seja, perceber que o *widget* foi usado/modificado através de uma sinalização visual). Sobre esta ferramenta a parte da identificação dos *widgets* poderia ser algo interessante de enquandrar no GUIRT, sendo que não foi possível, pelas razões mencionadas anteriormente.

Sobre a classificação de objetos através de um método de aprendizagem automática entende-se mais como uma forma viável para implementar a fase inicial da ferramenta, isto é, a parte da identificação dos *widgets*. Apesar de possuir bons resultados seria necessário despender bastante tempo em arranjar um *dataset* que fosse o mais abrangente possível em temos de *widgets* existentes numa interface. Para além de que todos os *screenshots* do *dataset*, para este estudo, foram tirados apenas para aplicações Windows. Perante este fato, é evidente uma limitação de *widgets* no *dataset* não podendo abranger outro tipo de aplicações.

Em relação ao Supple foi evidenciando anteriormente que é necessário existir um conhecimento *à priori*, ou seja, providenciar um primeiro *input*, de como será modelada a nova interface, antes de se começar a construi-la. Por exemplo, se a interface possuir várias *checkboxes*, estas são classificadas como booleanas, ou se existirem *sliders* é preciso saber o intervalo de valores, entre outros exemplos. Este tipo de passos antes de se chegar à finalização da interface, faz com que o processo seja bastante dispendioso, que requer uma grande preparação inicial.

Relativamente à ferramenta proxies de interação, possui uma limitação evidente pois apenas está posicionada para modificação de interfaces em aplicações móveis, não abrangendo aplicações de *desktop* ou para outros dispositivos. De acordo com o que foi abordado, e de acordo com os objetivos traçados na realização deste trabalho, foi evidente que não foi encontrada uma solução que se assemelhasse de algum modo ao que iria ser implementado. Apenas existem trabalhos que aplicam métodos diferentes para a construção da nova interface, tendo claro as suas limitações, como por exemplo, apenas abrangerem certos dipositivos.

Atualmente existe uma ferramenta que dá suporte a programadores nas áreas de engenharia de sistemas interativos, o ISI (*Interactive Systems Integration Tool*). Tem como objetivo integrar diferentes funcionalidades de aplicações, independentemente da linguagem de programação ou de sistema operativo, numa nova GUI, tendo em consideração aspetos de usabilidade. Visa em facilitar a integração entre vários sistemas, oferecendo interações simplificadas para o utilizador final. Isto é, reduz os desafios que os utilizadores enfrentam, quando tentam executar uma tarefa, usando um ou mais sistemas interativos. Para além disso também melhora a eficiência e o desempenho dessas tarefas [29]. Apesar de também possuir o recurso à máquina virtual para executar as tarefas na aplicação original que se encontra escondida, esta ferramenta não é capaz de redefinir as interfaces.

Para concluir e fazendo uma comparação entre a GUIRT e todas as ferramentas apresentadas, esta tem a capacidade de conseguir abranger interfaces de qualquer sistema (desde que não possua *widgets* muito complexos), podendo ainda ser adaptada para a área das aplicações móveis, pois a ferramenta OpenCV fornece essa possibilidade. Para além disso, o GUIRT tem a capacidade de redefinir interfaces para qualquer tipo de utilizador, tendo apenas a limitação de ter que consumir em tempo real a interface original (através de uma máquina virtual) fazendo com que esta parte do

processo seja lento. Sendo que em todos os outros exemplos não existe essa restrição. De referir também que com o GUIRT não é necessário ter um conhecimento de como é a interface antes de a redefinir, fazendo com que seja um processo muito mais genérico, ao contrário da ferramenta Supple. E com o recurso a uma máquina virtual é possível esconder a aplicação de origem e visualizar apenas a nova interface. E como se pode observar pela Tabela 2 nenhuma das ferramentas possui esta funcionalidade, exceto o ISI.

2.3. Conclusões

Tendo em conta toda a informação recolhida e todos os obstáculos existentes para o desenvolvimento deste trabalho, optou-se por seguir o caminho da visão computacional, usando a ferramenta OpenCV, deixando para trás a utilização da ferramenta Prefab, pois não foi possível colocar a funcionar a solução disponível no *github*. A utilização de um método de aprendizagem automática para classificar objetos numa imagem, também foi uma opção excluída, pois não foi fornecido pelo autor, o acesso ao conjunto de dados. Para além disso, ainda se pesquisou por possíveis conjunto de dados que tivessem um conjunto de *widgets* de uma interface, mas sem qualquer sucesso.

Todos os trabalhos relacionados mencionados anteriormente possuem o seu propósito, como por exemplo melhorar as interfaces para utilizadores específicos, ou simplesmente aplicar técnicas/soluções para melhorar a sua interface. Porém, e apesar de todos eles implementarem métodos específicos para o reconhecimento e identificação dos *widgets*, no qual este processo se assemelha ao comportamento do GUIRT, estas ferramentas não tem o foco de aplicar as suas interfaces melhoradas para qualquer utilizador nem para qualquer situação. Sendo que o GUIRT tem como preocupação redefinir uma interface quer para um utilizador “normal”, quer para um utilizador específico, chegando assim a um público-alvo maior.

Capítulo 3 – Abordagem

Neste capítulo é abordado todo o processo de criação da ferramenta, desde a sua arquitectura, passando por todas as fases de extração e segmentação dos *widgets* da GUI original, para dar origem à nova GUI redefinida. Também serão explicadas ambas as abordagens que foram definidas para o desenvolvimento do GUIRT: o modo de edição onde o interveniente principal é o programador e o modo de execução onde o utilizador final tem acesso. O objetivo deste capítulo é mostrar com clareza e de forma perceptível a forma como este ciclo de interações entre as várias GUIs se sucede.

3.1. Arquitectura

A GUIRT tem como objetivo principal criar interfaces gráficas, a partir de uma aplicação já existente, com maior usabilidade e potencialmente com um aspeto gráfico mais apelativo, para que, seja mais fácil e agradável a sua utilização. De salientar que a visualização final da interface gráfica tem também como opção ser orientada de acordo com as necessidades de um determinado tipo de utilizador. Para estes casos, está-se a falar dos utilizadores com necessidades especiais, e que necessitam de interfaces adaptadas. Através desta ferramenta foram identificadas soluções que os podem ajudar.

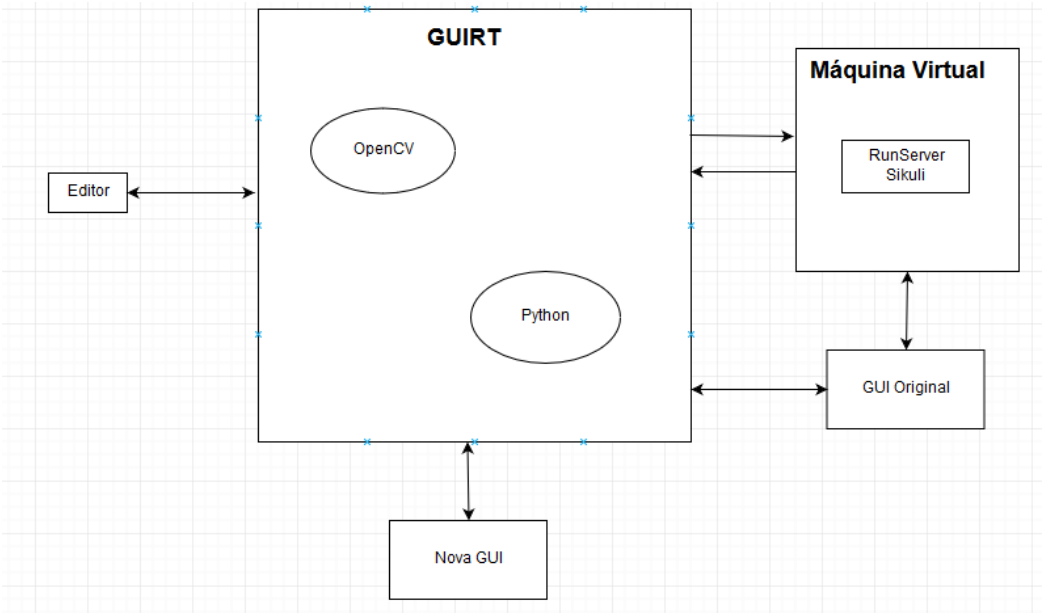

Figura 15 - Diagrama geral GUIRT

A Figura 15 apresenta a forma como toda a arquitectura foi desenhada. Para começar do lado esquerdo encontra-se a ferramenta GUIRT desenvolvida em *Python* na qual foi utilizada a biblioteca OpenCV responsável pela parte da visão computacional. No incício do processo, a GUI original encontra-se no computador local e atráves da aplicação de métodos específicos, são identificados e extraídos os *widgets* existentes na interface original (atualmente restrito a um subconjunto de *widgets*) e colocados na nova interface. Esta nova GUI pode conter algumas imperfeições aquando a sua criação, mas é possível através do GUIRT abrir um editor externo de *python*, onde se pode fazer ajustes à nova interface. Todo este processo de reconhecimento de interfaces é feito de forma cíclica, ou seja, até estarem todas as interfaces do sistema interativo antigo processadas.

Passando à segunda parte do processo, sendo que estas duas fases estão divididas em modo de edição e de execução (irá ser abordado em mais detalhe nas secções seguintes), abrimos a nova interface novamente através do GUIRT e quando é feita uma ação, essa mesma ação é replicada para a interface antiga. Este processo é feito através do uso da ferramenta Sikuli, onde são criados *scripts* que automatizam uma determinada ação sobre um evento que tenha acontecido na nova interface. Ou seja, quando é efetuado um evento sobre a nova interface, esta ação terá que realizar todos os mesmos passos na interface original. Nesta fase a interface original encontra-se dentro de uma máquna virtual, para o utilizador não se aperceber, que à medida que está a mexer na nova interface, ocorre ao mesmo tempo a execução de uma script sikuli, onde é feita a replicação do mesmo passo mas na máqina virtual. Esta conexão é feita através do servidor *RunServer*, que permite correr *scripts* entre diferentes máquinas.

Como já foi referido anteriormente a ferramenta tem dois modos de operação: o modo de edição e o modo de execução. Inicialmente a aplicação não foi pensada desta forma, ou seja, a primeira ideia era desenvolver uma solução que redefinisse a interface de forma automática, onde o utilizador em tempo real iria ter uma GUI nova a partir da GUI original, não existindo a necessidade de operar em dois modos. No entanto, não foi possível prosseguir com esta abordagem porque os algoritmos usados para o processamento de imagem eram limitativos no reconhecimento de todos os *widgets* possíveis. E também devido ao fato do método usado para o reconhecimento de texto não ser 100% eficaz, fazendo com que depois na parte da tradução (funcionalidade do GUIRT) tivesse impacto. Perante este cenário, foi necessário recorrer à

alternativa/solução de operar com os dois modos mencionados anteriormente. Em que no modo de edição o programador tem a função de melhorar a interface em todas as suas possíveis falhas e ainda poder melhorar aspetos na parte gráfica. Em seguida, mais em detalhe, irão ser descritas estas duas opções.

3.2. Modo de Edição

Inicialmente esta ferramenta é iniciada no modo de edição pelo utilizador inicial – o programador - para iniciar o processamento da interface original, escolhendo a aplicação que pretende. Para além disso e observando a Figura 16, o programador tem a opção de redefinir a interface para a língua que quiser e para um determinado tipo de utilizador. Por defeito aparece sempre em português para um utilizador "normal". Estas funcionalidades irão ser abordadas mais em detalhe na secção: funcionalidades extras.

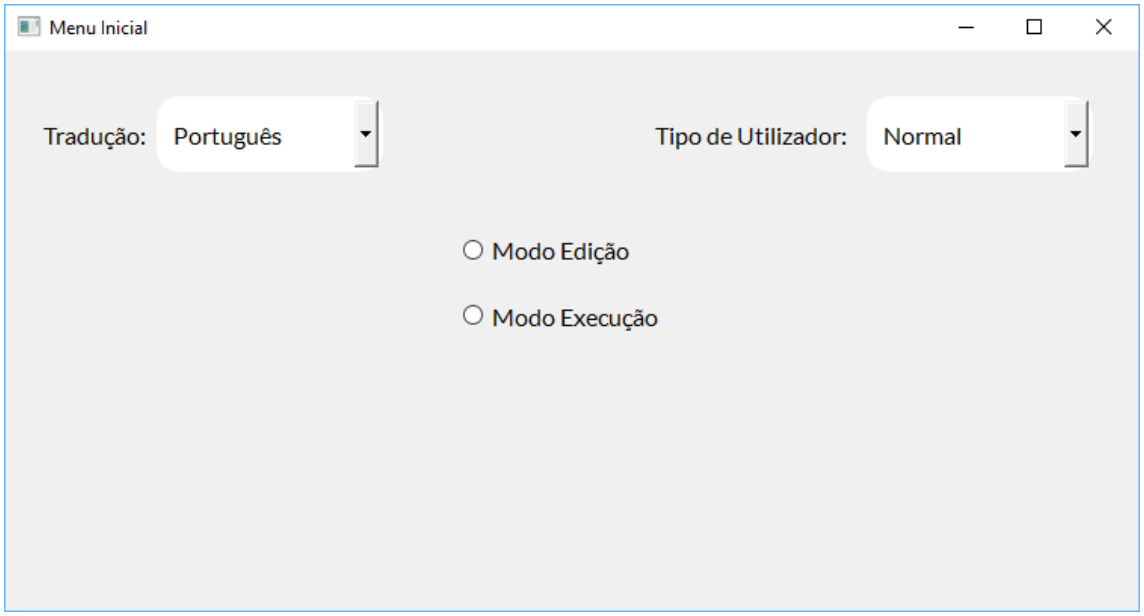

Figura 16 - Menu inicial do GUIRT

De seguida a ferramenta tem a opção "Redefinir" onde são executados os métodos responsáveis por extrair os *widgets*, para a construção da nova interface. Neste processo é utilizada a ferramenta OpenCV, para se efetuar a extração e manipulação dos *widgets*.

De seguida, é feito o *merge* desses mesmos *widgets*, elaborando assim a nova interface pretendida. Durante todo este modo de pré-processamento é mostrada uma barra de espera dando *feedback* ao programador de quando este se encontra finalizado. Na Figura 17 encontra-se a opção de edição no GUIRT onde existe o botão para redefinir a interface e o botão com a opção de abrir num editor a nova interface, sujeita a possíveis melhorias.

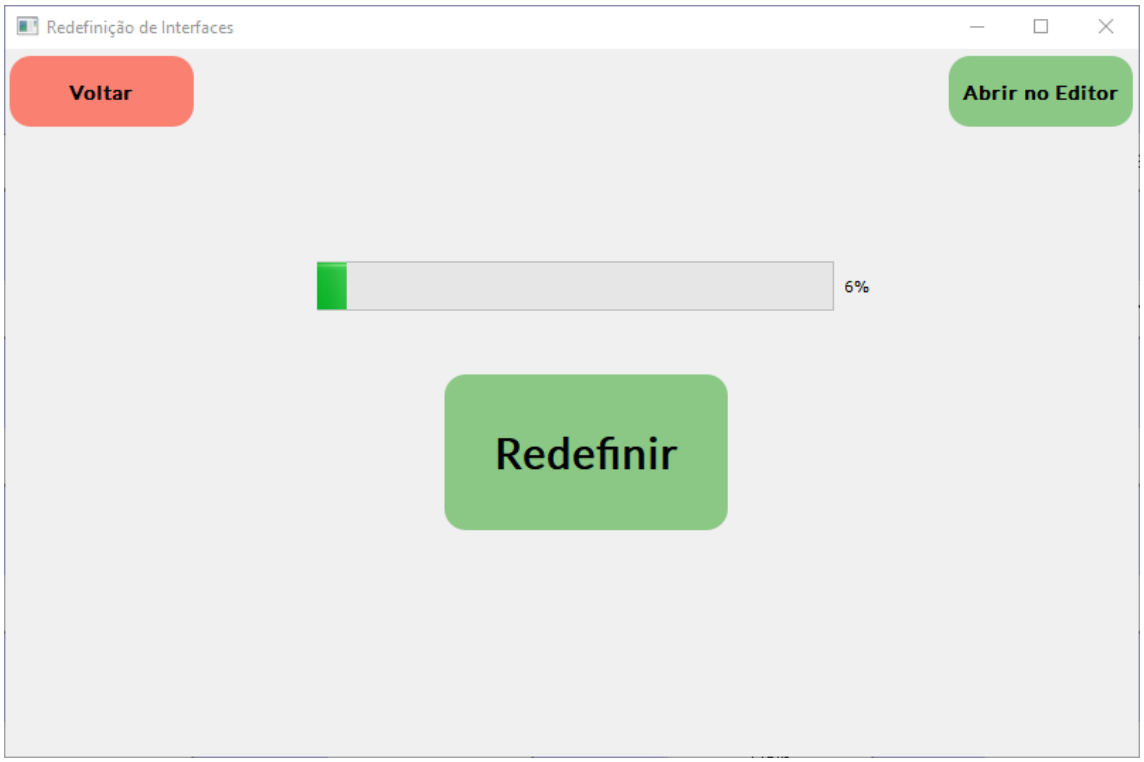

Figura 17 - Modo de edição onde é possível redefinir a interface ou abrir num editor

Apesar de todos estes processos de identificação, extração e o *merge* dos *widgets* serem feitos de forma automática, é necessário para finalizar, recorrer a um editor *python* para fazer correções/melhorias à nova interface. Isto porque com as funções do OpenCV, não é possível abranger todo um conjunto possível de extração dos vários *widgets* possíveis numa interface. Tudo isto levou que por vezes na nova interface faltasse um ou outro *widget* que não tivesse sido processado de forma correta. Todo este processo é feito de forma cíclica, ou seja, o programador sempre que quer processar mais uma interface tem de realizar todos os passos mencionados anteriormente. O modo de edição só se encontra disponível para o utilizador que entre com as credenciais de administrador. Na

Figura 19 está ilustrado um diagrama de sequência, onde é possível ter uma visão mais alargada do processo no modo de edição.

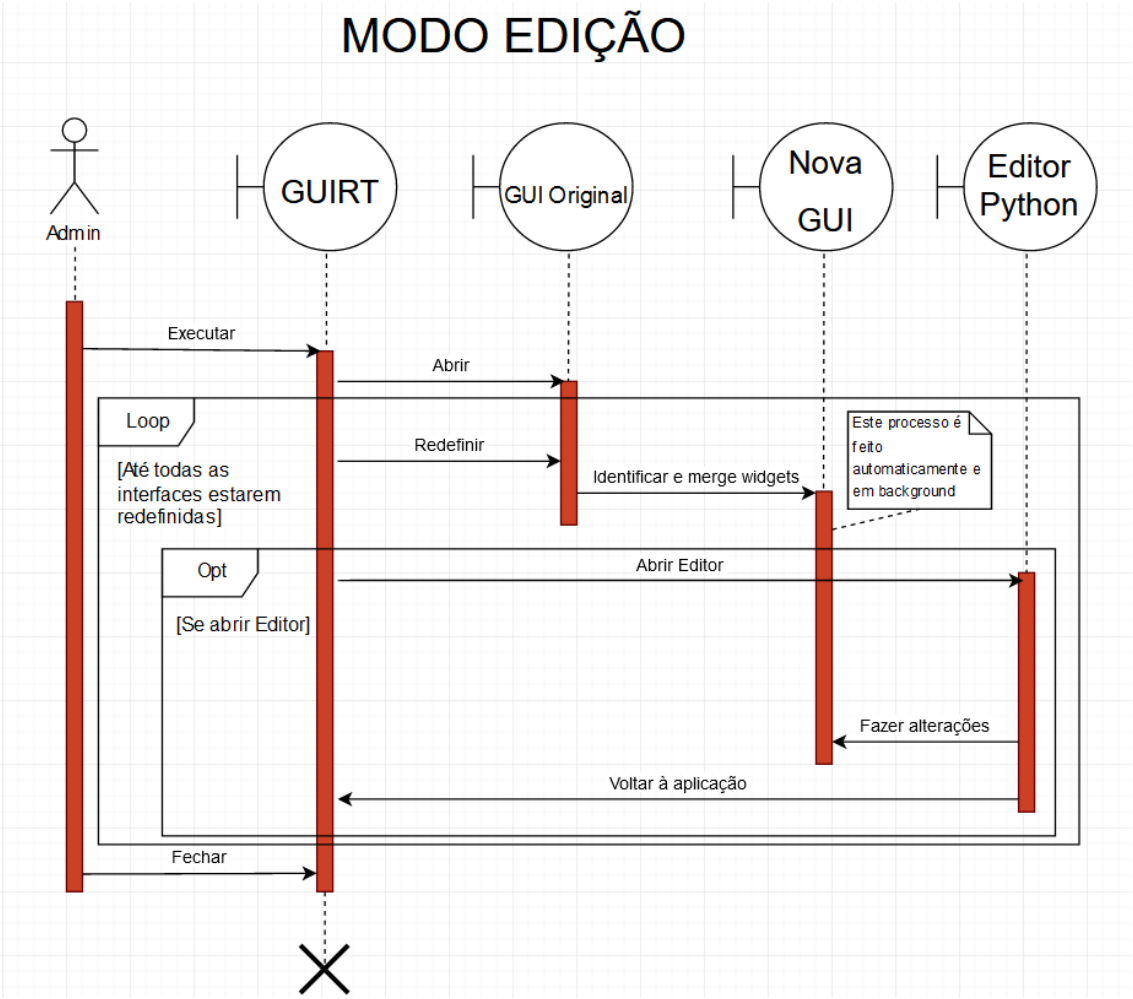

Figura 18 - Diagrama de sequência sobre o modo de edição

3.2.1. Identificação dos *widgets* da interface

Através de vários métodos existentes no OpenCV (irão ser detalhados na secção seguinte) foi possível reconhecer um conjunto de *widgets* numa interface. Nesse conjunto encontram-se a identificação de botões, caixas de texto, *labels* de texto, *dropdowns* e *radiobuttons*.

A forma como foi feita a distinção entre os vários *widgets* encontrados, teve como base a existência de texto no *widget*, a cor e a sua área. Isto é, depois de localizar o *widget*, se não possuir texto, for de cor branca e tiver uma determinada área, então é considerado

Figura 19 - Diagrama de sequência sobre o modo de edição

uma caixa de texto. Caso o *widget* tenha texto e tiver como base pixéis de cor branca então são identificados como *dropdowns*. Se o *widget* não tiver texto, tiver como base pixéis de cor branca e possuir uma área relativamente pequena, então são identificados como *radiobuttons*. O fato de se considerar a área para este *widget* é porque à partida os *radiobuttons* são os *widgets* de tamanho mais pequeno. Por exclusão de partes, o resto dos *widgets* são identificados como botões. De salientar que todo este processo podia ser melhorado, por exemplo, através de técnicas de aprendizagem automática, porém não foram encontrados *datasets* para ser explorada esta possibilidade.

Relativamente à forma como foi feito para extrair a parte das *labels* de texto, para além de se ter usado uma biblioteca específica de *python* - *connectedComponentsWithStats* - foi entender se aquele *widget* extraído possuía alguma borda ou não. No caso de não possuir então é identificado como *label* de texto.

3.2.1.1. Funções Usadas

A deteção e classificação dos *widgets* foi efetuada através dos seus contornos. Os contornos podem ser explicados simplesmente como uma curva unindo todos os pontos contínuos, com a mesma cor ou intensidade. Os contornos são uma ferramenta útil para análise de formas e deteção e reconhecimento de objetos [30].

Inicialmente a abordagem que se seguiu foi a utilização do método MSER (*Maximally stable extremal regions*) para o reconhecimento dos *widgets*, mas os resultados obtidos não foram os melhores. A função que acabou por se utilizar foi o *findContours*.

Foram também utilizadas algumas *features* na deteção dos contornos como o *boundingRect* ou o *contourArea*. Através destas funções é possível saber a localização do contorno, assim como o cálculo da área dos contornos para um processamento posterior (explicação mencionada na secção anterior). Para a deteção das *labels* de texto presentes numa imagem, foi utilizado o método *connectedComponentsWithStats*[2]. Para finalizar foi ainda utilizado a função de python *image_to_string*, pertencente à biblioteca *Python-tesseract* para o reconhecimento de texto e a ferramenta *translate*[3]

[2] https://docs.opencv.org/3.0-beta/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=connectedcomponents (último acesso: Maio de 2019)

[3] https://pypi.org/project/translate/ (último acesso: Maio de 2019)

onde em primeiro lugar indica-se a língua para o qual se quer o texto traduzido e só depois menciona-se o texto que se quer traduzir. Esta lógica da tradução foi aplicada para a funcionalidade na qual se obtém as interfaces traduzidas para a língua que se pretende.

3.2.2. Finalização da interface

Depois de finalizado o processo onde foi feita a extração e identificação dos *widgets* da interface, é necessário passar toda a informação sobre a interface para o formato de XML e construir de raíz esse mesmo ficheiro. Este processo foi necessário, pois tinha-se de passar a nova interface para um editor *python* e como o editor só consegue ler ficheiros num determinado formato, foi necessário implementar esta abordagem. Para além disso, foi preciso entender o modelo e as *tags* específicas para gerar o XML e no fim escrever toda esta informação para um ficheiro com o formato *.ui*. Este formato é específico para o editor *python* escolhido. A Figura 20 ilustra um exemplo de um ficheiro XML que foi gerado pelo GUIRT, contendo a informação sobre a nova interface.

```
<?xml version='1.0' encoding='UTF-8'?>
<ui version="4.0">
  <class>MainWindow</class>
  <widget name="MainWindow" class="QMainWindow">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>1366</width>
        <height>768</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>MainWindow</string>
    </property>
    <widget name="centralwidget" class="QWidget">
      <widget name="pushButton1" class="QPushButton">
        <property name="geometry">
          <rect>
            <x>775</x>
            <y>598</y>
            <width>120</width>
            <height>50</height>
          </rect>
        </property>
        <property name="text">
          <string>Sair</string>
        </property>
      </widget>
      <widget name="lineEdit1" class="QLineEdit">
        <property name="geometry">
          <rect>
            <x>761</x>
            <y>208</y>
            <width>113</width>
            <height>20</height>
          </rect>
        </property>
      </widget>
      <widget name="label" class="QLabel">
        <property name="geometry">
          <rect>
            <x>0</x>
            <y>122</y>
            <width>131</width>
            <height>31</height>
          </rect>
        </property>
        <property name="text">
          <string>habitações terminadas</string>
        </property>
      </widget>
    </widget>
  </widget>
</ui>
```

Figura 20 - Exemplo de um ficheiro xml gerado pelo GUIRT

A forma como este ficheiro foi construído no *python*, teve como base alimentar as *tags* específicas com os valores esperados. Ou seja, primeiro temos a *tag widget* que contem a classe "QMainWindow" para se constuir a janela da interface. Depois através da *tag property* é possível definir o tamanho, o nome e a cor que se pretende para a janela, com

as classes "*geometry*", "*windowTitle*" e "*styleSheet*", respetivamente. De seguida é feito um ciclo às estruturas onde se encontram guardadas todas a informações sobre os *widgets*. Nessas estruturas temos a informação sobre o tipo de *widget*, as suas cordenadas e o texto (caso tenham). Nesse loop adicionamos cada *widget* a uma nova *tag*, na qual se atribui a classe consoante o tipo de *widget*. Neste caso adicionam-se as *tags* com as classes: "*QLineEdit*", "*QComboBox*", "*QPushButton*", "*QLabel*" e "*QRadioButton*". Para cada *widget* também se adiciona ao XML as informações sobre as localizações dos mesmos, através da *tag property* com a classe "*geometry*" e ainda o texto também através da *tag property* mas com o name "*text*". No fim o ficheiro XML fica construído, sendo gravado no formato *.ui*.

Depois da criação e geração deste ficheiro é então possível abri-lo no editor e fazer as alterações que o programador achar necessário para melhorar a interface. De acordo com as figuras Figura 21, Figura 22 e Figura 23, encontra-se ilustrado um exemplo da interface da aplicação original, depois da nova interface gerada pelo GUIRT e por fim a da interface final depois de ter passado por melhorias através de um editor.

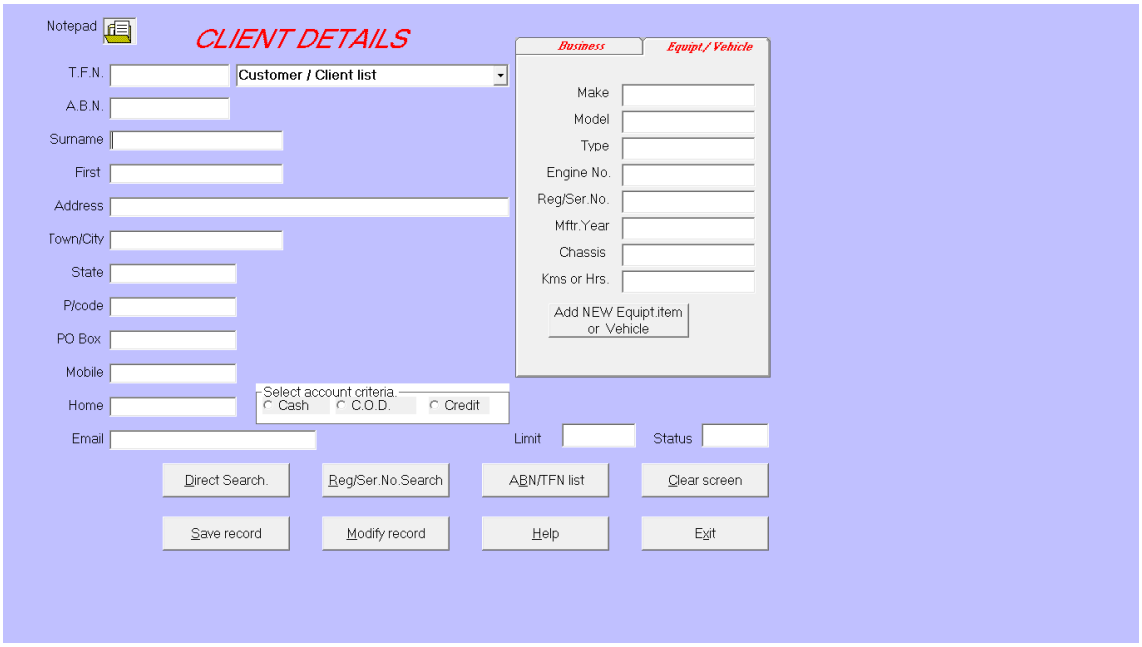

Figura 21 - Exemplo de interface original

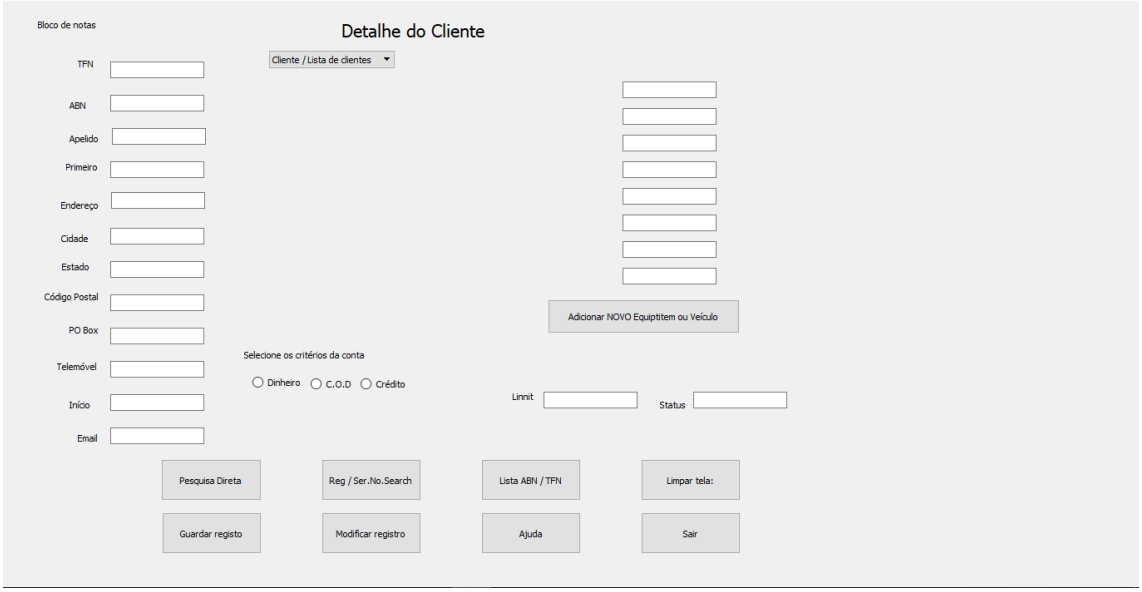

Figura 22 - Interface gerada pelo GUIRT (tendo por base a GUI da Figura 21)

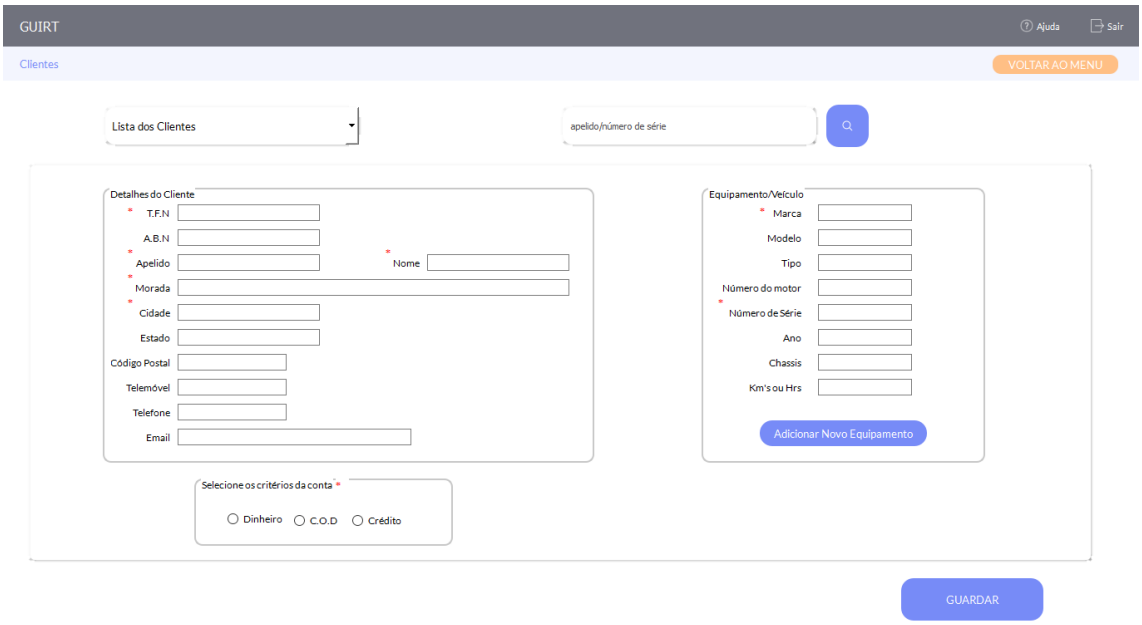

Figura 23 - Interface final depois de ter sido melhorada pelo programador através do editor python

3.2.2.1. QT Editor

Em relação à construção automática da nova GUI, esta pode não ficar perfeita. Por vezes o texto não aparece traduzido da forma correta ou fica a faltar um widget, ou até mesmo um dos widgets pode não ter sido processado da forma correta. Portanto, foi necessário recorrer a uma forma de melhorar a nova interface, que pode ser efetuado

através do editor python, o *QT Designer*[4]. *QT Designer* é uma ferramenta que permite criar interfaces gráficas através do módulo *QT widgets*. Estes fornecem um conjunto de elementos de interface, sendo possível personalizar cada *widget* adicionado [31]. Através deste editor o programador abre o ficheiro XML onde se encontra representada a nova interface (automaticamente gerado pelo GUIRT) e depois faz as modificações necessárias à mesma. Relativamente à escolha sobre este editor, esta teve apenas como base em optar por um editor que permitisse efetuar o que era pretendido e que fosse o mais agradável possível.

3.3. Modo de Execução

Quando todas as novas interfaces da aplicação original estiverem prontas, recorremos à opção modo de *execução* do GUIRT para iniciar a aplicação final com as novas interfaces.

Figura 24 - Modo de execução onde é possível executar a nova interface

[4] https://doc.qt.io/qt-5/qtdesigner-manual.html (último acesso: Maio 2019)

A Figura 24 ilustra a interface do GUIRT onde o utilizador tem a opção de executar a nova interface. Esta opção apenas está disponível para o utilizador final. Depois de a aplicação estar aberta e sempre que o utilizador realiza uma simples ação na nova interface, como por exemplo, carregar num botão, preencher uma *textbox*, etc, é executado um *script* sikuli, que se encontra na máquina virtual, através do servidor *RunServer*. Isto porque a aplicação antiga que foi usada, encontra-se aberta na máquina virtual e portanto quando se faz uma ação na nova interface, essa mesma ação é replicada na interface antiga. Na Figura 25 encontra-se ilustrado um diagrama de sequência, onde é possível ter uma visão mais alargada do processo modo de execução.

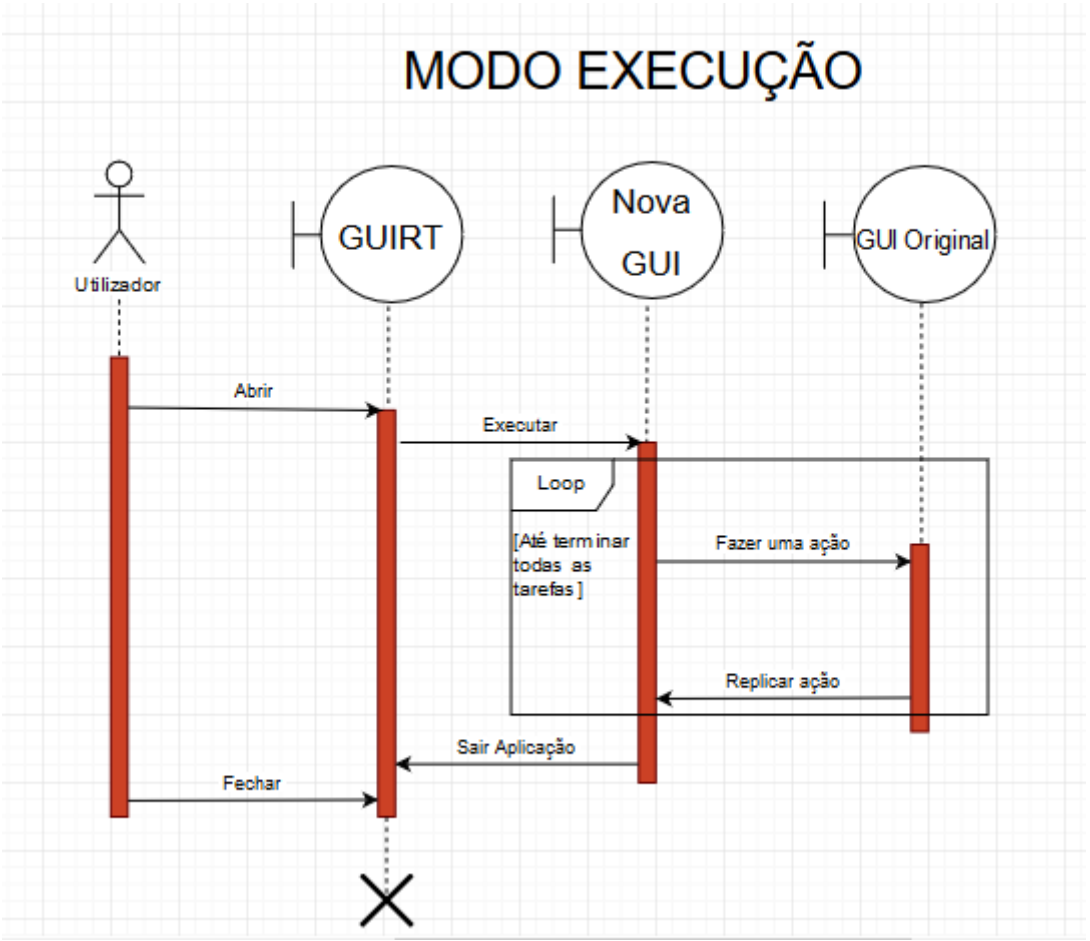

Figura 25 - Diagrama de atividade sobre o modo de execução

3.3.1. Processo de obtenção dos scripts

Para o processo da criação dos *scripts* foi utilizada a ferramenta Sikuli que serve para automatizar um conjunto de tarefas sem a necessidade de recorrer a linguagens de programação tradicionais. Através destes *scripts* é possível replicar um conjunto de passos para efetuar uma dada ação na interface. Estes *scripts* são criados através da invocação de funções *python* já pré-definidas (p.ex. *click()* ou o *find()*), onde depois são associados a essas mesmas funções *screenshots* aos elementos da interface onde se pretendem realizar ações. Em certos *scripts* foi necessário adicionar alguma complexidade para extração de texto ou o seu pré-processamento, assim como também foi necessário criar certos tipos de validações. Para estes casos foi preciso recorrer à linguagem python suportada pelo IDE (*Integrated Development Environment*).

3.3.1.1. Sikuli / Runserver

Sempre que é realizado um evento sobre a nova interface, esse mesmo evento tem de ser replicado na interface original que se encontra na máquina virtual. Mas para isso é necessário comunicar com os *scripts* sikuli que também se encontram na máquina virtual. A atribuição da responsabilidade da execução dos *scripts* é devido à existência de um servidor, de seu nome, *RunServer*. Para iniciar o servidor é necessário abrir a linha de comandos na máquina virtual e executar o seguinte comando:

```
C:\Users\ritaC\Desktop\Sikuli>runsikulix.cmd -s
```

Todo este processo é realizado da seguinte forma: sempre que é criado um novo *script*, por exemplo, para preencher a caixa de texto do nome, é necessário na máquina local criar um ficheiro *.bat*, que contenha o IP da máquina virtual e o diretório exato do script que se quer executar. Ou seja, para cada *script* sikuli criado tem que haver um ficheiro *.bat* associado. Nesse ficheiro é necessário indicar o IP da máquina virtual e o caminho onde se encontra o *script* sikuli. Na Figura 26 está ilustado um exemplo de um ficheiro *.bat* a invocar a *script* sikuli correspondente. Todos os ficheiros *.bat* são invocados no código *python* da GUIRT. Para ir buscar corretamente o *script* sikuli indicado é utilizado o comando *wget*. *Wget* é uma ferramenta de computador criada pelo Projeto GNU. É usado para recuperar conteúdo e arquivos de vários servidores da web. O nome

é uma combinação da *World Wide Web* e da palavra *get*. Suporta *downloads* via FTP, SFTP, HTTP e HTTPS [32].

```
wget "http://192.168.1.74:50001/scripts/home/Desktop/Sikuli/sikulixrunserver"
wget "http://192.168.1.74:50001/run/ESIBIS_clients_surname?%1=value1"
```

Figura 26 - Exemplo de um ficheiro .bat que invoca a script sikuli correspondente

3.3.1.2. Máquina Virtual

Durante o desenvolvimento desta aplicação, apercebeu-se que era necessário recorrer ao uso de *scripts* sikuli sobre a aplicação antiga. Chegando a altura em que são feitas as automatizações dos passos, era necessário que estes estivessem a ser feitos sem a percepção do utilizador. Posto isto, foi necessário recorrer ao uso de uma máquina virtual. Através da máquina virtual a interface original fica escondida aos olhos do utilizador, sendo que as partes do *script* sikuli estão a ser feitas à medida que o utilizador interage com a nova interface. Sempre que é necessário existir transferências de dados entre a máquina virtual e a local, estes são transferidos através de ficheiros de texto. Ou seja, primeiro é criada uma pasta num diretório à escolha na máquina local, depois na máquina virtual nas opções das pastas partilhadas cria-se uma nova com o caminho exato onde se encontra a pasta na máquina local. Portanto sempre que são adicionados ou modificados ficheiros de texto, esta informação automaticamente é sincronizada para a máquina local.

As principais vantagens que se pode retirar da execução dos *scripts* sobre uma máquina virtual, é existir a possibilidade de esconder a realização dos passos ao utilizador, sem haver uma perda de controlo do rato ou do teclado, por parte do mesmo.

3.4. Funcionalidades Adicionais

Para além do que foi descrito ao longo deste capítulo, o GUIRT também possui um conjunto de funcionalidades adicionais, todas elas feitas de forma automática, que foram implementadas com o intuito desta ferramenta poder vir a ser também virada para

utilizadores com necessidades especiais. Sobre este tipo de utilizadores são considerados aqueles que sofrem de dislexia, visão reduzida ou para os daltónicos. Estas funcionalidades não foram submetidas a avaliações por parte dos utilizadores.

A primeira funcionalidade implementada prende-se com a possibilidade de a nova interface poder ser traduzida para as línguas portuguesas, inglesas e espanholas. Foi utilizado a ferramenta *translate*, desenvolvida em *python* onde em primeiro lugar indica-se a língua para o qual se quer o texto traduzido e só depois menciona-se o texto que se quer traduzir. Neste momento é possível traduzir a interface para português, inglês e espanhol.

A segunda funcionalidade implementada foi a opção de existir o *auto complete* do texto para as caixas de texto, através da utilização de uma biblioteca python: *TextBlob*[5] usando o método: *correct* que deteta se existe algum erro de ortografia e se existir, corrige. Esta funcionalidade foi pensada para os utilizadores que sofrem de dislexia. Ou seja, para casos específicos em que os utilizadores têm dificuldades acrescidas na leitura das palavras e que por vezes se enganam a escrever e nem se apercebem. Por exemplo, no formulário para preencher os dados do cliente, no campo cidade escrever a palavra "Mancheter". Quando se carrega na tecla *enter* automaticamente a palavra muda para Manchester.

Por fim o GUIRT também tem uma funcionalidade específica para utilizadores que sofrem de daltonismo/visão reduzida, onde foi incorporado na interface um código de cores, através de um sistema de identificação de cores, colorADD[6]. ColorADD é uma linguagem única que permite a pessoas com visão reduzida ou que sofram de daltonismo, consigam identificar as cores [33]. Relativamente ao GUIRT, a forma como foi implementada esta solução, foi consoante a cor definida para o *widget* em questão, foi identificada uma imagem identificativa (do colorADD) para essa cor que ficou embutida no *widget*.

[5] https://textblob.readthedocs.io/en/dev/quickstart.html#spelling-correction (acesso: Maio 2019)

[6] http://www.coloradd.net/ (acesso: Maio 2019)

Figura 27 - Exemplo aplicado com a funcionalidade colorADD

De acordo com a Figura 27 é possível visualizar como foi aplicada esta funcionalidade nos botões da interface. O botão do lado esquerdo pertence à interface original, enquanto que o botão do lado direito já reflete o resultado final, ou seja, depois da interface estar redefinida. Neste caso como o botão tem a cor cinzenta clara, foi aplicado o ícone que corresponde a essa cor. De realçar que esta funcionalidade só se encontra implementada para os butões, pois o foco era implementar um protótipo de uma interface que não englobasse para já os utilizadores com necessidades especiais.

Capítulo 4 – Caso de Estudo e Avaliação

De acordo com o projeto desenvolvido foi escolhido um caso de estudo, tendo sido realizado uma avaliação com utilizadores. Este estudo teve como objetivo efetuar uma avaliação sobre a viabilidade do GUIRT baseado num caso de estudo, no qual não existiu restrição do público-alvo.

Neste capítulo é então apresentado o caso de estudo escolhido e as duas fases de avaliações que foram realizadas. São apresentados os resultados para o GUIRT e para o caso de estudo, onde é feita uma comparação dos mesmos. No fim foi pedido aos utilizadores que preenchessem um questionário de satisfação sobre a GUI gerada tendo por base o GUIRT. Este questionário encontra-se disponível no Anexo B – Questionário de Satisfação. Para terminar é feita uma discussão crítica sobre os resultados obtidos e apresentado o que podia ter sido feito para melhorar a aplicação.

4.1. ESIBIS

A aplicação escolhida para ilustrar o GUIRT foi o ESIBIS. O ESIBIS é um sistema independente para controlo de custos empresariais. Permite gerir custos de reparação e manutenção nas áreas de eletricidade, mecânica e construção, extrair relatórios e análises de componentes [34]. Este *software* foi o escolhido pois servia para os propósitos desejados, ou seja, tem uma interface antiga, usabilidade melhorável e os *widgets* presentes não possuem um nível alto de complexidade. Isto porque hoje em dia, as interfaces possuem por exemplo botões ou dropdowns já sem relevo, o que aumentaria a complexidade na fase do reconhecimento dos *widgets*. Portanto, o foco inicial foi utilizar um subconjunto de widgets (mais simples) para demonstrar a viabilidade da solução.

4.2. Descrição da Avaliação

A fase de avaliação foi pensada e planeada para ser dividida em duas partes. Inicialmente foi tido em conta uma abordagem crítica em perceber qual das aplicações

seria a primeira a ser testada. Por um lado se começássemos pelo ESIBIS, depois o utilizador ao testar a solução gerada pelo GUIRT poderia ser influenciado nos seus passos pela aprendizagem adquirida (*carryover effect* [35]) do que teria que executar, podendo assim afetar a fidelidade das avaliações. O mesmo efeito se aplicaria caso o utilizador começasse com a solução gerada pelo GUIRT primeiro. Segundo estas evidências foi decidido que era preferível iniciar os testes pelo GUIRT, ou seja, para não comprometer em nada (*carryover effect*) os passos a realizar na nova ferramenta, onde foi centrado o foco do desenvolvimento.

A aplicação ESIBIS foi testada na segunda fase de testes em que a execução dos passos eram exatamente iguais aos anteriores, para o GUIRT. Tendo-se optado por esta ordem houve a perfeita noção de que a facilidade em executar alguns passos no ESIBIS foi sujeita a um conhecimento já pré-adequirido, pois já tinham testado o GUIRT. Apesar de ambas as aplicações em termos de interface estarem bastante distintas. Ainda assim e de acordo com os resultados dos testes, foi detetado que certas tarefas foram concluídas com dificuldades ou até mesmo nem sequer chegaram a ser concluídas de todo. De mencionar que houve uma pausa entre a execução das duas fases de testes, pois foi necessário preparar para o utilizador o ESIBIS sem quaisquer dados gravados, isto para que o mesmo pudesse efetuar os mesmos passos e gravar a mesma informação.

Foi então elaborado um documento contendo todos os passos a realizar para a solução gerada pelo GUIRT e ESIBIS. Foi pedido a cada utilizador para realizarem três tipos de tarefas: criar um cliente, criar uma ordem de trabalho e no fim pesquisar o cliente criado. De forma voluntária, onde se disponibilizaram para realizar os testes (sem compensação), obteve-se um conjunto de 23 participantes com idades compreendidas entre 20 e os 58 anos, onde uma percentagem de 65,2% de homens e 34,8% de mulheres e onde todos eles tinham formação escolar/académica desde o secundário até ao doutoramento. Quanto ao uso de tecnologias de informação apenas 4,3% utilizam aplicações/*sites* numa base diária de 3 ou 4 dias por semana, sendo que o resto dos utilizadores, 95,7%, utilizam diariamente. Como as aplicações estavam na máquina local de desenvolvimento, o utilizador teve que realizar os testes nessa mesma máquina. As avaliações foram realizadas na minha casa ou em alguns casos, na casa do utilizador. Os utilizadores não obtiveram qualquer ajuda na execução dos passos, ou seja, guiando-se apenas pelo documento fornecido. A execução dos mesmos foi efetuada e gravada

através duma ferramenta *online* para gravação de vídeos[7]. A versão final do documento encontra-se no Anexo A – Documento de Avaliação.

4.3. Resultados

As métricas pensadas e definidas para demonstrar os resultados foram: número de erros cometidos, tempo total de cada tarefa e se terminou a tarefa. Uma métrica adicional foi o resultado das respostas a um questionário de satisfação, que foi preenchido logo após a execução dos testes.

Para analisar os resultados foi utilizado o teste do *t-student*, na qual uma das condições é que os dados devem seguir uma distribuição normal. Através do uso do teste do *t-student* é possível verificar se determinada hipótese é aceite ou rejeitada.

O *t-student* ou somente *t-test* é um teste de hipótese que usa conceitos estatísticos para rejeitar ou não uma hipótese nula quando a estatística de teste (t) segue uma distribuição *t-student* [36]. A distribuição de *t-student* é essencialmente uma distribuição normal para todas as amostras de tamanho "n" [37]. Depois de executado é necessário verificar o *p-value*. Através do *p-value* é possível verificar se uma hipótese é aceite ou rejeitada. De seguida é necessário definir um valor de alfa que por defeito ou é 0.01 ou 0.05 e portanto compara-se o *p-value* com 0.05 ou 0.01 para verificar se é possível aceitar ou rejeitar uma hipotese com 95% ou 99% de confiança, respetivamente. Para a métrica do término da tarefa foi necessário converter os dados para valores binários e para estes casos foi utilizada outra abordagem para a análise dos dados estatisticamente. Inicialmente foi pensado utilizar o teste z, mas para prosseguir com esta abordagem era necessário ter uma amostra de pelo menos 30 pessoas, para que os resultados fossem minimamente fiáveis. Portanto, recorreu-se ao uso do teste *McNemar*, que também é capaz de analisar dados binários.

Relativamente à primeira hipótese formulada - **H1: A alteração da interface gráfica diminui o nº de erros cometidos?** - esta aborda a questão de investigação mencionada

[7] https://screencast-o-matic.com/screen-recorder (último acesso: Junho 2019)

no capítulo 1, pois permite perceber se com a nova interface são introduzidas melhorias, neste caso, se o número de erros diminui com a nova ferramenta. Posto isto, para a primeira hipótese, aplicando o *t-student* e de acordo com a Tabela 3 em abaixo, estes foram os resultados obtidos.

Teste T: Duas amostras com variâncias iguais

Número de Erros Cometidos		
	GUIRT	**ESIBIS**
Média	0,304	0,652
P(T <=t) uni-caudal	0,021	

Tabela 3 - Resultados da H1 depois de se ter aplicado o t-student

De acordo com a realização do t-student, para o caso do número de erros cometidos, podemos verificar pela imagem acima que o valor de p uni-caudal é igual a aproximadamente 0,02. Logo afirmamos que a hipótese "a alteração da interface gráfica diminui o nº de erros cometidos?" é aceite com 95% de confiança.

A segunda hipótese formulada - **H2: A alteração da interface gráfica permite a diminuição do tempo para realizar cada tarefa? -** também aborda a questão de investigação, pois permite perceber se com a nova interface são introduzidas melhorias, neste caso, se existe uma diminuição no tempo da execução das tarefas com a nova ferramenta. A Tabela 4 apresenta os resultados obtidos aplicando o teste de *t-student*.

Teste T: Duas amostras com variâncias iguais

Tempo na Realização das Tarefas		
	GUIRT	**ESIBIS**
Média	91,159	83,841
P(T <=t) uni-caudal	0,169	

Tabela 4 - Resultados da H2 depois de se ter aplicado o t-student

De acordo com a realização do t-student, para o caso do tempo a realizar todas as tarefas, podemos verificar pela imagem acima que o valor de p uni-caudal é igual a aproximadamente 0,17. Logo afirmamos que a hipótese "A alteração da interface gráfica permite a diminuição do tempo para realizar cada tarefa? " não é aceite com 95% de confiança. Estes resultados prendem-se pelo fato do utilizador só conseguir avançar em certos passos na interface, quando o *script* sikuli estiver concluído.

Por fim, a terceira hopótese formulada – **H3: Os resultados para o término de todas as tarefas em ambas as interfaces são iguais? -** também responde à questão de investigação, pois permite perceber se com a nova interface são introduzidas melhorias, neste caso, se as tarefas são executadas com sucesso na nova ferramenta. A Tabela 5 apresenta os resultados obtidos aplicando o teste *McNemar*.

Teste McNemar (valor de p exato) / test bi-caudal

Término das Tarefas		
Valor p (bi-caudal)	1,000	

Tabela 5 - Resultados da H3 depois de se ter aplicado o teste McNemar

De acordo com a realização do teste *McNemar*, podemos verificar pela Tabela 5 que o valor de p bi-caudal é igual a 1, ou seja maior que o valor de alpha 0,05. Portanto esta hipotese não é rejeitada, mostrando assim que, a diferença nos resultados para o término de todas as tarefas em ambas as interfaces, não é estatisticamente diferente, podendo afirmar com certezas em 95% dos casos.

Relativamente ao questionário de satisfação, este foi dividido em 3 critérios: utilidade, fácil de usar e satisfação. De acordo com o que foi preenchido e segundo o resultado dos gráficos (Figura 28), podemos constatar que houve uma reação positiva à nova GUI, onde todos os critérios obtiveram uma moda de 5. Podemos então assumir que de um modo geral os participantes ficaram bastantes satisfeitos com a nova GUI.

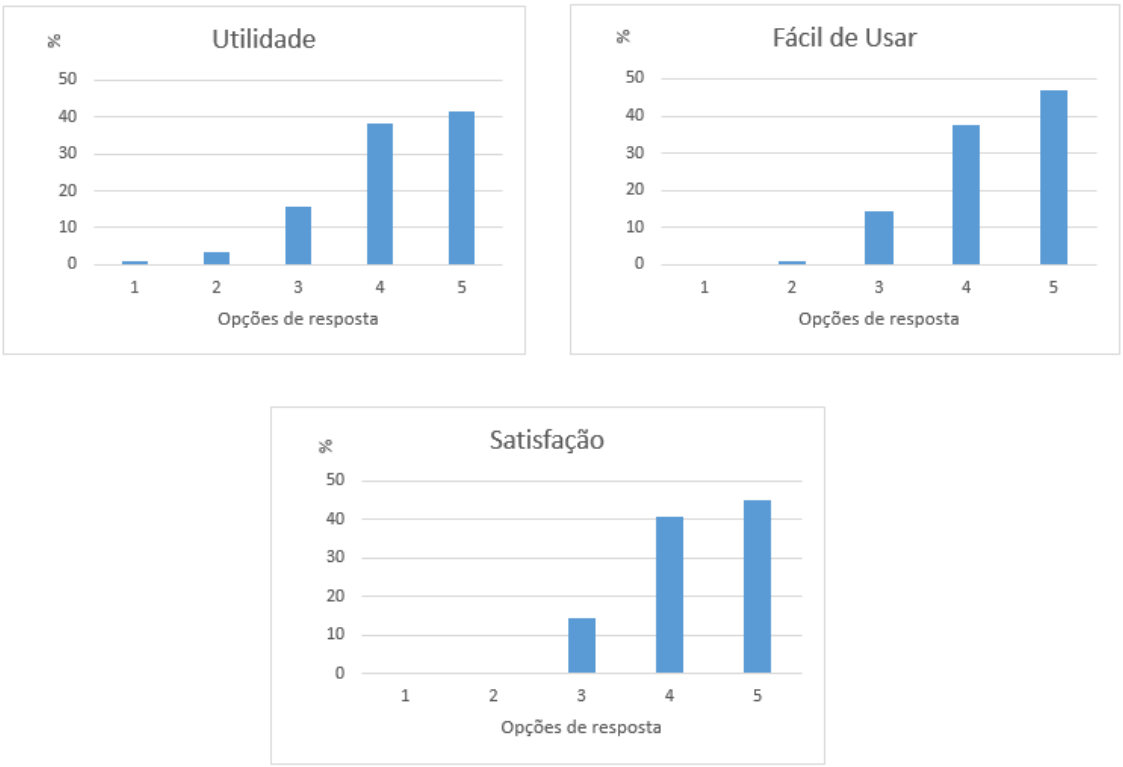

Figura 28 - Gráficos com a extração dos resultados do questionário

No final do questionário ainda existiam questões de resposta aberta, em que muitos dos participantes deram ênfase à simplicidade e forma fácil de interagir com a interface, assim como o aspeto gráfico e apelativo da mesma. O aspeto menos bom e o qual foi dado mais relevo no questionário, foi o fato de existir um elevado tempo de espera em determinadas ações. Por exemplo, um participante fez o seguinte comentário: "tempo de espera alto fazendo o utilizador clicar mais do que uma vez no botão". Os participantes também destacaram positivamente a organização, facilidade de uso e aparência da nova GUI. Um participante apontou como positivo o "*layout* simples e fácil de usar".

4.4. Discussão

Elaborando uma análise crítica de como foi pensada a melhor forma de executar os testes em termos de ordem e de acordo com os resultados obtidos, chegamos à conclusão que, em termos do tempo para executar uma tarefa, não difere assim tanto duma aplicação para a outra, porque apesar de na solução gerada via GUIRT a execução de certos passos demorarem bastante tempo, devido à espera que as scripts sikuli

terminassem, o fato de no ESIBIS a interface não ser nada intuitiva e apelativa, faz com que o utilizador fique confuso em certas ocasiões. Tudo isto também leva ao aumento do número de erros cometidos no ESIBIS, assim como se o utilizador consegue ou não terminar uma dada tarefa. Verifica-se também que pelos resultados, é evidente que houve muitos mais erros cometidos no ESIBIS. Um dos erros mais comuns realizados pelos utilizadores foi gravar a ordem de trabalho sem ter preenchido todos os campos obrigatórios. Outro erro também muito comum foi o utilizador carregar mais do que uma vez no botão para guardar o cliente, quando este já tinha sido gravado no primeiro *click* do mesmo.

Relativamente aos resultados globais do questionário de satisfação, é certo que existe uma clara satisfação na utilização da ferramenta GUIRT, assim como na sua forma de utilização, no qual os utilizadores, dum modo geral, acharam fácil e acessível de usar. Sobre as funcionalidades que foram implementadas a pensar nos utilizadores com necessidades especiais, estas não foram testadas pois era necessário arranjar um conjunto com esse tipo de utilizadores. Sendo que o foco principal foi a implementação de um protótipo capaz de redefinir uma interface para um utilizador "normal".

Com isto, é possível também demonstrar que quando uma interface é simples, intuitiva, onde o conteúdo se encontra bem organizado, facilita muito mais a forma como o utilizador a utiliza, diminuindo assim o número de erros ou de possíveis tarefas mal realizadas. Dando um exemplo prático, o fato de a interface no GUIRT ter visíveis quais os campos obrigatórios, dá uma maior clareza ao utilizador quais os campos que deve preencher *à priori*. Outro exemplo foi a existência de um botão para voltar atrás e outro para sair imediatamente da aplicação. Isto fez com que houvesse uma percepção clara de onde o utilizador tem de carregar para terminar a sua tarefa. Isto porque no ESIBIS em cada interface, existe apenas o botão "*Exit*", o que levou a uma certa confusão por parte dos utilizadores em entenderem, isto é, se ao carregarem nesse mesmo botão se iriam sair efetivamente da aplicação ou se era apenas para voltar para trás. Todos estes aspetos acabaram por atenuar o fato de existir alguma lentidão na ferramenta quando se pretendia gravar ou pesquisar um cliente, pois este foi o aspeto negativo mais mencionado nos comentários das avaliações. No entanto, é necessário realçar que nos dias de hoje é praticamente inaceitável um utilizador ficar largos segundos à espera que uma dada tarefa fique concluída (p.ex. no GUIRT quando o utilizador pesquisa um cliente, em média fica à espera cerca de 7 segundos). Este aspeto menos bom da

ferramenta sempre foi discutido e pensado ao longo do desenvolvimento da mesma, sendo que irá ser algo que no futuro será trabalhado para melhorar a experiência do utilizador.

Capítulo 5 – Conclusões

Para finalizar, é de extrema relevância dar a conhecer o conjunto de ilações a que foi possível se chegar, a partir de todo o trabalho descrito ao longo deste documento. Assim, importa não só entender e discutir os pontos de maior saliência, tanto positivos como negativos, apresentar as conclusões alcançadas e ainda falar sobre as limitações e possível trabalho futuro.

5.1. Principais conclusões

De acordo com os resultados finais, claramente houve uma preferência pela GUI suportada pelo GUIRT, ou seja, os utilizadores sentiram-se mais à vontade para completar as tarefas que lhes foram propostas. Isto é, acharam a interface mais apelativa, intuitiva e com maior facilidade em executar as ações. Para além de a interface ter um aspeto muito mais *clean* e *user-friendly*, esta também realça aspetos fundamentais a ter em conta aquando da utilização de uma interface. Apesar dos exemplos que foram dados e que potenciaram a nova interface, o fato de existir alguma lentidão aquando uma ação específica, faz com que seja um aspeto importante para ter em conta no futuro. Em relação aos resultados obtidos, foi possível comprovar estatisticamente que existiu uma diminuição do número de erros cometidos, perante a utilização do GUIRT e sobre o tempo realizado nas tarefas, os resultados demonstraram que não houve diferenças entre o GUIRT e o ESIBIS. Para além disso também ficou provado estatisticamente que não houve uma diferença nos resultados para o término de todas as tarefas em ambas as interfaces.

A base da realização deste trabalho foi entender que a usabilidade, a simplicidade no *design* e a organização de uma interface, têm um grande impacto na sua utilização. Este trabalho foi desenvolvido no sentido de ajudar a melhorar interfaces já bastante arcaicas e sem acesso ao seu código fonte mas que muitas entidades e empresas utilizam no seu dia-a-dia para gerirem os seus negócios. Temos como exemplo que, ainda hoje em dia, várias instituições utilizam *softwares* com interfaces muito limitadas (p.ex: vários

sistemas de IT do governo dos Estados Unidos[8]) e que com esta ferramenta, o GUIRT, é possível criar uma nova aplicação, com base na aplicação antiga. Criando interfaces mais apelativas e com maior usabilidade, melhorando o seu *design* e fazendo com que seja muito mais fácil de elaborar uma tarefa. Estes são os casos em que as empresas não pretendem investir dinheiro no *upgrade* do seu *software*. Para estas situações, esta ferramenta oferece a possibilidade às empresas de continuarem a ter o seu *software* mas com melhorias significativas na interface, dando uma nova experiência aos utilizadores. Além disso com o GUIRT não é necessário aceder ao código fonte.

Para concluir, a grande questão que foi colocada no capítulo 1 foi respondida com sucesso, visto que a utilização do GUIRT contribuiu para uma clara melhoria em termos de usabilidade na interface, sem o acesso ao código fonte da aplicação. Também contribuiu para uma maior experiência entre o utilizador e a nova interface. Para além disso todos os objetivos foram cumpridos, apesar de haver uma percepção de que esta ferramenta ainda pode ir muito mais além e ser cada vez mais abrangente.

5.2. Limitações

Este projeto teve como base o desenvolvimento de um protótipo que permite ao utilizador passar de uma interface antiga para uma nova, muito mais apelativa e com maior usabilidade. Mas como todos os protótipos iniciais, este possui algumas limitações, isto é, a ferramenta consegue detetar apenas um conjunto limitado de *widgets*. Dentro desse subconjunto existe uma deficiência na lógica de deteção das caixas de texto. A lógica atual tem em conta a área do objeto, a sua cor e se possui texto, mas a variável da cor não se deveria ter em conta, pois as caixas de texto podem ter qualquer cor. A forma como esta lógica está implementada limita apenas o reconhecimento de caixas de texto que tenham como fundo a cor branca. Para este ponto será necessário arranjar uma solução alternativa, para tornar este processo aplicável de forma genérica. Para além disso, a ferramenta que é usada para fazer a

[8] Relatório do Escritório de Contabilidade do Governo dos Estados Unidos: https://www.gao.gov/assets/680/677454.pdf (último acesso: Maio 2019)

tradução da interface numa determinada linguagem possui algumas falhas, isto porque o método usado para o reconhecimento de texto, em algumas ocasiões, não é eficaz. Outro ponto importante de realçar em relação à deteção dos *widgets* é que, hoje em dia, estes aparecem nas interfaces praticamente sem relevo. Por exemplo, temos o caso de botões que já não possuem qualquer bordo, sendo que, para o utilizador perceber se é um botão ou não, ou é porque tem uma imagem, ou então passando o rato por cima percebe que há uma mudança de cor. Para estes casos, a versão atual da ferramenta não consegue detetar de forma correta os *widgets*. De realçar também que a funcionalidade específica para os utilizadores que sofrem de daltonismo/visão reduzida, o colorADD, apenas funciona para os botões, sendo necessário continuar a implementação para o resto dos *widgets*. Esta funcionalidade encontra-se incompleta, pois o objetivo principal era desenvolver numa primeira fase um protótipo genérico para os utilizadores "normais".

Por fim, e esta sendo a limitação mais evidente e que requer uma maior necessidade de preocupação numa fase inicial, é o fato de o GUIRT necessitar de correr os *scripts* sikuli para realizar as tarefas na interface, e isso faz com que o processo de interação na nova GUI não seja tão rápido e fluído. Ainda assim com a utilização do *RunServer*, o processo de execução dos *scripts* torna-se mais eficiente, pois inicialmente o que era feito era guardar a script num ficheiro executável *.skl* e depois no código *python*, sempre que existisse uma ação para ser feita na interface, invocava-se esse executável. Porém, só o tempo até o executável abrir e depois executar a script, demorava-se mais tempo. Apesar da melhoria significativa com a utilização do *RunServer*, é visível que este é um dos pontos que tem de ser investido e melhorado, para uma melhor experiência entre o utilizador final e a nova interface.

5.3. Trabalho Futuro

Para futuros trabalhos poderá ser investido mais tempo no tema do processamento de imagem, com o intuito de abranger mais *widgets* e consequentemente ser capaz de detetar um maior número de interfaces e duma forma mais abrangente.

Outro possível trabalho futuro, seria arranjar um conjunto de pessoas para participarem nas avaliações que estivessem inseridas no grupo dos utilizadores com necessidades especiais, visto que o GUIRT é capaz de adaptar interfaces para esse tipo de utilizadores

(e.g. com dislexia, visão reduzida ou daltónicos). Para que assim existisse mais investimento nesta área, visto que em termos de funcionalidades o GUIRT ainda se encontra um pouco limitado.

Acima de tudo o grande objetivo é que no fim o GUIRT seja uma ferramenta capaz de redefinir qualquer tipo de interface (sem acesso ao seu código fonte), independentemente da sua complexidade, e adaptada a qualquer tipo de utilizador.

Bibliografia

[1] DE KOCK, Estelle; VAN BILJON, Judy; PRETORIUS, Marco. Usability evaluation methods: Mind the gaps. In: Proceedings of the 2009 annual research conference of the south african institute of computer scientists and information technologists. ACM, 2009. p. 122-131.

[2] SHNEIDERMAN, Ben. Shneiderman's eight golden rules of interface design, 2009. [online]. [Accessed 18 June 2019]. Available from: http://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html

[3] FDIS, ISO: 9241-110: Ergonomics of human system interaction-Part 110: Dialogue principles. International Organization for Standardization (ISO), Switzerland, 2009

[4] DIS, ISO. 9241-210: 2010. Ergonomics of human system interaction-Part 210: Human-centred design for interactive systems. International Standardization Organization (ISO). Switzerland, 2009

[5] MEISELWITZ, Gabriele, et al. Universal usability: Past, present, and future. Foundations and Trends® in Human–Computer Interaction, 2010, 3.4: 213-333.

[6] SCHNEIDER, Nicole, et al. Foundations of an age-differentiated adaptation of the human-computer interface. Behaviour & Information Technology, 2008, 27.4: 319-324.

[7] TRACTINSKY, Noam, et al. Evaluating the consistency of immediate aesthetic perceptions of web pages. International journal of human-computer studies, 2006, 64.11: 1071-1083.

[8] YAMAMOTO, Thiago Toshiyuki Izumi. Análise e avaliação da usabilidade de interfaces gráficas dos sistemas de Gestão hospitalar. Dissertação (Mestrado em Ciências) – Escola Paulista de Medicina, Universidade Federal de São Paulo. São Paulo, 2013.

[9] SILVA, José Luís; SILVA, J. C. Graphical User Interface Redefinition Addressing Users' Diversity. In: International Conference on Human-Centred Software Engineering. Springer, Cham, 2018. p. 319-326.

[10] HABIEB-MAMMAR, H.; SEFFAH, A. Universal Usability: Characterization and Measurement. Department of Computer Science and Software Engineering Concordia University, (sem ano)

[11] ANBARJAFARI, Gholamreza, 1. Introduction to image processing. Sisu@UT [online]. (sem ano) [Accessed 18 June 2019]. Available from: https://sisu.ut.ee/imageprocessing/book/1

[12] PODA, Xhensila; QIRICI, Olti. Shape Detection and Classification Using OpenCV and Arduino Uno. In: RTA-CSIT. 2018. p. 128-136.

[13] TUTORIALSPOINT.COM, OpenCV Simple Threshold. www.tutorialspoint.com [online]. 2019 [Accessed 18 June 2019]. Available from: https://www.tutorialspoint.com/opencv/opencv_simple_threshold.htm

[14] OpenCV shape detection. PyImageSearch [online], 2019 [Accessed 18 June 2019]. Available from: https://www.pyimagesearch.com/2016/02/08/opencv-shape-detection/

[15] OpenCV. OpenCV [online], 2019 [Accessed 18 June 2019]. Available from: https://opencv.org/about.html

[16] CVIPTools [online], 2017 [Accessed 18 June 2019]. Available from: https://wiki.tcl.tk/40608

[17] VLFeat.org. VLFeat [online], 2007 [Accessed 18 June 2019]. Available from: http://www.vlfeat.org/

[18] Computer Vision Toolbox. MATLAB & Simulink [online], 2019 [Accessed 18 June 2019]. Available from: https://www.mathworks.com/products/computer-vision.html

[19] Contours - 1: Getting Started. OpenCV [online]. 2012 [Accessed 18 June 2019]. Available from: http://opencvpython.blogspot.com/2012/06/hi-this-article-is-tutorial-which-try.html

[20] KOUROUSIAS, George; BONFIGLIO, Silvio. Picture-Driven Computing In Assistive Technology And Accessibility Design. In: 1st International ÆGIS Conference. 2010. p. 8.

[21] YEH, Tom; CHANG, Tsung-Hsiang; MILLER, Robert C. Sikuli: using GUI screenshots for search and automation. In: Proceedings of the 22nd annual ACM symposium on User interface software and technology. ACM, 2009. p. 183-192.

[22] CHANG, Tsung-Hsiang; YEH, Tom; MILLER, Robert C. GUI testing using computer vision. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2010. p. 1535-1544.

[23] DIXON, Morgan; NIED, Alexander; FOGARTY, James. Prefab layers and prefab annotations: extensible pixel-based interpretation of graphical interfaces. In: Proceedings of the 27th annual ACM symposium on User interface software and technology. ACM, 2014. p. 221-230.

[24] DIXON, Morgan; LEVENTHAL, Daniel; FOGARTY, James. Content and hierarchy in pixel-based methods for reverse engineering interface structure. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2011. p. 969-978.

[25] DUBROVINA, Anastasia, et al. Efficient and robust image descriptor for GUI object classification. In: Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012). IEEE, 2012. p. 3594-3597.

[26] GAJOS, Krzysztof; WELD, Daniel S. SUPPLE: automatically generating user interfaces. In: Proceedings of the 9th international conference on Intelligent user interfaces. ACM, 2004. p. 93-100.

[27] ZHANG, Xiaoyi, et al. Interaction proxies for runtime repair and enhancement of mobile application accessibility. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. ACM, 2017. p. 6024-6037.

[28] ZHANG, Xiaoyi; ROSS, Anne Spencer; FOGARTY, James. Robust Annotation of Mobile Application Interfaces in Methods for Accessibility Repair and Enhancement. In: The 31st Annual ACM Symposium on User Interface Software and Technology. ACM, 2018. p. 609-621.

[29] SILVA, José Luís; ORNELAS, Jorge Diogo; SILVA, João Carlos. Make it ISI: interactive systems integration tool. In: Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. ACM, 2016. p. 245-250.

[30] Structural Analysis and Shape Descriptors. Structural Analysis and Shape Descriptors - OpenCV 2.4.13.7 documentation [online], 2019 [Accessed 18 June 2019]. Available from: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

[31] Qt Designer Manual [online], 2019 [Accessed 18 June 2019]. Available from: https://doc-snapshots.qt.io/qt5-5.9/qtdesigner-manual.html

[32] B., Gediminas, What is the Wget Command and How to Use It (12 Examples Included). Hostinger Tutorials [online]. 2018. [Accessed 18 June 2019]. Available from: https://www.hostinger.com/tutorials/wget-command-examples/

[33] -- ColorADD --. ColorADD [online], 2010 [Accessed 18 June 2019]. Available from: http://www.coloradd.net/about.asp

[34] ESIBIS: http://esibis.com: Free Download, Borrow, and Streaming. Internet Archive [online], 2003 [Accessed 18 June 2019]. Available from: https://archive.org/details/tucows_314329_ESIBIS

[35] TORRICO, Damir, et al. Novel Modelling Approaches to Characterize and Quantify Carryover Effects on Sensory Acceptability. Foods, 2018, 7.11: 186.

[36] Teste t de Student. Wikipedia [online], 2018 [Accessed 18 June 2019]. Available from: https://pt.wikipedia.org/wiki/Teste_t_de_Student

[37] Eecis.udel.edu. [online], 2019 [Accessed 18 Jun. 2019]. Available from: https://www.eecis.udel.edu/~portnoi/classroom/prob_estatistica/2006_1/lecture_slides/aula17.pdf

[38] WANG, L.; SOUSA, W. P.; GONG, P. Integration of object-based and pixel-based classification for mapping mangroves with IKONOS imagery. International Journal of Remote Sensing, 2004, 25.24: 5655-5668.

Anexos

Anexo A – Documento de Avaliação

Documento de Avaliação

Introdução:

Este documento tem como objetivo a realização de uma avaliação sobre as funcionalidades da aplicação GUIRT (GUI Redefinition Tool) e perceber se é uma mais-valia para substituir aplicações antigas nas quais ainda têm muitos aspetos para melhorar, quer em termos de processos de *workflow* e também de usabilidade.

A avaliação irá ser dividida em duas partes. A 1ª parte consiste em executar a nova aplicação, GUIRT e a 2ª parte em executar a aplicação ESIBIS.

A interação desta avaliação irá ser gravada, mas com o intuito de ser anónima e no fim pressupõe responder a um questionário sobre a aplicação desenvolvida.

Ao continuar este estudo estou a consentir o que foi descrito acima.

Testar GUIRT:

CRIAR UM CLIENTE:

Passos:

1. Visualizar a interface do Menu Principal.
2. Carregar no botão “Clientes”.
3. Visualizar a interface dos Detalhes do cliente.
4. Preencher os campos obrigatórios e criar um novo cliente.
    a. Terá que aguardar até ser guardado o cliente criado. Este processo poderá demorar alguns segundos. Irá ver o rato com o símbolo de espera.

Dados:

TFN: 124578

Apelido: Gomes

Primeiro Nome: Ricardo

Endereço: Avenida Elias Garcia n5

Cidade: Lisboa

Critério da conta: Dinheiro

Marca: Audi

Número de Série: 1245

CRIAR UMA ORDEM DE TRABALHO:

Passos:

1. Voltar atrás para o menu inicial.
2. Carregar em Ordens de Trabalho.
3. Visualizar a interface das Ordens de Trabalho.
4. Selecione o cliente anteriormente criado.
    a. Terá que aguardar até serem carregados os clientes criados. Este processo poderá demorar alguns segundos. Irá ver o rato com o símbolo de espera.
5. Crie uma nova ordem de trabalho e guarde no fim.
    a. Terá que aguardar até ser guardada a ordem de trabalho criada. Este processo poderá demorar alguns segundos. Irá ver o rato com o símbolo de espera.

Dados:

Ordem de Trabalho: Instalacao do equipamento de luzes

PESQUISAR UM CLIENTE:

Passos:

1. Voltar atrás para o menu inicial.
2. Carregue em Clientes.
3. Pesquise o cliente que criou no passo anterior.
    a. Este processo poderá demorar alguns segundos. Irá ver o rato com o símbolo de espera.
4. Assim que visualizar a informação desejada fechar a aplicação.

Testar o ESIBIS:

CRIAR UM CLIENTE:

Passos:

1. Visualizar a interface do "Main Menu"
2. Carregar no botão "Clients".
3. Visualizar a interface dos "Client Details"
4. Preencher os campos obrigatórios e criar um novo cliente.

Dados:

TFN: 124578

Surname: Gomes

First: Ricardo

Address: Avenida Elias Garcia n5

Town/City: Lisboa

Account Criteria: Cash

Make: Audi

Reg/Ser. No: 1245

CRIAR UMA ORDEM DE TRABALHO:

Passos:

1. Voltar atrás para o "Main Menu".
2. Carregar em "Work Orders".
3. Selecione o cliente anteriormente criado.
4. Crie uma nova ordem de trabalho e guarde no fim.

Dados:

Ordem de Trabalho: Instalação do equipamento de luzes

PESQUISAR UM CLIENTE:

Passos:

1. Voltar atrás para o "Main Menu".
2. Carregue em "Clients".
3. Pesquise o cliente que criou no passo anterior.
4. Assim que visualizar a informação desejada fechar a aplicação.

Anexo B – Questionário de Satisfação

Redefinição de Interfaces Gráficas

Questionário sobre o uso da aplicação GUIRT no âmbito do mestrado de engenharia informática

*Obrigatório

Idade *

Sua resposta

Género *

- Masculino
- Feminino

Profissão *

Sua resposta

Nível de Escolaridade *

Sua resposta

Frequência de acesso a aplicações informáticas / websites *

- ◯ Diariamente
- ◯ 3-4 vezes por semana
- ◯ Semanalmente
- ◯ Mensalmente
- ◯ Outro

Utilidade

1- Ajuda-me a ser mais eficaz. *

	1	2	3	4	5	
Discordo Fortemente	◯	◯	◯	◯	◯	Concordo Plenamente

2 - Faz com que as tarefas que eu quero realizar sejam mais fáceis de serem feitas. *

	1	2	3	4	5	
Discordo Fortemente	◯	◯	◯	◯	◯	Concordo Plenamente

3 - Poupo tempo quando a uso. *

	1	2	3	4	5	
Discordo Fortemente	◯	◯	◯	◯	◯	Concordo Plenamente

4- O tempo de espera é aceitável. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

5 - Faz tudo o que eu esperaria que fizesse. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

Fácil de Usar

6 - É fácil de usar. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

7 - É simples de usar. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

8 - É user friendly. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

9 - Requer o menor número de passos possíveis para realizar o que eu pretendo fazer. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

10 - É flexível. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

11 - Eu posso usá-la sem instruções escritas. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

12 - Eu não notei nenhuma inconsistência enquanto a usei. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

13 - Eu consigo recuperar de erros com rapidez e facilidade. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

14 - Eu uso-a com sucesso todas as vezes. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

Satisfação

15 - Estou satisfeito(a) com a aplicação. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

16 - Funciona da maneira que eu quero que funcione. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

17 - É agradável de usar. *

	1	2	3	4	5	
Discordo Fortemente	○	○	○	○	○	Concordo Plenamente

18 - Gostaria de destacar aspetos positivos da aplicação?

Sua resposta

19 - Gostaria de destacar aspetos negativos da aplicação?

Sua resposta

ENVIAR

Nunca envie senhas pelo Formulários Google.