FATEC – FACULDADE DE TECNOLOGIA DE MOGI DAS CRUZES

MULTIKEYS - APLICAÇÃO PARA REMAPEAMENTO DE TECLAS EM *WINDOWS*

Mogi das Cruzes - SP

2017

FATEC – FACULDADE DE TECNOLOGIA DE MOGI DAS CRUZES

MULTIKEYS - APLICAÇÃO PARA REMAPEAMENTO DE TECLAS EM *WINDOWS*

Relatório Técnico Científico – Trabalho de Conclusão de Curso apresentado à FATEC Mogi das Cruzes como parte das exigências do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas para a obtenção do título de Tecnólogo.

Orientador: Prof. Dr. Bruno Marques Panccioni

Mogi das Cruzes - SP

2017

TAKAHASHI, Rafael Kenji. ***Multikeys*:** Aplicação para Remapeamento de Teclas em Windows. Mogi das Cruzes, 2017. 62 pág. Relatório Técnico Científico – Trabalho de Graduação, Faculdade de Tecnologia de Mogi das Cruzes.

RESUMO

Existem várias situações em que o usuário *desktop* precisa inserir caracteres especiais, o que é especialmente verdade para os usuários que frequentemente trabalham com símbolos menos comuns, como matemáticos e linguistas. Além disso, teclas que correspondem a atalhos de teclado ou a sequências de tecla são frequentemente usados em jogos e aplicações. Uma possível solução para estes casos seria utilizar um teclado adicional para estender as opções do usuário de acordo com suas necessidades, permitindo inclusive a entrada de texto em múltiplos idiomas ao mesmo tempo. No entanto, embora existam *softwares*, como editores de texto, que são capazes de inserir uma grande variedade de símbolos em *Unicode*, não existe uma solução simples para alterar o comportamento das teclas de teclados específicos conectados ao computador, impossibilitando o uso da solução proposta por parte do usuário. Este trabalho é uma pesquisa qualitativa e detalha o desenvolvimento de um *software* que utiliza a API *RawInput* juntamente com um *hook* de teclado para manipular a entrada de teclado do usuário a fim de mapear ações, incluindo a inserção de caracteres *Unicode*, associando-as a teclas de teclados específicos, e o desenvolvimento de uma interface gráfica para a configuração facilitada de um *layout* customizado. A abordagem iterativa foi escolhida para o desenvolvimento, empregando princípios da metodologia de desenvolvimento de software ágil, junto com alguns diagramas da UML. Devido às frequentes interações com a API do *Windows*, o módulo responsável por interceptar e alterar a entrada de usuário foi implementado na linguagem C++, enquanto foi utilizado C# com a tecnologia *WPF* para codificar a interface de usuário. Para a comunicação e persistência de dados, usou-se *XML*. O *software* desenvolvido neste trabalho tem várias aplicações em potencial, como a digitação em idiomas diferentes e a utilização de um teclado adicional para a inserção de atalhos de teclado.

Palavras-chave: teclado, remapeamento, *API Windows*, *RawInput, C++, C#, WPF, XML.*

TAKAHASHI, Rafael Kenji. ***Multikeys***: Application for Key Remapping in WIndows. Mogi das Cruzes, 2017. 62 pag. Conclusion Work Project, Faculdade de Tecnologia de Mogi das Cruzes.

ABSTRACT

There are many situations where a desktop user needs to insert special characters, which is especially true for user who frequently deal with less common symbols, such as mathematicians or linguists. Furthermore, keys that correspond to shortcuts and sequences of keys are frequently used in games and apps. A possible solution for these cases would be to use an additional keyboard to extend the user's options according to his or her needs, even allowing for input in multiple languages at once. However, even though there is software, such as text editors, that are capable of inserting a wide array of Unicode symbols, there does not exist a simple solution for changing the behavior of keys in specific keyboard connected to the computer, making it impossible for the user to implement the proposed solution. This work is a qualitative research, that details the development of a piece of software that uses the RawInput API and a keyboard hook to manipulate keyboard input in order to remap actions, including Unicode input, to keys in specific keyboards, and the development of a graphical user interface for ease of configuration of custom layouts. An iterative design was chosen for development, employing the principles of agile software development methodology, as well as some UML diagrams. Because of the frequent interations with the Windows API, the module responsible for intercepting and changing user input was implemented in the C++ language, while C# and WPF were used for coding the user interface. XML was used for communication and data persistence. The software developed in this work has many potential applications, such as typing in many languages, and using a second keyboard for shortcuts.

Keywords: keyboard, remapping, Windows API, RawInput, C++, C#, WPF, XML.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT – Associação Brasileira de Normas Técnicas;

API – *Application Programming Interface*, Interface de programação de aplicação;

ASCII – *American Standard Code for Information Exchange*;

BSD – Berkeley Licence Distribution, uma licença aberta de software

DLL – *Dynamic-Link Library*, biblioteca de link dinâmico;

GoF – *Gang of Four*, a Gangue dos Quatro, composta por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides.

IBM – *International Business Machines*, empresa de tecnologia;

IBM PC – IBM *Personal Computer;*

IBM XT – IBM *Extended Tecnology;*

IDE – *Integrated Development Environment*, ambiente de desenvolvimento integrado;

ISO – *International Organization for Standardization*, organização internacional de padronizações;

MS-DOS – *Microsoft Disk Operating System*, sistema operacional

MSDN – *Microsoft Developer Network*, website com a documentação oficial de várias tecnologias da Microsoft, incluindo .NET e a API do Windows;

.NET – *Framework* de software desenvolvido pela Microsoft para uso primariamente no sistema operacional Windows;

SGBD – Sistema gerenciador de banco de dados;

SGML – *Standard Generalized Markup Language*;

UML – *Unified Modeling Language*, linguagem de modelagem unificada;

UTF – *Unicode Transformation Format*, formato de transformação *Unicode*;

WinAPI – Abreviação de Windows API, a API do sistema operacional Windows;

WndProc – Nome abreviado do *Window Procedure*, procedimento de janela;

WPF – *Windows Presentation Foundation*, um susbsistema gráfico da Microsoft para a interface de usuário baseada em janelas;

XML – *Extensible Markup Language*, um subconjunto do SGML;

SUMÁRIO

INTRODUÇÃO

1.1 APRESENTAÇÃO DO PROBLEMA

Usuários de um computador frequentemente precisam inserir caracteres especiais dependendo da linguagem ou situação. Para a maioria dos usuários, os diferentes *layouts* de teclado oferecidos pelo sistema operacional são suficientes para atender estas necessidades.

No entanto, existem caracteres mais especializados como símbolos matemáticos ou letras de alfabetos fonéticos que não têm uma maneira fácil de serem inseridos. Além disso, devem ser considerados os atalhos de teclado para jogos e aplicações, e letras específicas de certos idiomas. Certos *softwares* permitem remapear teclas específicas a um comportamento customizado, mas não há solução que permita um comportamento diferenciado dependendo de qual teclado uma entrada originou.

Embora seja possível combinar mais de um *software* para atingir este comportamento, o processo requer conhecimento sobre linguagens de *scripts*, e a criação de *layouts* customizados é imprática para *layouts* com muitas teclas mapeadas.

Isto significa que os usuários que fazem uso frequente de caracteres especiais não têm uma maneira fácil de customizar um *layout* de acordo com suas necessidades, e não é possível mapear as teclas de um teclado adicional para expandir suas possibilidades de uso.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver e publicar uma ferramenta de *software* capaz de atribuir comportamentos customizados a teclas específicas, distinguindo entre entradas dos diferentes teclados conectados ao computador.

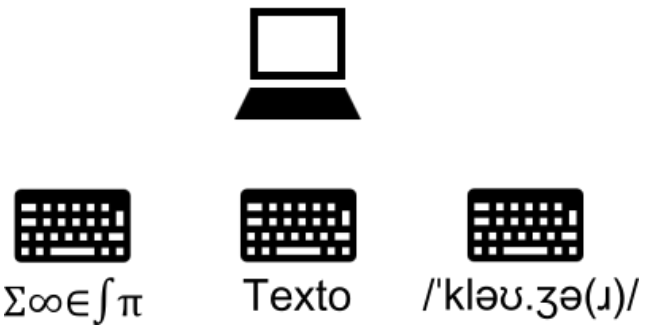
1.2.2 Objetivos Específicos

- Investigar e detalhar o funcionamento da interceptação de entrada de teclados no sistema operacional *Windows*;
- Desenvolver as funcionalidades da aplicação;
- Prover uma interface gráfica para a edição de *layouts*;

1.3 JUSTIFICATIVA

Remapear teclas é comum para usuários que desejam expandir a funcionalidade de seus teclados; por questão de conveniência e produtividade, um segundo dispositivo de entrada seria ideal para certos tipos de usuário, como aqueles de precisam regularmente inserir diversos caracteres matemáticos, ou que gostariam de realizar tarefas longas e/ou repetitivas (como acessar um *website* ou pressionar atalhos de teclado) com uma única tecla através de teclas de atalho. O uso de múltiplos teclados também permitiria a entrada de múltiplos idiomas simultaneamente.

Figura 1: Uso de Múltiplos Teclados



Fonte: Autoria Própria (2017)

Segundo Wells (2007), a entrada de caracteres especiais é uma questão não inteiramente resolvida atualmente. Existem *softwares* capazes de facilitar a entrada de caracteres *Unicode*, mas cada um possui suas próprias restrições.

Uma das grandes limitações a serem consideradas é que os sistemas operacionais da família *Windows NT (New Technology)* não distinguem entre dois ou mais teclados conectados a um computador, e, portanto, não permitem que *softwares*

respondam a diferentes teclados de maneira diferenciada. Ferramentas atualmente disponíveis para remapeamento de teclas (como *KeyTweak* e *AutoHotKey*) não oferecem a funcionalidade de distinguir entre teclados.

Além disso, aplicações com funcionalidades avançadas (como *AutoHotKey* e *LuaMacros*) requerem que o usuário empregue linguagens de *script* até mesmo para tarefas simples como remapear uma tecla, dificultando o uso e limitando o público-alvo.

O desenvolvimento deste *software* atende a estas deficiências, não só multiplicando o número de ações que o usuário pode realizar com seus dispositivos de entrada, mas também dando a ele uma fácil customização sobre a funcionalidade de tais dispositivos.

1.4 ESTRUTURAÇÃO DOS CAPÍTULOS

O tópico Introdução apresenta os objetivos e a justificativa deste trabalho, assim como o problema a ser resolvido.

O segundo tópico Metodologia apresenta a organização do desenvolvimento do projeto.

O terceiro tópico Fundamentação Teórica concentra sua atenção sobre os diferentes tópicos pesquisados para o desenvolvimento do projeto, assim como para a redação deste trabalho. Contém os seguintes subtópicos:

- Arquitetura de Aplicações em *Windows*: Descreve a arquitetura dos sistemas operacionais da família *Windows NT* e sua interação com os programas que executam em tal ambiente.

- Mensagens: Contém o funcionamento de mensagens e filas de mensagens, que é um tópico central na arquitetura de mensagens do sistema operacional *Windows*.

- Entrada de Teclado: Discute como ocorre a entrada de texto em *Windows*, dando uma base teórica para a interceptação de mensagens de tecla.

- Simulação de Entrada: Traz a ideia de simular mensagens como se houvessem sido gerados pelo usuário.

- *Software* Existente: Descreve brevemente os diferentes *softwares* disponíveis que oferecem funcionalidades similares, apresentando também suas limitações em relação aos requisitos deste projeto.

O quarto tópico Desenvolvimento documenta os problemas encontrados ao desenvolver o projeto, assim como estes problemas foram resolvidos. Neste tópico, são apresentadas e justificadas as escolhas arquiteturais do projeto. Contém os seguintes tópicos:

- Arquitetura: Apresenta a arquitetura do projeto, e justifica as decisões arquiteturais.

- Tecnologias Utilizadas: Descreve brevemente as ferramentas, padrões e linguagens utilizadas para o desenvolvimento deste projeto, e justifica a escolha destas tecnologias.

- Manipulação de Mensagens: Relata os métodos utilizados para realizar a manipulação de mensagens pelo programa;

- Obstáculos: Relata as dificuldades relevantes encontradas durante o desenvolvimento, assim como o processo de resolução para cada problema.

- Simulação de Entrada: Apresenta como foi implementada a funcionalidade de simular ações do usuário.

- Edição de *Layout*: Discute a camada visual do programa, seus componentes, problemas e soluções.

- Limitações: Apresenta pontos que não são resolvidos pelo sistema *Multikeys*, por impossibilidade técnica ou por questão de escopo.

- Publicação: Brevemente apresenta como foi realizada a publicação do sistema *Multikeys*.

As Considerações Finais fecham com um retrospecto deste trabalho, discutindo o que foi aprendido, se os objetivos foram atingidos, e outros tópicos.

2 METODOLOGIA

Este trabalho possui uma natureza tecnológica, visto que resultará no desenvolvimento de uma aplicação de *desktop*. Sua realização foi dividida nas seguintes etapas:

- Uma pesquisa bibliográfica sobre a detecção e interceptação de entrada de teclado, assim como a simulação de entrada usando a API (*Application Programming Interface*) do *Windows*. Nesta etapa também foi pesquisada a representação e manipulação de texto em *Unicode*, e foi investigado o comportamento das mensagens de teclado e dos mecanismos de interceptação de mensagens em *Windows*.

- A codificação das funcionalidades do programa como uma aplicação em *Visual C*++ para *Windows*, utilizando o que foi aprendido ao estudar a API do Windows.

- O desenvolvimento de uma interface gráfica em C# para uma interação fácil e intuitiva com o programa. A arquitetura do programa é tal que as funcionalidades principais não dependam da interface gráfica.

Em relação ao desenvolvimento do projeto, o sistema possuirá uma arquitetura modularizada; isto é, vários de seus componentes são bibliotecas e executáveis separados que podem possuir seus próprios requisitos. Assim, uma abordagem ágil e iterativa foi escolhida por ser adequada ao desenvolvimento deste *software*; para tal, as seguintes etapas foram identificadas:

- Análise de requisitos: No início de cada iteração, os requisitos do sistema foram revisados e atualizados. Os requisitos foram baseados no comportamento de um *layout* de teclado em *Windows* (incluindo teclas modificadoras, teclas de acentuação e outros casos), assim como nas questões de usabilidade de múltiplos teclados.

- Modelagem: Planejamento da arquitetura (módulos em C++, C# e sua comunicação, persistência de dados em XML e interface em WPF), modelagem com diagramas em UML usando o *StarUML* e *Microsoft Visio*, persistência de dados em arquivos XML com estrutura definida em XSD (esquemas XML) ao invés do uso de banco de dados, e interfaces entre os elementos do sistema através de mensagens.

- Geração de código: Implementação do sistema de acordo com a descrição previamente estabelecida; usando as linguagens C++ e C# no *IDE Visual Studio Community*.

- Testes: Os componentes implementados passam por testes para determinar a conformidade com os requisitos, em particular o comportamento de *layouts* de teclado em *Windows*, incluindo teclas de acentuação, teclas modificadoras e vários detalhes e casos específicos. Foram realizados testes unitários usando as ferramentas padrões de testes em C# do *Visual Studio*; no entanto, devido a dificuldades de implementação, os testes do núcleo da aplicação (interceptação de mensagens) possuem execução manual.

Estas etapas são repetidas iterativamente e continuamente. As iterações principais que foram identificadas são:

- Iteração 1: Interceptação e interpretação de teclas; resulta em um sistema capaz de bloquear e extrair informações sobre as teclas pressionadas em diferentes teclados conectados ao computador.

- Iteração 2: Modelo interno de teclados e *layouts*; resulta em um sistema capaz de responder a certas teclas em determinados teclados conectados ao computador por meio de ações pré-determinadas, como enviar um caractere *Unicode* a uma janela, ou simular a pressão de várias teclas em sequência.

- Iteração 3: Leitura de *layouts* customizados; resulta em um sistema capaz de ler configurações armazenadas em um arquivo para determinar quais substituições serão realizadas em tempo de execução.

- Iteração 4: Interface gráfica; resulta em um ambiente gráfico voltado à criação e edição de *layout*s customizados.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 ARQUITETURA DE APLICAÇÕES EM WINDOWS

Segundo a MSDN (2017) (*Microsoft Developer Network*), “Diferente de aplicações baseadas em MS-DOS(*Microsoft Disk Operating System*), aplicações em *Windows* são orientadas a eventos”. Ao invés de fazer chamadas a funções para obter entrada, aplicações esperam mensagens adereçadas a programas específicos.

Um sistema dirigido por eventos realiza ações de acordo com atividades externas, ao invés de receber chamadas e devolver uma resposta. Estes eventos são transmitidos em *broadcast* para quaisquer partes interessadas sem uma ordem previamente conhecida e sem a espera de resposta, ou seja, de maneira assíncrona. Outras características da orientação a eventos incluem a ausência de uma *stack call* em favor de um canal de eventos, e uma interação mais distante entre componentes, que não conhecem, por exemplo, nomes de métodos uns dos outros (HOHPE, 2006).

Eventos correspondem a mudanças de estado que influenciam o resultado final, sendo representados por objetos de eventos. Levina e Stantchev (2009) definem os seguintes termos na arquitetura orientada a eventos:

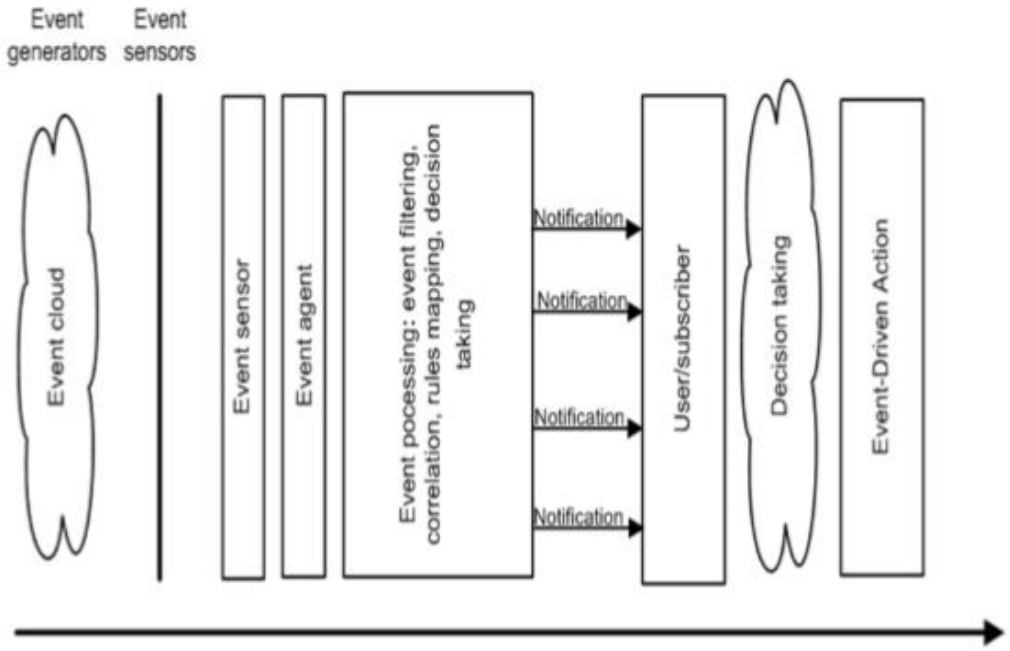
Fonte de eventos: Uma mudança de estado abstrata, interna ou externa à aplicação, que causa a criação de um objeto de evento;

Consumidor de eventos: Cada interessado que recebe eventos para realizar algum processamento ou para redirecionar o evento.

Agente de eventos: A parte responsável por capturar e enviar eventos.

Os termos citados estão sumarizados na Figura 2.

Figura 2: Arquitetura movida por eventos



Fonte: LEVINA e STANTCHEV (2009)

O termo “evento” é usado para significar mais de um conceito em computação, o que pode causar confusão (FOWLER, 2017). Neste contexto, o termo é usado para representar a notificação de eventos, que ocorre quando mensagens são enviadas para notificar partes do sistema de ocorrências relevantes no domínio.

3.1.1 Sistemas Reativos

Segundo Aceto et al. (2007), em contraste com a visão clássica de um programa de computador como uma caixa-preta que recebe uma entrada e produz uma saída, um sistema reativo continuamente responde aos estímulos do ambiente, sem mesmo a necessidade de chegar a um estado final em algum ponto.

Deve-se notar que há diferença entre programação reativa, focada em fluxo de dados e orientada a eventos, e sistemas reativos, focados na resiliência e orientados a mensagens. Mensagens são diferentes de eventos por possuírem um destinatário específico, e, portanto a arquitetura orientada a mensagens coloca mais ênfase nos componentes e a comunicação entre eles do que nos fatos que os causam (BONÉR e KLANG, 2016).

Fowler (2017) utiliza o termo mensagem de evento para descrever a comunicação entre componentes que informa uma mudança de estado no domínio. Ainda segundo Fowler (2006a) embora haja similaridades entre eventos e commands, há a diferença de que eventos informam sobre uma mudança de estado sem se preocupar com o que o destinatário faz com o objeto recebido, em contraste com um command, que encapsula uma tarefa a ser realizada.

Uma das qualidades desejáveis de um sistema reativo, orientado a mensagens, é a resiliência: a capacidade do sistema permanecer responsivo mesmo após sofrer uma falha, recuperando-se dela sem a propagar para outros componentes. O uso de mensagens permite um acoplamento mais fraco entre componentes e evita que o cliente seja responsável por se recuperar das falhas ocorridas em outros pontos no sistema (BONÉR e KLANG, 2016).

No entanto, uma consequência negativa do fraco acoplamento desta arquitetura é a dificuldade de se rastrear o fluxo de eventos devido às colaborações implícitas; uma questão particularmente problemática é a cascata de eventos, em que existe uma árvore composta de eventos, cada um causando o próximo, que se torna extremamente difícil de depurar conforme a complexidade do código aumenta. (FOWLER, 2006b).

3.1.2 Sistemas de Tempo Real

Em sistemas de tempo real, o comportamento correto é determinado não somente pelos resultados produzidos, mas também pelo aspecto temporal da resposta; isto é, o sistema deve realizar o processamento dentro do prazo especificado (LIPARI e PALOPOLI, 2015).

Sistemas de tempo real são um tipo de sistema reativo em que o processamento, além de ocorrer em resposta a estímulos de entrada vindos do ambiente, está sujeito a prazos determinados para ser realizado (FARINES, FRAGA e OLIVEIRA, 2000).

Stankovic (1992) nota a diferença entre sistemas em que a falha de realizar processamento dentro de um prazo causa a falha do sistema (*hard real-time*), e sistemas em que processamento atrasado meramente degrada o valor da tarefa realizada (*soft real-time*); esta diferença é o *rigor* do prazo. Segundo Juvka (1998), tarefas em *hard real-time* devem ser previsíveis e planejadas com cuidado, pois são

tipicamente tarefas críticas. Numerosos algoritmos de agendamento existem para garantir que tarefas possuam recursos necessários (incluindo tempo) para serem realizadas conforme os prazos.

3.2 MENSAGENS

Como cada processo em *Windows* possui seu próprio espaço de memória, ponteiros só podem referenciar endereços dentro do mesmo processo; isto contribui para a robustez e segurança do sistema, mas requer mecanismos adicionais para permitir a comunicação entre processos (RICHTER e NASARRE, 2008).

Esta comunicação ocorre através de mensagens, que são um conceito central na arquitetura dos sistemas operacionais da família *Windows NT*. Estruturalmente, o sistema utiliza orientação a objetos para organizar janelas (incluindo janelas de controle como botões), mas a comunicação entre componentes é dirigida por eventos (PETZOLD, 1998).

Segundo a documentação da API do Windows na MSDN (*MICROSOFT*, 2017), mensagens são definidas no cabeçalho winuser.h como:

```
typedef struct tagMSG
{
    HWND hwnd;
    UINT message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
} MSG, *PMSG;
```

No qual *hwnd* é um *handle* (uma referência opaca) para a janela adereçada, *message* contém o número que identifica o tipo de mensagem (que pode ser um valor definido pelo sistema ou pelo usuário), e o conteúdo de *wParam* e *lParam* são de uso específico da mensagem. Além disso, *time* guarda o momento no qual a mensagem foi colocada na fila e *pt* armazena a posição do cursor quando a mensagem foi gerada.

Segundo Petzold (1998), o sistema operacional é responsável por enviar mensagens às janelas, e mantém uma fila de mensagens específica para cada programa, contendo as mensagens adereçadas ao programa em questão. A janela, por sua vez, aceita estas mensagens através de um loop em seu procedimento de janela. O procedimento é sempre definido da seguinte forma:

```
LRESULT CALLBACK WndProc
        (HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
```

De acordo com a MSDN (*MICROSOFT*, 2017), aplicações podem definir suas próprias mensagens e enviá-las a outros processos usando a função *SendMessage*, mas grande parte das mensagens é gerada pelo sistema operacional, e incluem comandos para manipular a janela, finalizar a aplicação, e outros; adicionalmente, cada entrada do usuário causa uma mensagem do sistema. Mensagens são inicialmente colocadas em uma fila de mensagens do sistema, e o sistema operacional as direciona, uma a uma, para a fila de mensagens da *thread* apropriada.

3.3 ENTRADA DE TECLADO

3.3.1 Scancodes

Cada tecla física no teclado possui um valor chamado *scancode*, consistindo em um ou mais bytes que são enviados ao computador quando uma tecla é pressionada (KAPLAN e WISSINK, 2003).

Quando o usuário pressiona uma tecla, o *Windows* gera um *interrupt* que é tratado pelo *driver* de teclado. O driver lê o *scancode* gerado pelo *hardware*, gerando uma mensagem para a janela em foco contendo os dados da tecla pressionada (HOSKEN, 2003).

O *scancode* é chamado de *make* quando representa uma tecla sendo pressionada, e *break* quando representa uma tecla sendo solta. Existem principalmente três tabelas de *scancodes*: o *scancode set 1* é o originalmente usado no IBM (*International Business Machines*) PC (*Personal Computer*) (e também no IBM XT (*Extended Technology*)); o *scancode set 2* foi introduzido pelo IBM AT; internamente, um computador moderno traduz os sinais vindos de um teclado para os *scancodes* do *set* 1. Adicionalmente, o *scancode set* 3 foi introduzido pelo IBM PS/2, mas raramente usado (CHAPWESKE, 2013).

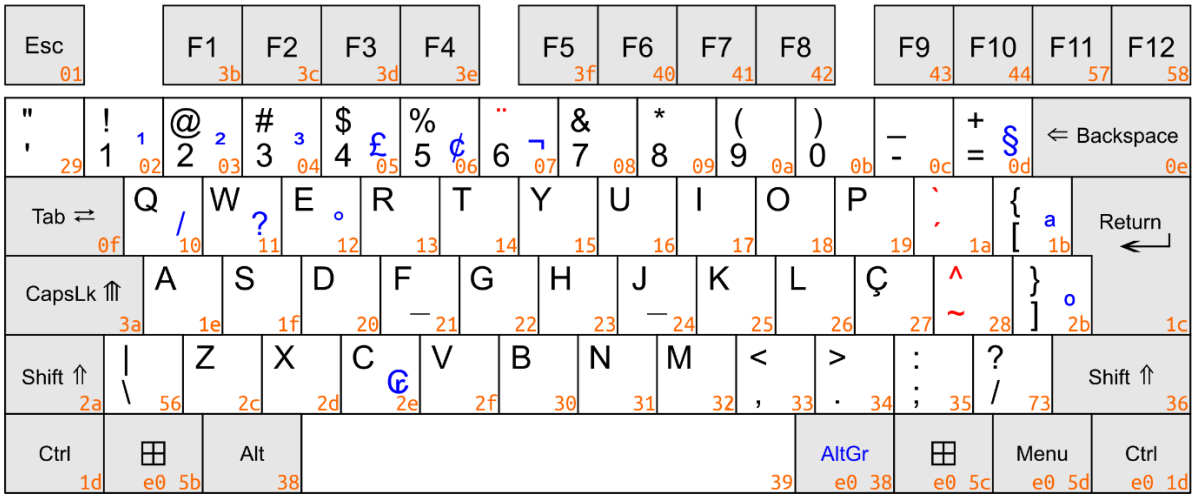
A tradução entre os *sets* 1 e 2 se dá porque o *set* 2 (IBM AT) era incompatível com o *set* 1 (IBM XT), motivando o uso do microprocessador 8042, responsável por

traduzir *scancodes* do *set* 2 para o *set* 1. Esta tradução se tornou uma prática padrão em computadores modernos (BROUWER, 2009).

Cada código corresponde a uma tecla física no teclado, e não necessariamente representa o caractere impresso na tecla. (CHAPWESKE, 2013). Como *scancodes* são associados a teclas físicas, seus valores não mudam dependendo da linguagem do sistema; ainda assim, existem diferentes *layout* físicos no mercado, causando uma certa inconsistência no posicionamento de teclas (KAPLAN e WISSINK, 2003).

A Figura 3 ilustra os *scancodes*, em hexadecimal, associados a cada tecla de um teclado em conformidade com a norma ABNT variante 2.

Figura 3: *Scancodes* do *layout* ABNT-2



Fonte: Adaptado de BROUWER (2009).

A pressão de teclas em *Windows* causa mensagens *WM_KEYDOWN* e *WM_KEYUP* a serem adicionadas à lista de mensagens do sistema, e estas são depois direcionadas à lista de mensagens da *thread* apropriada (geralmente a janela que possui foco). O sistema só direciona mensagens de teclado à *thread* correta depois que esta termina de processar a mensagem anterior (PETZOLD, 1998).

3.3.2 *Layout* de Teclado

O *layout* de um teclado é a coleção de dados correspondentes às combinações de estado e pressões de tecla para algum determinado *driver* de dispositivo (KAPLAN e WISSINK, 2003).

3.3.2.1 Teclas Modificadoras

O uso de teclas modificadoras, como *Shift*, AltGr, *Option* e outros é a maneira mais comum para se atribuir caracteres adicionais às teclas disponíveis em um teclado (HOSKEN, 2003).

3.3.2.2 Teclas de Acentuação

Teclas de acentuação são comuns em vários *layout*s de teclado, e consistem em teclas que não enviam um caractere e não causam feedback visual, mas modificam o próximo caractere a ser inserido (HOSKEN, 2003).

3.3.2.3 Teclas Operadoras

Outra opção são teclas operadoras, que funcionam de maneira similar às teclas de acentuação, mas são inseridas *após* o caractere a ser modificado. O obstáculo maior é na implementação deste tipo de tecla, pois deve haver alguma camada de software que consegue alterar um caractere já inserido (HOSKEN, 2003).

3.3.3 Teclas Virtuais

Diferente dos *scancodes* produzidos pelas teclas físicas, teclas virtuais são independentes da posição da tecla no teclado, e são traduzidas de *scancode*s de acordo com um determinado *layout* (KAPLAN e WISSINK, 2003). É possível ter mapeamentos completamente diferentes entre *scancodes* e teclas virtuais dependendo do *layout* ativo (HOSKEN, 2003).

De acordo com a documentação da API do *Windows* na MSDN (*MICROSOFT*, 2017), teclas virtuais são independentes de *layout* físico e podem corresponder a caracteres diferentes dependendo da linguagem; por exemplo, a tecla virtual 0xba é traduzida do *scancode* 0x27 no *layout* americano e produz um ponto-e-vírgula, enquanto a mesma tecla virtual (0xba) é traduzida do *scancode* 0x1a no *layout* alemão e produz a letra “u com *Umlaut* (‘ü’) ”.

Teclas virtuais possuem o comprimento fixo de um *byte*, e podem representar diversos comandos além do teclado convencional, como teclas de mídia, cliques de mouse e comandos específicos a certas linguagens (como 0x15: "*VK_HANGUL*", presente no teclado coreano).

3.3.4 *Hook*s de Teclado para Manipulação de Mensagens

Além de receber mensagens pelo procedimento de janela, uma aplicação pode interceptar mensagens que não são adereçadas a ela, através de um *Hook*.

A documentação da API do Windows na MSDN define um *hook* como "um mecanismo pelo qual uma aplicação pode interceptar eventos [...]. Uma função que intercepta um tipo de evento em particular é chamada de *hook procedure*" (*MICROSOFT*, 2017).

Segundo Richter e Nasarre (2008), *hooking* é uma maneira de injetar uma DLL (*Dynamically Linked Library*) no espaço de execução de outro processo; através disto, é possível que uma aplicação execute seu próprio código no lugar de outro processo.

Ao interceptar uma mensagem de teclado, uma aplicação tem acesso a uma mensagem contendo *scancode*, código de tecla virtual, e *flags* com informações adicionais. No entanto, *hooks* de teclado não têm acesso ao nome do dispositivo que gerou a mensagem.

Para obter o nome do teclado que enviou uma mensagem de tecla, é necessário usar outro recurso do *Windows*, chamado *Raw Input* API. Sua documentação detalha que através desta interface é possível interceptar uma mensagem em um momento mais anterior do que um *hook* de teclado, e extrair o nome do dispositivo que a gerou. Esta API não oferece, porém, a possibilidade de alterar o conteúdo das mensagens (*MICROSOFT*, 2017).

MEDEK (2014) e BLECHA (2014) demonstram que é possível utilizar o *RawInput* API para decidir se uma determinada entrada deve ser bloqueada, juntamente com um *hook* de teclado para aplicar tal decisão.

O *hook* de teclado intercepta mensagem primariamente de acordo com mensagens de tecla virtual, enquanto o *Raw Input* utiliza informações de *scancode*. Portanto, combinar estes dois mecanismos requer cuidado em situações em que *scancodes* e teclas virtuais não têm uma correspondência um-a-um, que levam os dois mecanismos de interceptação a identificarem mensagens conflitantes.

Um exemplo de mensagens conflitantes é a tecla Alt direita, que produz dois eventos de teclas virtuais (Ctrl e *Left* Alt) traduzidos de um só *scancode* em *layouts* que contém a tecla AltGr, gerando dois eventos para o *RawInput* enquanto o *Hook* de teclado identifica uma mensagem apenas.

3.4 SIMULAÇÃO DE ENTRADA

A simulação de entrada de texto deve ser feita em *Unicode*, uma vez que este padrão é amplamente adotado e possibilita a entrada de numerosos caracteres especiais (UNICODE CONSORTIUM, 2016). Esta seção detalha o padrão *Unicode*, e a simulação de entrada de texto em Windows.

3.4.1 Unicode

O *Unicode* é o padrão universal para codificar caracteres internacionalmente, definindo um valor único para cada caractere em numerosos idiomas e para diversos propósitos (UNICODE CONSORTIUM, 2016).

Segundo John Wells (2007), o *Unicode* é um padrão para a codificação de todas as linguagens escritas do mundo, assim como para caracteres de disciplinas especializadas. Por ser *multi-byte*, o padrão permite a codificação de um grande número de caracteres.

Diferente de outros repertórios de caracteres, como o ASCII, o padrão *Unicode* define um conjunto universal de caracteres para uso internacional, a fim de atender às necessidades causadas pelos numerosos padrões incompatíveis então existentes, sobretudo quanto a textos multilíngues. O padrão foi criado com o objetivo de fornecer uma maneira simples de representar todos os sistemas de escrita em caracteres de 16-bits (depois estendido para 32-bits) (CONSTABLE, 2001).

Cada caractere em *Unicode* é representado por um número inteiro positivo chamado de ponto de código (*codepoint*), que é convencionalmente representado por “U+”, seguido por no mínimo quatro dígitos hexadecimais. Atualmente, o padrão ocupa toda a extensão U+0000..U+10FFFF dividido em onze planos de 65536 ($=2^{16}$) pontos de código cada. Isto significa que uma variável de 32-bits é suficiente para representar qualquer caractere *Unicode*.

No entanto, é incomum representar cada caractere usando 32 bits, uma vez que as letras mais comumente usadas precisam de menos espaço para serem representadas. Todos os caracteres do ASCII, por exemplo, ocupam os pontos de código U+0000..U+007F. Maneiras de se representar qualquer ponto de código em uma ou mais variáveis de 8 bits e 16 bits são chamados de UTF-8 e UTF-16, respectivamente, e ambos são definidos dentro do padrão *Unicode* (neste caso, cada caractere possui um comprimento variável). Adicionalmente, a representação de pontos de código usando variáveis de 32 bits é chamada de UTF-32 (UNICODE CONSORTIUM, 2016).

A família *Windows NT* suporta caracteres *Unicode* usando caracteres de 16 bits de comprimento, e define o tipo WCHAR no cabeçalho winnt.h como um *typedef* de wchar_t, que por sua vez é definido no padrão C como uma variável de 16 bits ao invés dos 8 bits de um char (PETZOLD, 1998).

3.4.2 UTF-16

Quando ficou claro que valores de 16 bits não seriam suficientes para cobrir todos os sistemas de escrita necessários, foi necessário substituí-los por valores de 32 bits. Esta mudança foi formalizada no *Unicode* 2.0 em 1996, que também define formas de representar qualquer ponto de código de 32 bits de maneira não ambígua usando caracteres de 16 bits ou 8 bits de comprimento (CONSTABLE, 2001).

O *Unicode Transformation Format* (UTF) ficou definido então como uma codificação do Unicode que atribui uma sequência única reversível de bytes para cada ponto de código. Os possíveis formatos são UTF-32, UTF-16 e UTF-8, que representam cada ponto de código usando um ou mais caracteres de 32, 16 e 8 bits de comprimento, respectivamente (UNICODE CONSORTIUM, 2016).

Segundo o padrão, em um texto *Unicode* que usa o formato UTF-16, cada caractere é representado por um ou dois valores de 16 bits cada, chamados de unidades de código (*code units*). Pontos de código iguais ou menores a U+FFFF são representados por um valor de 16-bits numericamente igual ao ponto de código, enquanto os pontos de código acima de U+FFFF são representados por um par de unidades de código, chamado de par substituto (*surrogate pair*), calculado da seguinte maneira (*lead* = primeiro valor, *trail*= segundo valor):

```
// constants
const UTF32 LEAD_OFFSET = 0xD800 - (0x10000 >> 10);
const UTF32 SURROGATE_OFFSET = 0x10000 - (0xD800 << 10) - 0xDC00;

// computations
UTF16 lead = LEAD_OFFSET + (codepoint >> 10);
UTF16 trail = 0xDC00 + (codepoint & 0x3FF);

UTF32 codepoint = (lead << 10) + trail + SURROGATE_OFFSET;
```

Os dois valores produzidos pelo algoritmo são chamados de substituto posterior (*high surrogate*) e um substituto anterior (*low surrogate*). Nota-se que o algoritmo sempre produzirá um substituto posterior dentro da faixa U+D800..U+DBFF, e um substituto anterior dentro da faixa U+DC00..U+DFFF. Os pontos de código dentro destas faixas são reservados pelo padrão *Unicode* para uso como pares substitutos (não são designados caracteres), e sua ocorrência isolada deve ser considerada um erro de codificação (CONSTABLE, 2001).

Os sistemas operacionais da família *Windows NT* trabalham internamente com caracteres de 16 bits de comprimento, e suportam tanto ASCII quanto *Unicode* (PETZOLD, 1998). A documentação da API do *Windows* detalha que as funções que trabalham com *strings* de caracteres de 16 bits esperam *strings* formatadas em UTF-16.

3.4.3 SendInput

A documentação da API do *Windows* descreve a função *SendInput*, que permite a simulação de entrada de usuário na forma de teclas virtuais ou caracteres *Unicode*. Além disso, o tipo wchar_t definido no padrão C++ para suporte a *Unicode* é implementado como uma variável de 16-bits em *Visual C++* (*MICROSOFT*, 2017).

SendInput torna possível simular a entrada de qualquer sequência de caracteres *Unicode* ou teclas virtuais, para a simulação de tarefas como pressionar um atalho no teclado.

3.5 *SOFTWARE* EXISTENTE

Esta seção descreve os *softwares* e ferramentas já existentes que realizam tarefas similares ao software proposto neste trabalho. A seção também aponta as limitações destes *softwares*.

3.5.1 Tavultesoft Keyman

Keyman é um *software* multiplataforma desenvolvido pela *Tavultesoft* para a criação e uso de *layouts* customizados, com o intuito de permitir ao usuário usar seu teclado para escrever em numerosos sistemas de escrita (RAYMOND, 2014).

O *software* tem foco em linguagens, e oferece uma interface intuitiva e responsiva, incluindo teclas modificadoras. Ele não oferece, porém, a possibilidade de utilizar mais de um *layout* em teclados diferentes, e o fato de ser proprietário pode causar possíveis preocupações com *keyloggers* e *softwares* similares por parte do usuário.

3.5.2 AutoHotKey

AutoHotKey é uma linguagem de script para *Windows* XP ou mais recente que permite ao usuário atribuir a execução de scripts a teclas ou sequências de teclas. Scripts escritos nesta linguagem são capazes de interagir com a API do *Windows* para realizar numerosas tarefas ativadas por sequências de teclas pré-determinadas (GITHUB, 2017).

Embora o *AutoHotKey* possa ser usado para criar um *layout* customizado através de *scripts* que enviam caracteres *Unicode* para a janela em foco, a linguagem não possui a capacidade de distinguir entre dispositivos diferentes. Além disso, qualquer usuário que deseja criar seus próprios mapeamentos de teclado precisa aprender a linguagem, dificultando a usabilidade.

3.5.3 LuaMacros

LuaMacros, antigamente chamado de *HidMacros*, é uma aplicação *Windows* para a execução de macros ativados de diferentes dispositivos USB (*Universal Serial Bus*). O programa é capaz de abrir executáveis e enviar sinais de tecla simulados, assim como atribuir estas ações a teclados específicos, e foi desenvolvido tendo em mente os simuladores de aviação, para permitir o uso de múltiplos teclados como controle (MEDEK, 2014).

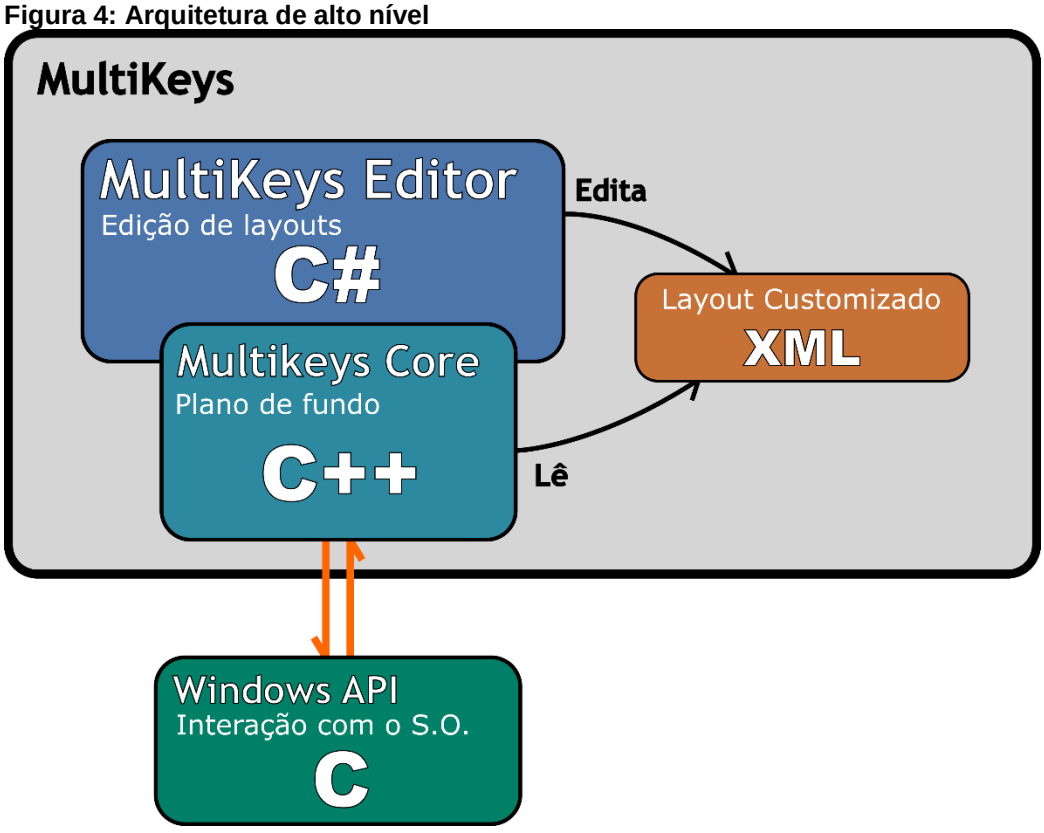
No entanto, *LuaMacros* não suporta a simulação de entrada de caracteres *Unicode*, e também não é feito para simular as características de um *layout* de teclado, como teclas modificadoras. Isto faz com que o programa seja inadequado para simular a funcionalidade de um *layout* de teclado, que entre outras coisas deve conter estados internos.

4 DESENVOLVIMENTO

4.1 ARQUITETURA

A Figura 4 ilustra a arquitetura de alto nível do sistema *Multikeys*. *Multikeys Editor* e *Multikeys Core* são os principais módulos do programa, que possuem diferentes funcionalidades e requisitos. Conforme ilustrado, o módulo *Multikeys Editor* depende do *Multikeys Core*, mas não o oposto.

Figura 4: Arquitetura de alto nível

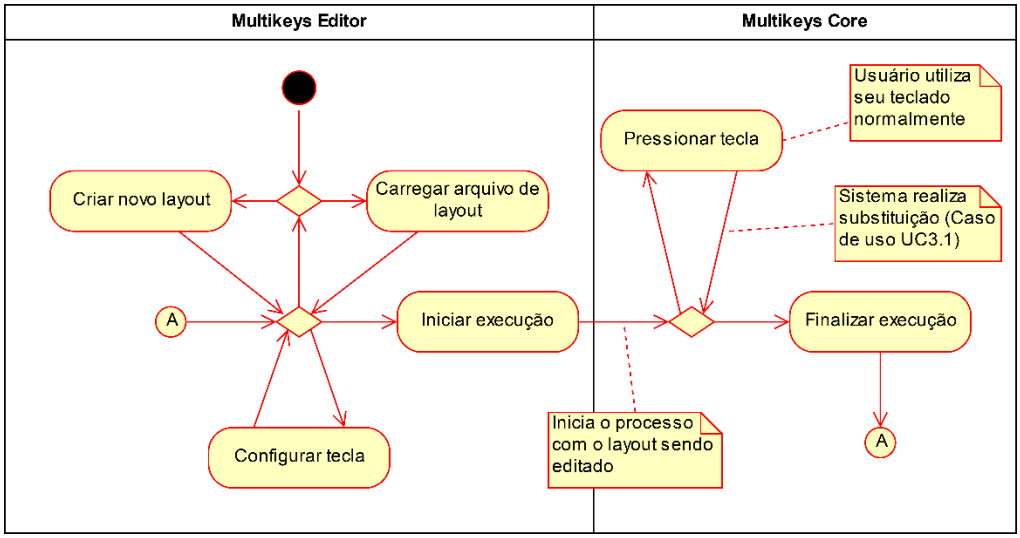


Fonte: Autoria Própria (2017)

Através do módulo *Multikeys* Editor, o usuário cria e manipula layouts, que são usados pelo módulo *Multikeys* Core para realizar os remapeamentos de tecla.

A Figura 5 contém um diagrama de atividades mostrando as principais ações que o usuário realiza durante a utilização do sistema.

Figura 5: Diagrama de atividades do sistema



Fonte: Autoria Própria (2017)

4.1.1 *Multikeys* Editor

Este subsistema é uma aplicação gráfica voltada para a configuração dos *layouts* customizados; através de janelas gráficas, o usuário pode abrir, visualizar e editar *layouts*, assim como associar *layouts* com os teclados conectados ao computador.

O subsistema *Multikeys Editor* abrigará todas as funcionalidades relacionadas à configuração do sistema, incluindo a linguagem de exibição. O objetivo é oferecer ao usuário uma experiência intuitiva, que não requeira conhecimento aprofundado sobre o funcionamento de *layouts* de teclado, e nem o aprendizado da lógica por trás dos *layouts*.

WPF (*Windows Presentation Foundation*) foi escolhido como a tecnologia para a criação do ambiente gráfico pela relativa facilidade de criar formas customizadas. Dado isso, este subsistema é escrito na linguagem C# e utiliza a plataforma .NET.

4.1.2 Multikeys Core

Este subsistema é uma aplicação escrita em C++ para *Windows*, e consiste em um processo em plano de fundo que realizará as funcionalidades chave do sistema, como interceptar mensagens de teclado e identificar sua origem.

Um processo em *Windows* precisa de uma janela controlada por um *Window Procedure* para que possa receber mensagens do sistema, o que é fundamental para o funcionamento do programa. Porém, a janela não possuirá um elemento gráfico visível, e como consequência não ficará visível na barra de tarefas. O processo se caracteriza como um processo em plano de fundo porque o usuário não tem acesso direto a este.

Este processo será inicializado e terminado de acordo com as decisões do usuário, através da interface gráfica. O subsistema *Multikeys Editor* tem controle sobre as instâncias do subsistema *Multikeys* Core.

4.1.3 Persistência de *Layout*s

A persistência de dados do sistema *Multikeys* não utilizará bancos de dados, uma vez que não será necessário realizar o armazenamento de grandes volumes de dados, nem grandes quantidades de consultas a dados. Além disso, a possibilidade do usuário manipular um *layout* customizado na forma de um arquivo salvo em disco é uma facilidade de uso, e não requer a instalação prévia de um Sistema Gerenciador de Banco de Dados (SGBD).

Um arquivo de configuração em XML (*Extended Markup Language*) atende às necessidades do sistema, por ser próprio para armazenar estruturas hierárquicas de dados com uma estrutura bem definida. O subsistema *Multikeys Editor* acessará os *layouts* customizados na forma de arquivos de configuração XML (não necessariamente usando a extensão XML, devido à convenção em *Windows* de se usar extensões que identifiquem o programa que abre o tipo de arquivo); o subsistema *Multikeys Core*, por outro lado, abre os arquivos de configuração para leitura e carrega o *layout* em memória.

4.1.4 *Windows API*

A API do *Windows* será usada para ter acesso a várias operações de baixo nível do sistema operacional. A documentação oficial desta API é publicada pela Microsoft no website *Microsoft Developer Network* (Rede de Desenvolvedores da Microsoft), MSDN.

As funções e estruturas de dados da API do *Windows* são escritas na linguagem C, possibilitando que um programa escrito em C++ as utilize sem configurações adicionais.

4.2 TECNOLOGIAS UTILIZADAS

Esta seção descreve as tecnologias utilizadas para o desenvolvimento do projeto.

4.2.1 Visual Studio

O *Visual Studio* é uma IDE (*Integrated Development Environment*) desenvolvida e mantida pela *Microsoft*, que suporta o desenvolvimento para uma variedade de plataformas, incluindo o .NET. Como o sistema faz bastante uso de tecnologias da *Microsoft*, em particular as janelas WPF da plataforma .NET, o *Visual Studio* foi escolhido como a ferramenta de desenvolvimento do projeto.

Além disso, o *Visual Studio* tem um editor de XML integrado, incluindo um editor de esquemas XML para a edição e visualização de arquivos XSD (*XML Schema Definition*). Esta ferramenta foi utilizada para a criação da documentação referente aos arquivos de configuração.

4.2.2 WPF

Windows Presentation Foundation, ou WPF, é uma plataforma gráfica do *Windows*, parte da plataforma .NET, para o desenvolvimento de aplicações em janela. Na versão 4.5 da plataforma .NET, aplicações WPF podem ser desenvolvidas para *Windows* Vista ou superior. WPF oferece uma API de maior nível em comparação a outras tecnologias similares, com funcionalidades como independência de resolução (MACDONALD, 2012, pag. 3-6).

Esta tecnologia atende às necessidades do sistema *Multikeys* por ser voltada a interfaces em *desktop*. Embora seja possível programar a interface em C++, o uso do WPF implica em muitas vantagens, tanto quanto à facilidade de programação quanto às funcionalidades gráficas.

4.2.3 Linguagens

4.2.3.1 C/C++

Segundo Petzold (1998, p. 7), o uso da linguagem C para interagir com a API do *Windows* oferece a melhor performance e versatilidade em relação às operações a serem realizadas pelo sistema operacional. Uma alternativa comum é utilizar a linguagem C++ para escrever uma aplicação que precise interagir com a API do *Windows*.

A linguagem C++ é uma extensão de C que possui características da orientação a objetos, como herança e polimorfismo (SAHAY, 2012, p.7-8). A linguagem C++ foi escolhida para o subsistema *Multikeys Core* devido aos requisitos de performance e também à interação com a API do Windows, ao mesmo tempo que permite o emprego dos princípios de orientação a objetos.

4.2.3.2 C#

C# é uma linguagem orientada a objetos desenvolvida pela *Microsoft* que foi inspirada por diversas outras, como C++ e *Java*; a linguagem é integrada na plataforma .NET, com acesso a uma vasta quantidade de recursos e interoperabilidade com outras linguagens da plataforma (PALMER e BARKER, 2008, p. 4-5).

A linguagem foi escolhida para a interface gráfica do sistema *Multikeys* por fazer parte da plataforma .NET, e também devido ao uso da tecnologia WPF (*Windows Presentation Foundation)* para a criação de janelas em *desktop*.

4.2.3.3 XML / XSD

XML (*Extensible Markup Language*) é uma linguagem de *markup* (marcação) usada neste projeto para a persistência de dados.

XML é um subconjunto do SGML (*Standard Generalized Markup Language*) usado para armazenamento de dados e metadados, especialmente em estrutura hierárquica, como por exemplo, um arquivo de sistema. Seu uso possibilita a interoperabilidade entre programas que não precisam conhecer dados um sobre o outro, funcionando como um protocolo de dados e metadados (FAWCETT, QUIN e AYERS, 2012, p. 6-13).

XML *Schema* (“esquema” XML) é uma recomendação da W3C (*World Wide Web Consortium*) para uma maneira de se representar o vocabulário e as regras formais de um arquivo XML; um modelo contendo um esquema é tipicamente contido em um arquivo XML com a extensão XSD (*XML Schema Definition*). Estes arquivos são usados para definir e validar sintaxes e vocabulários, além de várias definições de metadados (FAWCETT, QUIN e AYERS, 2012, p. 117-120).

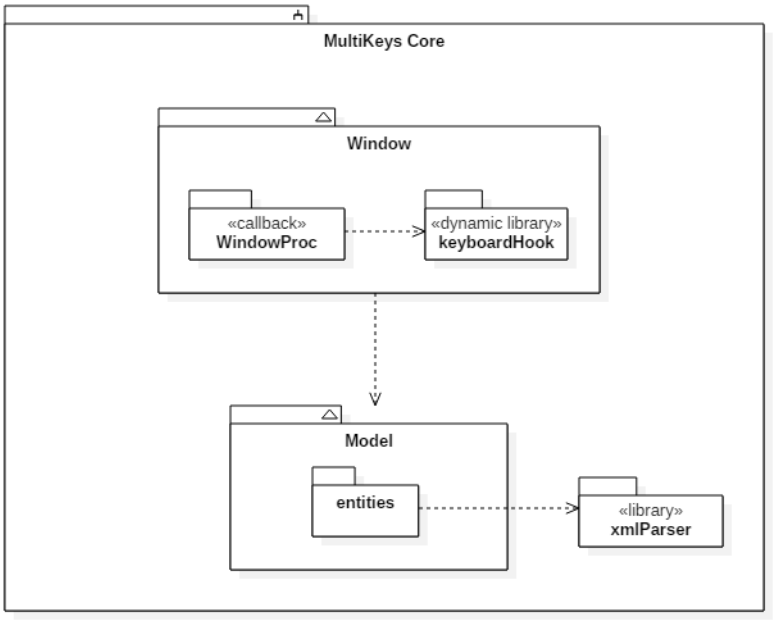
O uso da linguagem de *markup* XML se justifica pela estrutura hierárquica de um *layout* de teclado: cada *layout* contém um ou mais níveis (cada um relacionado a uma combinação de teclas modificadoras), cada nível contém vários mapeamentos de teclas físicas para ações, e cada uma destas ações podem consistir em um ou mais caracteres (geralmente apenas um). Além disso, o usuário pode configurar um ou mais teclados físicos e atribuir *layout*s diferentes a eles.

As vantagens de arquivos XML se alinham às necessidades do programa: arquivos manipuláveis pelo usuário (em disco), interoperáveis, legíveis por humano e máquina, e capazes de representar dados e metadados de maneira hierárquica. Um esquema em XSD define formalmente a estrutura dos arquivos em XML, o que é necessário uma vez que XML será usado como o protocolo de dados entre os dois principais subsistemas do *Multikeys*.

4.3 MANIPULAÇÃO DE MENSAGENS

A parte essencial do *software Multikeys* é sua capacidade de interceptar e manipular a entrada de teclado. Esta funcionalidade está localizada no subsistema *Multikeys Core*, inicializada e finalizada pela interface gráfica do programa. O diagrama de pacotes na Figura 6 descreve a estrutura arquitetural da aplicação.

Figura 6: Diagrama de pacotes da aplicação

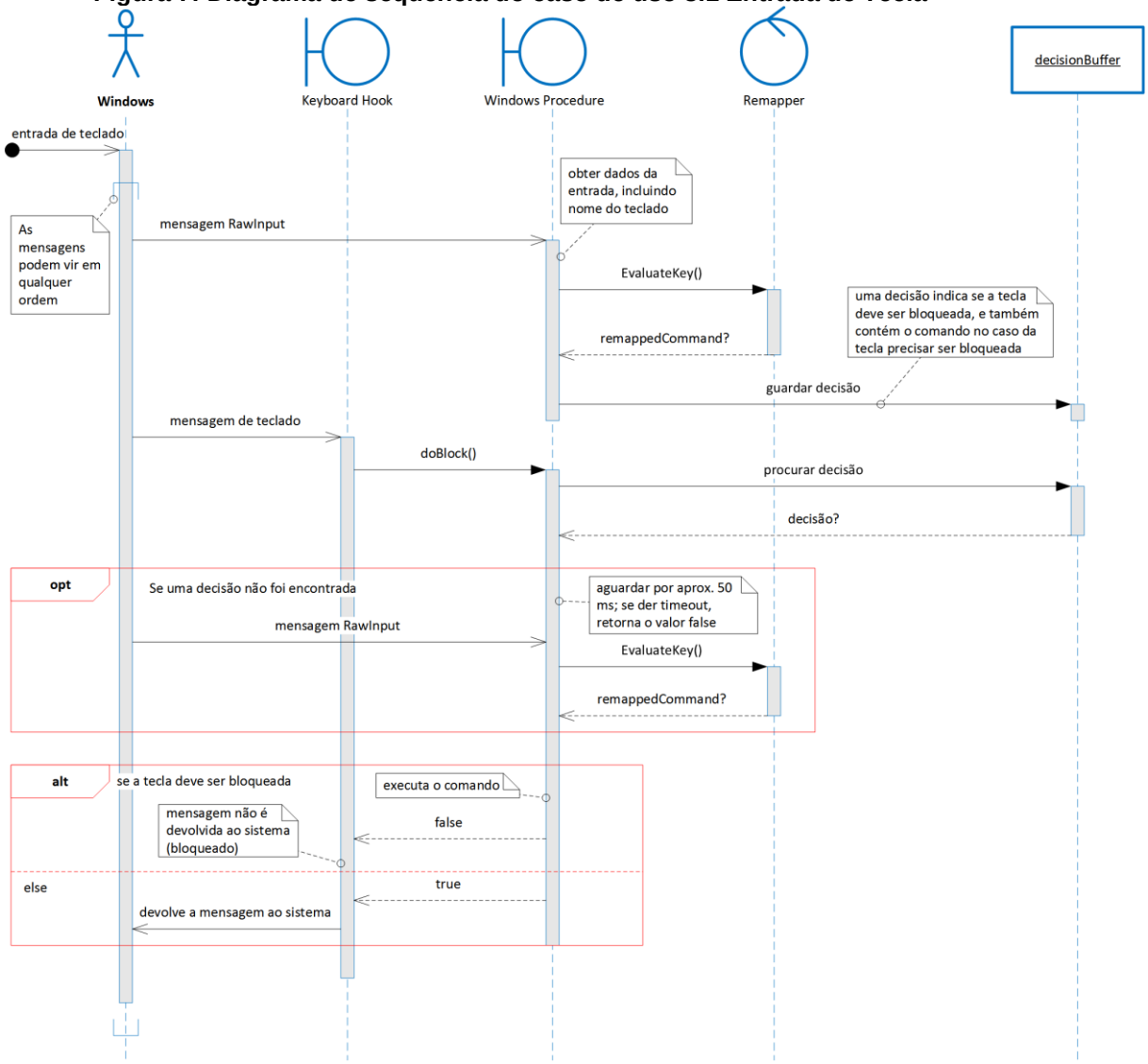


Fonte: Autoria Própria (2017)

O procedimento de janela e o *hook* de teclado são incluídos como componentes da janela principal. Além disso, o modelo interno dos teclados do usuário é implementado como uma biblioteca.

O diagrama de sequência na Figura 7 ilustra o fluxo principal do caso de uso UC3.1 Entrada de Tecla.

Figura 7: Diagrama de sequência do caso de uso 3.1 Entrada de Tecla



Fonte: Autoria Própria (2017)

O símbolo IO representa uma interface; neste caso, a interface é a interação com o sistema operacional através de mensagens, uma vez que o processo em plano de fundo nunca interage diretamente com o usuário. O diagrama ilustra que após receber uma mensagem de teclado do usuário, o sistema operacional envia duas mensagens ao sistema *Multikeys*. A primeira (não necessariamente cronologicamente anterior) é uma mensagem *RawInput*, necessária para a detecção do aparelho que enviou o sinal, e a segunda é uma mensagem de tecla interceptada pelo *hook* de teclado.

Como a mensagem *RawInput* é utilizada para a tomada de decisão (usado para encontrar o comando remapeado se houver algum), este dado deve ser armazenado. O objeto *decisionBuffer* (implementado como uma fila) é usado para guardar estas

decisões até que a mensagem *hook* seja recebida. O *hook* de teclado pode não devolver a mensagem recebida, dependendo da decisão tomada.

O *WndProc* do sistema *Multikeys* é prototipado da seguinte maneira:

```
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
```

E definido mais adiante:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    ...
}
```

A *WndProc* é um procedimento que deve ser chamado a cada vez que uma mensagem precisa ser processada pela janela da aplicação. Dentro da função principal do programa, existe um loop que chama o *WndProc*:

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                     _In_ LPWSTR lpCmdLine,
                     _In_ int nCmdShow)
{
    ...
    while (GetMessage(&msg, nullptr, 0, 0))
    {
            ...
            TranslateMessage(&msg);
            DispatchMessage(&msg);
    }
}
```

A função *DispatchMessage* envia a mensagem para a *WndProc* da janela adequada; neste caso, somente uma *WndProc* existe na aplicação.

Embora o propósito principal do procedimento de janela seja processar mensagens enviadas à aplicação, a maioria das mensagens a serem processadas pelo *Multikeys Core* são mensagens interceptadas adereçadas a outras janelas. A interceptação ocorre tanto através da API *RawInput* e um *Hook* de teclado.

4.3.1 *RawInput* API

A janela principal do programa recebe mensagens da API *RawInput* através de um sistema de assinatura. As seguintes linhas existem para informar ao sistema de

que este programa quer receber mensagem *RawInput* relacionadas à entrada de teclado direcionadas a qualquer janela:

```
// Register for receiving Raw Input for keyboards
   RAWINPUTDEVICE RawInputDevice[1];
   RawInputDevice[0].usUsagePage = 1;
   // usage page = 1 is generic and usage = 6 is for keyboards
   RawInputDevice[0].usUsage = 6;
   // (2 is mouse, 4 is joystick, 6 is keyboard, there are others)
   RawInputDevice[0].dwFlags = RIDEV_INPUTSINK;
   // Receive input even if the registered window is in the background
   RawInputDevice[0].hwndTarget = hWnd;
   // Handle to the target window (NULL would make it follow kb focus)
   RegisterRawInputDevices(RawInputDevice, 1, sizeof(RawInputDevice[0]));
```

A partir deste momento, a janela passa a receber mensagens WM_*INPUT* do sistema operacional, contendo informação de baixo nível referente a toda entrada de tecla do usuário.

Uma parte essencial para o funcionamento do programa é que seja possível extrair o nome do teclado de onde uma entrada de tecla originou. Para isso, as seguintes linhas de código são necessárias para a extração de informação de uma mensagem *RawInput*:

```
GetRawInputData
   ((HRAWINPUT)lParam,
   RID_INPUT,
   rawKeyboardBuffer,
   &rawKeyboardBufferSize,
   sizeof(RAWINPUTHEADER));

raw = (RAWINPUT*)rawKeyboardBuffer;
```

```
GetRawInputDeviceInfo
   (raw->header.hDevice,
   RIDI_DEVICENAME,
   keyboardNameBuffer,
   &keyboardNameBufferSize);
```

As duas primeiras linhas extraem os dados da mensagem *RawInput*, enquanto a próxima chamada extrai o nome do aparelho que gerou a mensagem (outras informações também são extraídas, como *scancodes*).

É necessário usar a API *RawInput* porque esta é a única maneira de distinguir a mensagem entre teclados. A seguir está a chamada de função ao Remapper, que recebe os dados da tecla e responde se a tecla deve ser bloqueada ou não, assim como um comando caso a tecla precise ser bloqueada (o comando realiza diversas ações, como enviar caracteres *Unicode* e simular outras teclas):

```
IKeystrokeCommand * possibleAction = nullptr;
BOOL DoBlock = remapper.EvaluateKey
   (&(raw->data.keyboard), keyboardNameBuffer, &possibleAction);
decisionBuffer.push_back
   (DecisionRecord(raw->data.keyboard, possibleAction, DoBlock));
```

Como não é possível bloquear teclas a partir da API *RawInput*, é necessário armazenar esta decisão. A estrutura *DecisionRecord* serve para armazenar os dados da tecla, uma flag indicando se a tecla deve ser bloqueada, e um comando caso a tecla precise ser bloqueada. A seguir está a definição desta estrutura, com construtores e comentários omitidos para brevidade:

```
struct DecisionRecord
{
   RAWKEYBOARD keyboardInput;
   IKeystrokeCommand * mappedAction;
   BOOL decision;
};
```

4.3.2 *Keyboard Hook*

Um *hook* de teclado intercepta mensagens de entrada de tecla; diferente das mensagens de *RawInput*, *hooks* permitem que o programa bloqueie mensagens, mas não é possível detectar o dispositivo de origem de uma mensagem. O *hook* deve existir em um processo separado para poder interceptar mensagens globais, isto é, não só as direcionadas ao processo atual. Por isso, em um dll separado, o seguinte código existe para colocar um *hook* de teclado na *hook chain* do sistema:

```
BOOL InstallHook(HWND hwndParent)
{
   if (hwndServer != NULL)
   {
            // Already hooked
            return FALSE;
   }
   hookHandle = SetWindowsHookEx(WH_KEYBOARD, (HOOKPROC)KeyboardProc,
            instanceHandle, 0);
   if (hookHandle == NULL)          // security check
            return FALSE;
   hwndServer = hwndParent;
   return TRUE;
}
```

Esta função é exportada no cabeçalho da dll para que possa ser chamada a partir de outro processo:

```
__declspec(dllexport) BOOL InstallHook(HWND hwndParent);
```

A função é chamada normalmente na *WndProc*:

```
InstallHook(hWnd);
```

O parâmetro passado é o *handle* para a janela principal, que é necessário para que o *hook* na dll tenha uma maneira de se comunicar com o resto da aplicação.

A partir deste momento, o procedimento *KeyboardProc* fica registrado na *hook chain*, e toda entrada de tecla deve passar por ele. A seguir está ilustrada uma versão resumida do corpo do procedimento *KeyboardProc*:

```
static LRESULT CALLBACK KeyboardProc
   (int code, WPARAM wParam, LPARAM lParam)
{
   if (code != HC_ACTION)
            return CallNextHookEx(hookHandle, code, wParam, lParam);

   if (SendMessage(hwndServer, WM_HOOK, wParam, lParam))
            return 1;

   return CallNextHookEx(hookHandle, code, wParam, lParam);
}
```

Neste procedimento também ficam certas checagens de segurança que estão descritos nos fluxos alternativos do caso de uso UC3.1, e que são omitidas no código apresentado.

Quando o procedimento retorna 1, a mensagem não é passada adiante; senão, o procedimento deve chamar *CallNextHookEx*() para entregar a mensagem para o próximo procedimento na *hook chain*; nota-se que a mensagem é bloqueada se qualquer *hook* na *hook* chain retorna 1. Adicionalmente, a chamada a *SendMessage* serve para enviar uma mensagem ao procedimento de janela principal, perguntando se a tecla recebida deve ser bloqueada, passando os dados da tecla nos parâmetros *wParam* e *lParam*.

O procedimento de janela principal realiza uma busca pela decisão tomada anteriormente, comparando por tecla virtual e *scancode*.

4.3.3 Prazos e Bloqueamentos

Conforme visto na Figura 7, o fluxo principal começa quando uma mensagem de tecla é interceptada, e termina quando esta é devolvida (ou não) ao destinatário

original. Todo o processamento que ocorre entre estes dois momentos aumenta a latência da mensagem, e pode chegar a degradar a experiência do usuário.

Adicionalmente, o código executado nesta janela de tempo deve ser necessariamente executado de maneira síncrona, uma vez que o resultado do processamento (o bloqueio ou não de uma tecla, assim como uma possível ação remapeada) deve ocorrer sempre na mesma ordem em que o usuário realizou a entrada de sinal.

Isto significa que o fluxo precisa ser bloqueado neste intervalo, uma vez que é necessário saber que ação tomar antes de possivelmente bloquear uma mensagem de teclado. Para evitar que falhas causem atrasos arbitrários no fluxo principal, um prazo é introduzido, com a finalidade de manter o sistema responsivo.

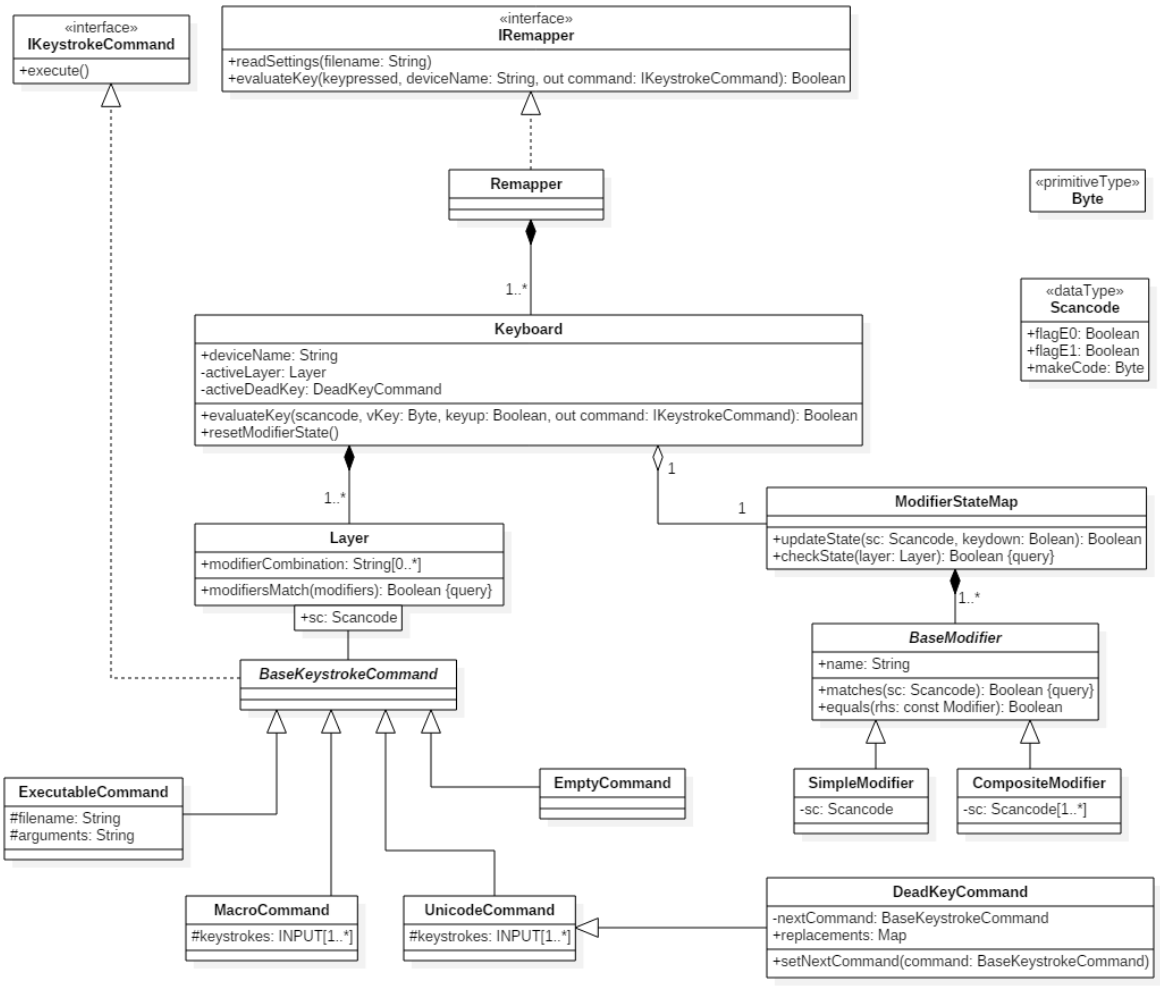
Quando o sistema não consegue processar uma mensagem dentro deste prazo determinado, considera-se que houve falha, e a mensagem não é processada; embora este processo seja *soft real-time*, há a possibilidade de acumulação de mensagens atrasadas, agravando o problema.

4.3.4 Remapper

O componente do programa que mantém o modelo interno dos teclados do usuário, incluindo status de modificadores e teclas de acentuação, é mantida em uma biblioteca. O modelo foi separado em uma biblioteca para maximixar a modularidade do programa, separando as operações primariamente dirigidas a eventos na *WndProc* do modelo orientado a objetos. A Figura 8 contém o diagrama de classes do modelo interno do sistema *Multikeys*.

As interfaces e comandos são implementados de acordo com os padrões de projeto da GoF (*Gang of Four*, autores da obra *Padrões de Projeto*).

Figura 8: Diagrama de classes do modelo interno



Fonte: Autoria própria (2017)

Os seguintes métodos e classes são expostos no *header RemapperAPI*.h; a seguir está uma versão resumida e não comentada para brevidade:

```
namespace Multikeys
{
    typedef class IKeystrokeCommand
    {
    public:
            virtual BOOL execute(BOOL keyup, BOOL repeated) const = 0;
            virtual ~IKeystrokeCommand() = 0;
    } *PKeystrokeCommand;

    typedef class IRemapper
    {
    public:

    virtual bool loadSettings(const std::wstring xmlFilename) = 0;
            virtual bool evaluateKey(
                    RAWKEYBOARD* const keypressed,
                    WCHAR* const deviceName,
                    OUT PKeystrokeCommand* const out_action) const = 0;

            virtual ~IRemapper() = 0;

    } *PRemapper;

    void Create(OUT PRemapper* instance);
    void Destroy(PRemapper* instance);
}
```

De acordo com os princípios de orientação a objetos, *IRemapper* e *IKeystrokeCommand* são interfaces, implementadas em C++ como ponteiros para classes puramente abstratas (isto é, classes que contém somente métodos virtuais). Os destrutores virtuais são necessários para que as classes de implementação possam ter seus próprios destrutores, que são chamados a partir da superclasse.

Além disso, *constantes* são uma particularidade da linguagem C bastante úteis para implementar imutabilidade; o método *evaluateKey*(), por exemplo, é marcado const, indicando que esta chamada de função está garantida a não alterar o objeto no qual é chamado. Um de seus comandos, out_action, é um argumento constante, significando que o parâmetro não pode ser alterado no corpo da função.

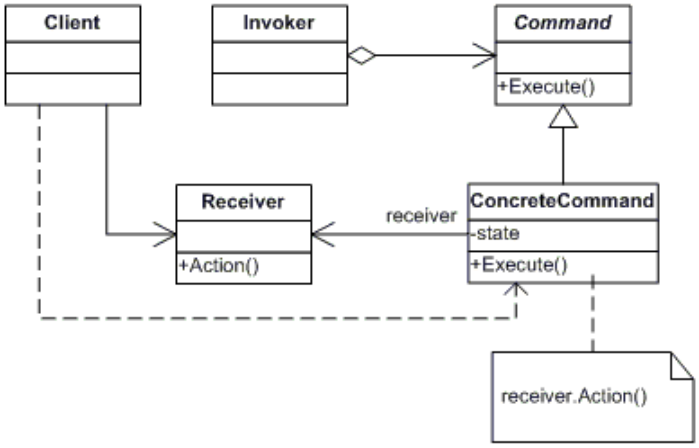
Outro ponto notável são os métodos *Create*() e *Destroy*(), que encapsulam a criação do objeto concreto; desta maneira, a classe de implementação não fica visível. Esta implementação é baseada no padrão *factory method* não parametrizado, mas os métodos devem ser implementados ao invés de classes inteiras dedicadas à criação de objetos.

Outra classe que se vê na Figura 8 é o *IKeystrokeCommand*; a aplicação deve ser capaz de mapear teclas a ações variadas, e de acordo com a GoF, o padrão command é utilizado para “parametrizar objetos com ações a realizar”, e também “para executar [...] pedidos em momentos diferentes”. O uso de comandos remove a

responsabilidade do invocador (neste caso a *WndProc*) de saber quaisquer detalhes sobre como realizar a atividade (simular ações como enviar caracteres *Unicode* ao sistema), promove a extensibilidade (neste caso a possibilidade de adicionar tipos variáveis de ações mapeáveis a teclas), e faz com que as ações possam ser extendidas (como o comando de tecla de acentuação, que é uma especialização do comando de caractere *Unicode*).

A Figura 9 mostra o diagrama de classe do padrão *Command*, de acordo com Gamma et al. (1995).

Figura 9: Padrão de *Design Command*



Fonte: http://www.dofactory.com/net/command-design-pattern (2017)

O Quadro 1: Implementação do Padrão *Command*, a seguir, detalha como o padrão é implementado neste projeto.

Quadro 1: Implementação do Padrão *Command*

Componente	Propósito	Implementação
Client (cliente)	Cria comandos concretos	*Remapper*
Invoker (invocador)	Chama a execução da operação no comando	Procedimento de janela
Command (comando)	Interface para a execução de uma operação	IKeystrokeCommand
ConcreteCommand (comando concreto)	Implementa a execução de uma operação	*UnicodeCommand*, *ExecutableCommand*, etc.
Receiver (receptor)	Realiza as operações associadas aos comandos	API do *Windows*

Fonte: Adaptado de GAMMA et al (1995)

Como a arquitetura do sistema *Multikeys Core* não é puramente orientada a eventos, algumas implementações não podem ser descritas como classes; ainda assim, é possível separar e identificar os diferentes papeis do padrão *Command*.

Em detalhes, uma instância da classe *Remapper* lê as ações associadas a teclas de um arquivo de configuração e instancia comandos para cada um destes, parametrizando-os com os dados lidos do arquivo. Em outro momento, o procedimento de janela busca um comando na instância de *Remapper*, e o executa; as ações consistem em chamadas à API do *Windows*, que funciona como o receptor por realizar tarefas como abrir um arquivo em disco ou enviar um caractere *Unicode* para a janela em foco.

Uma alternativa ao uso de *Commands*, que também é mencionada por Gamma et al. (1995), é o uso de funções de *callback*, que são passadas como parâmetro para serem chamadas em um momento posterior. O problema com esta implementação é que ela não permite (ao menos não facilmente) a implementação de *estados*, enquanto os comandos usados neste sistema precisam conter dados sobre as ações a serem executados (por exemplo, um comando que envia uma sequência de caracteres *Unicode* precisa conter os pontos de código a enviar).

4.4 OBSTÁCULOS

Os dois mecanismos de captura de tecla usados pelo *Multikeys* (*Raw Input* e *keyboard hook*) são empregados em conjunto para possibilitar o bloqueio seletivo de teclas baseado no teclado de origem. Esta estratégia foi documentada por Blecha (2014), e usado anteriormente por Petr Medek no *software* LuaMacros.

Blecha menciona vários obstáculos ao uso desta estratégia; uma vez que *RawInput* e o *hook* de teclado recebem mensagens de teclado em momentos diferentes, existem várias teclas que causam sinais inconsistentes entre as APIs.

Esta seção discute teclas específicas que são problemáticas para o funcionamento do sistema, assim como as estratégias que foram empregadas para a solução de tais problemas.

Nesta etapa, foi considerado apropriado realizar certos experimentos para observar e documentar o comportamento de algumas destas teclas problemáticas. Os resultados apresentados a seguir foram obtidos através de logs no console de debug, através da colocação de chamadas a *OutputDebugString* em partes chave do código, como por exemplo:

```
#if DEBUG
   WCHAR * text = new WCHAR[128];
   swprintf_s(text, 128,
             L"Raw Input: Virtual key %X scancode %s%s%X (%s)\n",
            raw->data.keyboard.VKey,          // virtual keycode
            (raw->data.keyboard.Flags & RI_KEY_E0 ? L"e0 " : L""),
            (raw->data.keyboard.Flags & RI_KEY_E1 ? L"e1 " : L""),
            raw->data.keyboard.MakeCode,      // scancode
            raw->data.keyboard.Flags & RI_KEY_BREAK ? L"up" : L"down");
            // keydown or keyup (make/break)
   OutputDebugString(text);
   memcpy_s(debugText, DEBUG_TEXT_SIZE, text, 128);
   delete[] text;
#endif
```

Os tópicos a seguir também estão listados como regras de negócio no Documento de Requisitos de Negócio do sistema *Multikeys*. Esta seção lista somente as situações mais importantes e relevantes para o trabalho.

4.4.1 AltGr

A tecla *AltGr* é problemática por se comportar de maneira diferente dependendo do *layout* ativo. Nos *layout*s que utilizam esta tecla, o sistema interpreta o Alt direito (*scancode* 0xe0 0x38) como uma combinação de Ctrl e Alt. A sequência de mensagens recebida pelo sistema foi constatada como a seguinte:

Quadro 2: Ordem de mensagens da tecla AltGr

Ordem	API	*Scancode*	Tecla virtual
1	*Hook*		L-Ctrl make
2	*RawInput*	0xe0 0x38 (right alt) make	
3	*RawInput*	0xe0 0x38 (right alt) break	
4	*Hook*	L-Ctrl make: timeout	
5	*Hook*		R-Alt make (2)
6	*Hook*		L-Ctrl break
7	*Hook*	L-Ctrl break: timeout	
8	*Hook*		R-Alt break (3)

Fonte: Autoria Própria (2017)

Nota-se que o *hook* de teclado recebe uma mensagem adicional da tecla *Control* esquerda, que não corresponde a nenhuma mensagem recebida pelo *RawInput*. Neste caso, o padrão a ser detectado é uma mensagem do *hook* contendo um *Control* esquerdo, imediatamente seguido por uma mensagem *RawInput* atrasada do Alt direito. Esta combinação deve ocorrer somente neste contexto.

4.4.2 Pause/Break

Esta é uma tecla sem uso em programas modernos, que possui a propriedade única de enviar um *scancode* de três bytes de comprimento, e é também a única tecla a usar o prefixo 0xe1. Brouwer (2009) afirma que o prefixo 0xe1 serve para indicar que o sinal break também é enviado junto com o sinal make quando a tecla é pressionada, o que é consistente com os resultados do Quadro 3.

Quadro 3: Ordem de mensagens da tecla Pause/Break

Ordem	API	*Scancode*	Tecla virtual
1	*RawInput*	0xe1 0x1d *make*	0x13 Pause *make*
2	*Hook*	0x45 (numlock) *make*	0x13 Pause *make*
3	*RawInput*	0x45 (numlock) *make*	0xff Error *make*
4	*RawInput*	0xe1 0x1d *break*	0x13 Pause *break*
5	*RawInput*	0x45 (numlock) *break*	0xff Error *break*
6	*Hook*	0x13 Pause *make* timeout	
7	*Hook*	0x45 (numlock) *break*	0x13 Pause *break*
8	*Hook*	0x13 Pause *break* timeout	

Fonte: Autoria Própria (2017)

Nota-se que, devido ao *scancode* de três bytes, a API *RawInput* detecta duas mensagens: a primeira contém o *scancode* 0xe1 0x1d, e a segunda contém 0x45 (que por si só representa a tecla *numlock*). Nenhuma destas corresponde à mensagem recebida pelo *hook* (*scancode* 0x45, tecla virtual 0x13). A solução deste problema é fazer com que o *hook* procure pelo *scancode* 0xe1 0x1d quando recebe a mensagem na linha 2.

Além disso, a tecla *pause*/*break* envia um *scancode* diferente quando pressionado juntamente com a tecla Ctrl. No entanto, o *scancode* modificado, 0xe0 0x46, não possui o problema do *scancode* de três bytes, e a mensagem recebida pelo *hook* de teclado contém o *scancode* correto.

4.4.3 *PrintScreen*/*System Request*

A tecla *printscreen* envia dois *scancodes* em sequência, um *shift* (0xe0 0x2a) seguido de seu próprio *scancode* (0xe0 0x37). O *shift* é omitido quando a tecla é pressionada juntamente com *Shift* ou *Ctrl.* O excesso de mensagens *RawInput* não causa problemas ao sistema.

Esta tecla possui o aspecto curioso de não gerar uma mensagem *make* para o *hook* (somente a mensagem *break* é recebida). Por isso, o *hook* deve checar pela mensagem de *scancode* 0xe0 0x37 e tecla virtual 0x2c, e procurar pelas mensagens

RawInput make e *break*. O processo é descrito em mais detalhes na Especificação de Caso de Uso 3.1.

4.5 SIMULAÇÃO DE ENTRADA

A biblioteca *Remapper* é responsável por guardar os mapeamentos entre teclas físicas (identificados por *scancode*) e as ações. Durante a inicialização, o programa lê um arquivo XML contendo o *layout* customizado do usuário, e instancia todas as classes de teclado, modificadores e outros para corresponder às configurações. Isto elimina a necessidade de instanciar quaisquer objetos em tempo de execução.

A estratégia para simular teclas e caracteres é o uso da função *SendInput*, da API do *Windows*. Esta função leva como parâmetro um vetor de *INPUTs*, cada um contendo informação sobre uma entrada de usuário. O vetor é inicializado no construtor, com as informações lidas do arquivo XML.

A ponto de exemplo, o seguinte trecho de um arquivo de configuração mapeia a tecla de *scancode* 0x1f para enviar o caractere *Unicode* U+03C3, corresponde à letra sigma (‘Σ’).

```
<unicode Scancode="1F" TriggerOnRepeat="True">
   <codepoint>3C3</codepoint> <!-- Sigma -->
</unicode>
```

A leitura ocorre através da biblioteca Xerces. Um objeto do tipo *UnicodeCommand* é instanciado e armazenado em um mapa.

Todo comando implementa o método *execute*(); A seguir está a implementação deste método na classe *UnicodeCommand*:

```
BOOL execute(BOOL keyup, BOOL repeated) const override
{
   if (keyup)
            return TRUE;
   else if (!repeated || (repeated && triggerOnRepeat))
            return  (SendInput(
                     inputCount,
                     keystrokes,
                     sizeof(INPUT)) == inputCount ? TRUE : FALSE);
   else return TRUE;
}
```

O objeto *UnicodeCommand* guarda o vetor de *INPUTs* que identifica sua sequência de caracteres Unicode, assim como um número inteiro representando a quantidade de caracteres a serem enviados.

Vale notar que a estrutura *INPUT* é um tipo de união que pode representar vários tipos de entrada de usuário, incluindo teclas virtuais e caracteres *Unicode*. No caso da classe *UnicodeCommand*, a estrutura é utilizada para simular a entrada de caracteres *Unicode*.

Um caso diferente é o *ExecutableCommand*, que ao ser executado abre um arquivo em disco. Um destes comandos é escrito no arquivo de configuração da seguinte maneira:

```
<execute Scancode="06">
        <path>C:\Program Files (x86)\Google\Chrome\Application\chrome.exe</path>
        <arguments>http://stackoverflow.com/</arguments>
</execute>
```

A função *ShellExecute* da API do *Windows* é utilizada para realizar uma operação em um arquivo; neste caso, a operação “open” é realizada no arquivo especificado pelo usuário, causando sua execução.

```
BOOL execute(BOOL keyup, BOOL repeated) const override
{
   if (repeated || keyup) return TRUE;
   HINSTANCE retVal = ShellExecute(NULL,
                              L"open",
                              filename.c_str(),
                              arguments.c_str(),
                              NULL,
                              SW_SHOWNORMAL);
   if ((LONG)retVal <= 32)
            return FALSE;
   return TRUE;
}
```

Estes comandos são obtidos pelo procedimento de janela, e executados no momento adequado, de acordo com o diagrama de sequência da Figura 7.

4.6 EDIÇÃO DE *LAYOUT*

Uma das funcionalidades principais do programa é a customização de *layouts* de teclado pelo usuário, permitindo a ele configurar os comandos associados às teclas físicas de seus teclados. Esta funcionalidade existe no módulo *Multikeys Editor*, na

forma de janelas gráficas onde o usuário pode navegar pelo *layout*, visualizando e editando comandos.

De acordo com os requisitos não funcionais RNF010 – Arquivos de Configuração e RNF011 – Execução Portável, o propósito principal da interface gráfica do programa é permitir que o usuário edite os arquivos de *layout* que são usados pelo *Multikeys Core* para aplicar os remapeamentos de teclas. Conforme o apresentado no documento de arquitetura do software *Multikeys*, este módulo é organizado em uma arquitetura de camadas.

4.6.1 *Multikeys* Editor

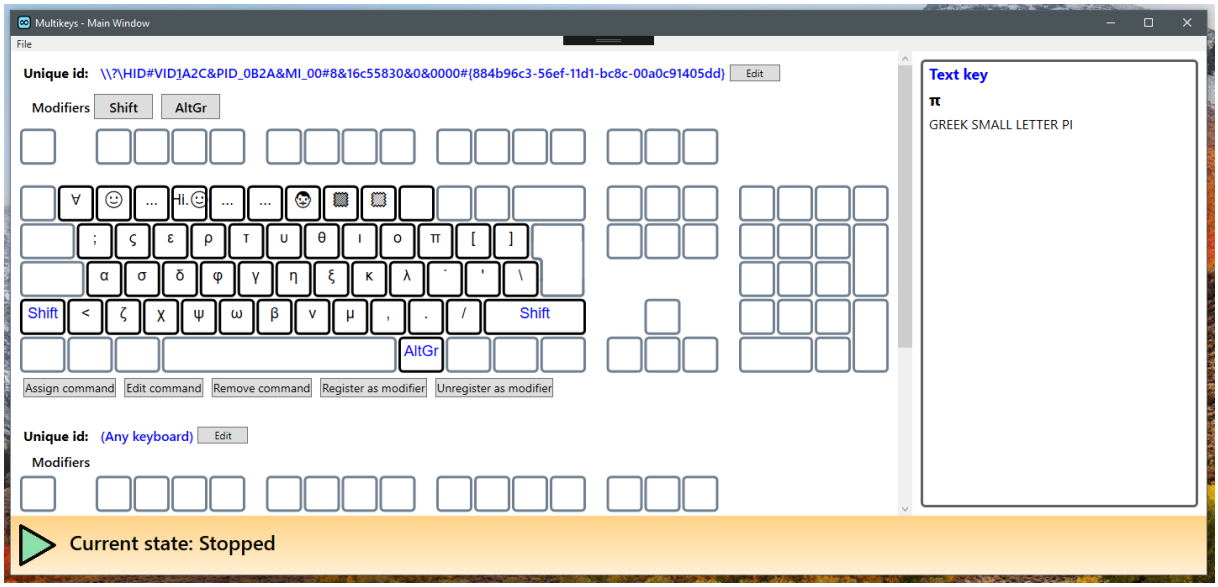
A Figura 10 contém o logo utilizado na aplicação gráfica, que aparece no título da janela e na barra de tarefas.

Figura 10: Logo da aplicação gráfica *Multikeys*

Fonte: Autoria própria (2017)

A interface gráfica do programa utilizou vários controles customizados (*user controls*), a fim de representar graficamente os itens do modelo, assim como para organizar os elementos gráficos da interface. Por exemplo, o controle chamado *KeyControl* representa uma tecla e a ação a ser executada quando esta é pressionada no teclado físico. A Figura 11 mostra uma captura de tela da janela principal do programa.

Figura 11: Captura de tela do *Multikeys Editor*



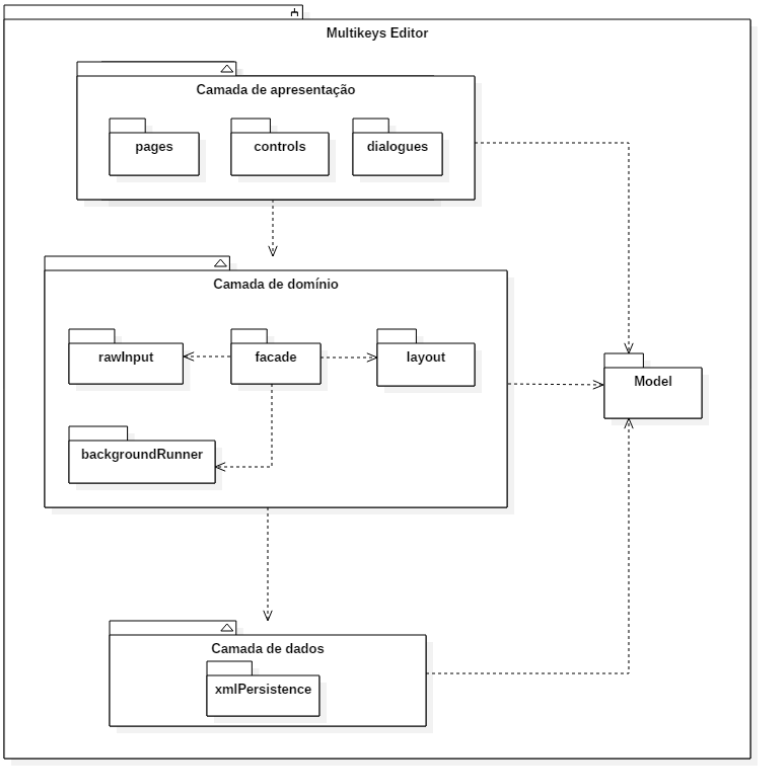
Fonte: Autoria própria (2017)

Na imagem, o programa tem um *layout* carregado, contendo remapeamentos para dois teclados. A tela mostra as informações relevantes e opções para editar os *layout*s; as teclas modificadoras, indicadas por teclas com texto em azul, podem ser clicadas na tela para acessar as outras camadas do *layout*.

Através do controle na parte inferior da janela, o usuário é capaz de iniciar e terminar a execução do *Multikeys* Core, que aplica os remapeamentos.

A Figura 12 ilustra a arquitetura do módulo da interface gráfica do sistema *Multikeys*. A arquitetura separa a lógica de domínio da apresentação, e também a camada de dados responsável por ler e escrever os arquivos de configuração em XML.

Figura 12: Diagrama de modelo da Interface Gráfica



Fonte: Autoria própria (2017)

4.6.2 Estrutura dos Controles

Um controle em WPF é um elemento gráfico que pode ser usado como um componente na interface gráfica. É possível definir controles customizados, chamados de *user controls*, que são usados neste projeto para representar entidades como teclas e camadas de teclado.

Conforme a Figura 12, os controles usados na aplicação são definidos no *namespace controls* da camada de apresentação. Os controles correspondem aproximadamente ao modelo de domínio; por exemplo, o controle *LayerControl* corresponde a uma camada de teclado, e contém uma coleção de controles *KeyControl*, que podem assumir diversos formatos (incluindo o formato de um *return* do padrão ISO).

Assim como o modelo de domínio, os controles são organizados em uma árvore; a janela principal contém um controle representando um *layout* (isto é, contendo o arquivo atualmente carregado, e outras informações relevantes ao arquivo de configuração), que contém vários controles de teclado, e assim por diante.

Cada controle pode ser inicializado com um objeto de domínio, e também possui um método para obter um novo objeto de domínio que o representa. Estes métodos chamam os métodos correspondentes nos controles contidos neste, e desta maneira, a janela principal *precisa interagir somente com o controle de Layout para carregar e obter objetos de layout.*

4.6.2.1 Uso de Eventos nos Controles

Os eventos disponíveis na plataforma .NET são usados em WPF para vários tipos de notificação, como cliques em botões e entrada de texto; os controles implementados no pacote *controls* utilizam eventos para notificar outros controles das atualizações no modelo; por exemplo, quando uma tecla é selecionada no controle de *Layer*, o controle de *Layout* é notificado para mostrar um painel contendo os detalhes da tecla. Na Figura 11, o painel de detalhes pode ser visto à direita do controle de *layout*.

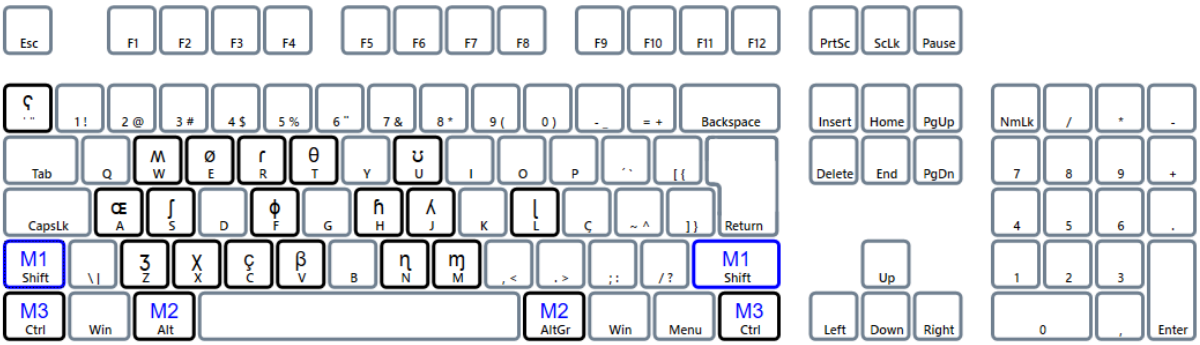
4.6.3 Localização

Uma característica importante dos controles em WPF é a possibilidade de carregar elementos de texto a partir de um dicionário. Desta maneira, a localização pode ser implementada através de vários dicionários que podem ser alterados em tempo de execução, afetando títulos de janelas, textos de *labels* e botões, e outros.

A utilização de dicionários permite a adição futura de novos idiomas.

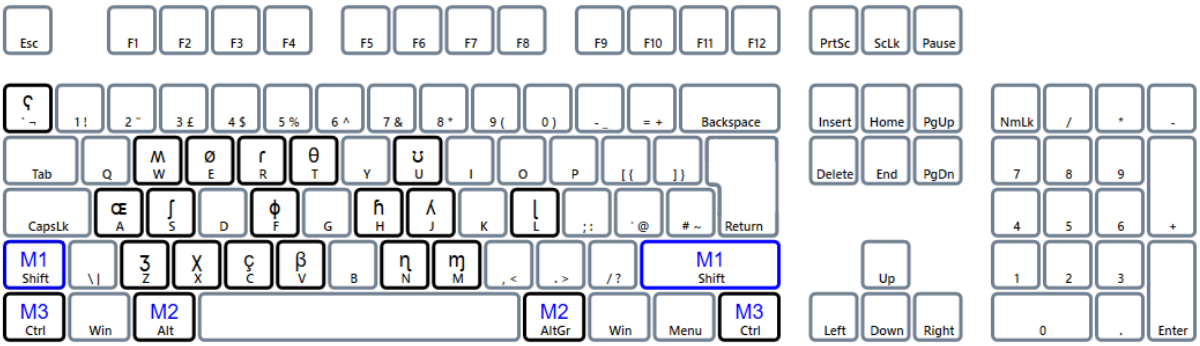
Adicionalmente, o *layout* usado para renderizar o controle de teclado pode ser alterado de acordo com as preferências do usuário, de acordo com vários padrões físicos e lógicos. As capturas de tela nas figuras: Figura 13 Figura 14 e Figura 15 mostram, respectivamente, um *layout* configurado em ABNT-2 (*layout lógico* ABNT-2), ISO (*layout* lógico En-UK) e ISO (*layout* lógico DE).

Figura 13: Exemplo de *layout* configurado em ABNT-2



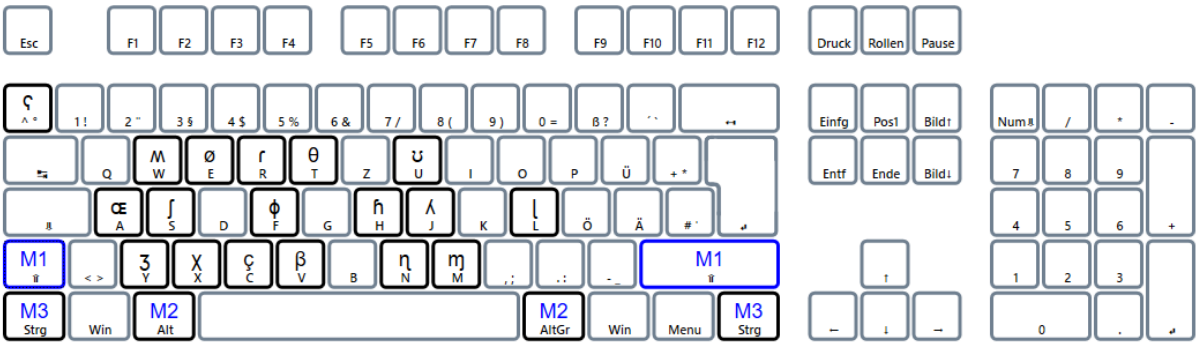
Fonte: Autoria própria (2017)

Figura 14: Exemplo de *layout* configurado em ISO (En-UK)



Fonte: Autoria própria (2017)

Figura 15: Exemplo de *layout* configurado em ISO (DE)



Fonte: Autoria própria (2017)

Os *layouts* físicos determinam quais teclas físicas são apresentadas visualmente, e também determinam seu posicionamento. Por exemplo, o *layout* ABNT-2 possui duas teclas não presentes no ISO (*International Organization for Standardization*): a tecla à esquerda do *Shift* direito, e o divisor de milhares do teclado numérico.

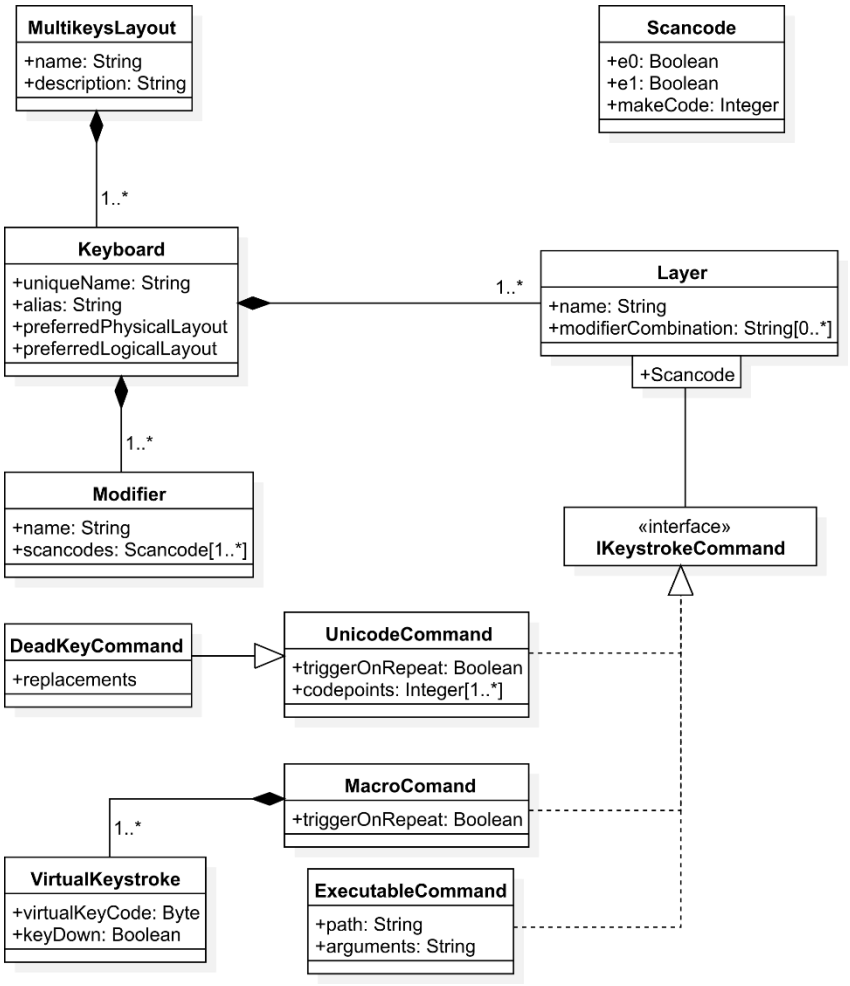
O *layout* lógico é o usado para apresentar visualmente os nomes das teclas para finalidades de referência pelo usuário. A Figura 14 e a Figura 15 apresentam *layout*s cujas configurações somente diferem pelas anotações escritas na tela.

O sistema carrega os *layouts* a partir de arquivos na pasta de recursos do projeto, que é lido até mesmo para apresentar as opções ao usuário. Desta maneira, a edição e adição futura de *layout*s são facilitadas.

4.6.4 Modelo de Domínio

O modelo de domínio do módulo *Multikeys Editor* (Figura 16) possui várias similaridades com o modelo presente no módulo *Multikeys Core*. Porém, este modelo não possui comportamentos, e serve somente como modelo de dados. Uma única instância da classe *MultikeysLayout* armazena toda a informação de um *layout*.

Figura 16: Diagrama de classes do pacote *model* (*Multikeys Editor*)



Fonte: Autoria própria (2017)

Devido ao uso de arquivos XML para persistir as informações contidas em um *layout*, a camada de persistência do sistema é relativamente simples, sendo responsável por primariamente duas funções:

1. Ler um arquivo de configuração e gerar uma instância da classe *MultikeysLayout* que represente as informações presentes no arquivo, e

2. Persistir uma instância da classe *MultikeysLayout* em um arquivo apontado por um caminho especificado.

Além de simplificar a persistência de dados, o uso de XML permite a validação do modelo usando-se o arquivo de esquema XSD que define a estrutura do arquivo de *layout*. O arquivo de esquema é usado para validar um arquivo antes da leitura, e para determinar se uma instância de *MultikeysLayout* gera uma estrutura *xml* válida para escrita.

4.6.4.1 Quantidade Possível de Camadas de Teclado

Como cada combinação de teclas modificadoras em um teclado pode corresponder a uma camada válida de mapeamentos, a quantidade de camadas possíveis cresce exponencialmente em relação à quantidade de modificadores registrados em um determinado teclado.

Para *layouts* que usam uma pequena quantidade de modificadores, este problema não afeta a performance do sistema; um teclado com apenas os modificadores *Shift* e AltGr registrados possui quatro camadas válidas (sem modificadores, *Shift*, AltGr e *Shift*+AltGr). Porém, é possível registrar uma quantidade arbitrária n de modificadores em um teclado, resultando em 2^n possíveis camadas.

Uma possível solução para este problema seria a implementação de um limite de modificadores registrados em cada teclado, ou um limite de camadas não vazias mapeadas em um teclado.

Porém, para evitar a introdução de limites artificiais, uma estratégia alternativa foi usada. A lista de camadas em um teclado somente contém instâncias das camadas mapeadas. Uma camada vazia é apresentada para o usuário normalmente, disponível para mapeamentos; porém, o objeto de classe *Layer* subjacente só é instanciado quando há ao menos um mapeamento registrado na camada.

4.6.5 Interação com *Multikeys* Core

Conforme ilustrado na Figura 4, o subsistema *Multikeys Editor* é uma camada acima do *Multikeys Core*, e encapsula suas funcionalidades. A comunicação entre os dois módulos é mínima; não existem, por exemplo, chamadas de métodos entre estes.

Ao invés de utilizar uma comunicação por meio de chamadas de métodos, a interação entre os subsistemas se dá por meio de mensagens do sistema. O subsistema *Multikeys Editor* é capaz, por exemplo, de enviar uma mensagem de finalização ao processo *Multikeys Core* para requisitar sua terminação. O envio de mensagens é possível porque o *Editor* inicia o processo *Core* e mantém uma referência a sua janela (mesmo que esta não seja desenhada na tela).

Além de iniciar o processo em plano de fundo apenas para remapear teclas, existe outro ponto em que a aplicação *Multikeys Editor* interage com o *Multikeys Core*, para detectar o nome de um dispositivo de teclado do usuário.

Neste caso, um processo é iniciado com uma opção para somente escutar uma entrada de teclado e devolver o nome do dispositivo. Como processos podem retornar somente códigos inteiros, a saída padrão é utilizada para devolver o nome do dispositivo para o *Multikeys Editor*, que obtém esta informação através do redirecionamento da saída padrão do processo.

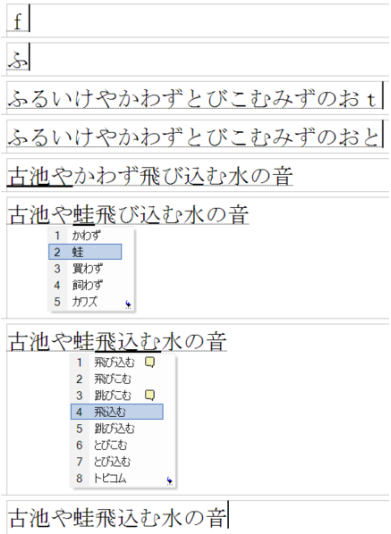
4.7 LIMITAÇÕES

4.7.1 IMEs

Um editor de método de entrada (*Input Method Editor*, abreviado IME) é usado no processamento da entrada de texto em linguagens que possuem muitos caracteres para serem dispostos em um teclado.

A Figura 17 ilustra o funcionamento de uma IME, transformando o texto escrito conforme necessário; uma IME é tipicamente utilizada em conjunto com um *layout* específico; por exemplo, as teclas adicionais no *layout* físico JIS (*Japanese Industrial Standards*) (Figura 18) têm a função primária de interagir com uma IME para a entrada de texto em japonês.

Figura 17: Funcionamento de um IME



Fonte: adaptado de https://commons.wikimedia.org/wiki/File:IME_demonstratie_-_Matsuo_Bashou_-_Furu_ikeya_kawazu_tobikomu_mizuno_oto.png (2007)

Figura 18: Parte alfanumérica do *layout* JIS

Fonte: Adaptado de BROUWER (2009).

Como a implementação de uma IME está fora do escopo do sistema *Multikeys* (conforme foi especificado no documento de visão do sistema *Multikeys*), não é possível usar o sistema para replicar o funcionamento completo de *layout*s que utilizam IMEs. É possível, porém, utilizar um IME em um teclado não mapeado, em conjunto com um ou mais teclados mapeados no sistema *Multikeys*.

Nota-se que as teclas adicionais de entrada de texto ainda podem ser remapeadas normalmente no sistema.

4.7.2 Tecla Fn

A tecla de função está presente como uma tecla modificadora em vários teclados de tamanho reduzido. Porém, como foi descrito na regra de negócio RN3.4 Tecla Modificadora Fn, esta tecla não envia um *scancode* ao sistema operacional, e por isso, não pode ser detectada pelo sistema.

Como resultado disso, o usuário não pode remapear a tecla Fn, nem quaisquer combinações contendo a tecla Fn. A descrição completa da regra de negócio RN3.4 pode ser encontrada no documento de regras de negócio do sistema *Multikeys*.

4.8 PUBLICAÇÃO

O código foi publicado em um repositório público hospedado no GitHub, acessível pela url https://github.com/rafaelktakahashi/*Multikeys*.git. O código foi publicado sob a licença BSD (*Berkeley Software Distribution*) modificada, presente no arquivo LICENSE na raiz do projeto. Um README foi provido contendo uma introdução ao projeto.

CONSIDERAÇÕES FINAIS

Embora a entrada de teclado em uma aplicação seja tipicamente uma funcionalidade trivial de se implementar, a manipulação e o bloqueio seletivos de sinais de acordo com o dispositivo de origem requer mais conhecimento técnico sobre o processamento de mensagens realizado pelo sistema operacional.

O *software Multikeys* implementou a funcionalidade em questão através do uso conjunto da API *RawInput* e um *hook* de teclado, ambos pertencentes à API do *Windows*. Porém, a utilização destas duas estratégias simultaneamente resulta em numerosos comportamentos incorretos que devem ser tratados individualmente. A correção destes comportamentos precisou de um bom entendimento do padrão de teclado PS/2, assim como o funcionamento de mensagens no sistema *Windows*.

A arquitetura do sistema foi um estágio interessante no desenvolvimento por conter tanto componentes orientados a eventos (devido ao sistema de mensagens em *Windows*) quanto orientados a objetos (a fim de implementar o modelo de *layouts* e a aplicação de edição de *layouts*).

Além disso, várias decisões arquiteturais foram tomadas diretamente como resultado de certos requisitos, como o armazenamento de dados em XML e a utilização das linguagens C++ e C# em módulos diferentes do sistema.

A aplicação resultante possui um conjunto único de funcionalidades, por possibilitar a configuração e o uso de *layout*s de teclado diferentes em cada dispositivo de entrada. Considera-se que os objetivos propostos no início deste trabalho foram atingidos, uma vez que as funcionalidades propostas estão em funcionamento, incluindo a interface gráfica do programa.

Espera-se que o sistema desenvolvido possa ser utilizado para atender as necessidades inicialmente colocadas, e também que este continue a ser desenvolvido, possivelmente por terceiros para fins relacionados ou não; a publicação do código em *open-souce* reflete este sentimento.

REFERÊNCIAS

ACETO, Luca et al., ***Reactive Systems: Modelling, Specification and Verification***, Cambridge University Press, 2007. Disponível em: < http://www.cs.ioc.ee/yik/schools/win2007/ingolfsdottir/sv-book-part1.pdf>. Acesso em 12 out. 2017.

GITHUB. ***AutoHotKey_L***. Repositório GitHub, 2017 Disponível em: <https://github.com/Lexikos/AutoHotkey_L>. Acesso em 05 jun. 2017.

BONÉR, Jonas; KLANG, Viktor, ***Reactive Programming vs. Reactive Systems***. O'Reilly Media, www.oreilly.com, 2016. Disponível em: <https://www.oreilly.com/ideas/reactive-programming-vs-reactive-systems>. Acesso em 21 out. 2017.

BROUWER, Andries. ***Keyboard Scancodes***, Fakulteit Wiskunde en Informatica, Eindhoven, Países Baixos, 2009. Disponível em: <https://www.win.tue.nl/~aeb/linux/kbd/*scancode*s.html#toc1>. Acesso em 05 mai. 2017.

BLECHA, Vít. ***Combining Raw Input and Keyboard Hook to Selectively Block Input from Multiple Keyboards***, Code Project, 2014. Disponível em: <https://www.codeproject.com/Articles/716591/Combining-Raw-Input-and-keyboard-*Hook*-to-selective>. Acesso em 05 jun. 2017.

CHAPWESKE, Adam. ***The PS/2 Keyboard Interface***, Computer-Engineering.org, 2013. Disponível em: <http://www.computer-engineering.org/ps2keyboard/>. Acesso em 06 jun. 2017.

CONSTABLE, Peter. ***Understanding Unicode***, SIL International, 2001. Disponível em: <http://scripts.sil.org/IWS-Chapter04a>. Acesso em 06 jun. 2017.

FARINES, Jean-Marie; FRAGA, Joni da Silva; OLIVEIRA, Rômulo Silva, **Sistemas de Tempo Real**, Universidade Federal de Santa Catarina. Florianópolis, 2000. Disponível em: <http://www.romulosilvadeoliveira.eng.br/livro-tr.pdf>. Acesso em 17 out. 2017.

FAWCETT, Joe; QUIN, Liam R. E.; AYERS, Danny. ***Beginning XML***. 5th ed. John Wiley and Sons: Indiana, EUA, 2012

FOWLER, Martin**,** ***Focusing on Events***, *www.martinfowler.com*, *Thoughtworks*, 2006a. Disponível em: <https://martinfowler.com/eaaDev/EventNarrative.html>. Acesso em 20 out. 2017.

FOWLER, Martin, ***Event Collaboration***, www.martinfowler.com, *Thoughtworks*, 2006b. Disponível em: <https://martinfowler.com/eaaDev/EventCollaboration.html>. Acesso em 20 out. 2017.

FOWLER, Martin, ***What do you mean by "Event Driven"?***, *Martinfowler*.com*, Thoughtworks*, 2017. Disponível em: <https://martinfowler.com/articles/201701-event-driven.html>. Acesso em 08 out. 2017.

GAMMA et al., ***Design Patterns: Elements of Reusable Object-Oriented Software***, Reading, Mass: Addison-Wesley, 1995.

HOHPE, Gregor. ***Programming without a Call Stack****: Event-Driven Architectures*. *Enterprise Integration Patterns*, 2006. Disponível em: <http://www.enterpriseintegrationpatterns.com/docs/EDA.pdf>. Acesso em 06 out. 2017.

HOSKEN, Martin. ***An Introduction to Keyboard Layout Design Theory***: What Goes Where?, SIL *International*, 2003. Disponível em: <http://scripts.sil.org/keybrddesign> Acesso em 07 jun. 2017.

JUVKA, Kanaka, ***Real-Time Systems****. Carnegie Mellon University,* 1998. Disponível em: <https://users.ece.cmu.edu/~koopman/des_s99/real_time/>. Acesso em 20 out. 2017.

KAPLAN, Michael S.; WISSINK, Cathy. ***Unicode and Keyboards on Windows***. *23rd Internationalization and Unicode Conference*, República Checa, 2003. Disponível em: <https://sites.google.com/site/talapornmon/Unicode-KbdsonWindows.pdf>. Acesso em 07 jun. 2017.

LEVINA, Olga; STANTCHEV, Vladmir, *A Model and an Implementation Approach for Event-Driven Service Orientation*. ***International Journal on Advances in Software***, vol. 2, 2009. Disponível em: <http://www.academia.edu/993297/A_Model_and_an_Implementation_Approach_for_Event-Driven_Service_Orientation>. Acesso em 07 jun. 2017.

LIPARI, Giuseppe; PALOPOLI, Luigi, ***Real-Time Scheduling:*** *From Hard to Soft Real-Time Systems, Cornell University,* arXiv, 2015, Disponível em: <https://arxiv.org/pdf/1512.01978.pdf>. Acesso em 12 out. 2017.

MACDONALD, Matthew, ***Pro WPF 4.5 in C#***: *Windows Presentation Foundation in .NET 4.5, 4th Ed*. Editora Apress, 2012.

MEDEK, Petr, ***HID macros*****,** Repositório GitHub, *2014.* Disponível em: <https://github.com/me2d13/luamacros>. Acesso em 07 jun. 2017.

MICROSOFT, ***Windows API Index***, Microsoft, 2017. <https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx>. Acesso em 05 nov. 2017.

PALMER, Grant; BARKER, Ken, ***Beginning C# 2008 Objects***, 1st Ed. Editora Apress, 2009.

PETZOLD, Charles, ***Programming Windows****, 5th ed*. *Microsoft Press,* Washington, 1998.

RAYMOND, David. ***Keyman*** – *Keyboard Systems for Windows, Mobile Devices and Web*. Tavultesoft Pty Ltd (SIL International) 2014. Disponível em: <http://scriptsource.org/cms/scripts/page.php?item_id=entry_detail&uid=vsphmhqr6h>. Acesso em 07 jun. 2017.

RICHTER, Jeffrey; NASARRE, Christopher. ***Windows via C/C++***, 5th ed. *Microsoft Press*, Washington, 2008.

SAHAY, Sourav, ***Object Oriented Programming with C++***, *2nd ed. New Delhi: Oxford University Press*, 2012.

STANKOVIC, John A. ***Real-Time Computing***. University of Massachusetts. Amherst, MA, 1992. Disponível em: <https://pdfs.semanticscholar.org/9f6a/3616dc9d9e1a8481681131dd7ae42aac3ac3.pdf>. Acesso em 17 out. 2017.

UNICODE CONSORTIUM. ***The Unicode Standard****, Version 9.0.0*. *Mountain View,* CA: *The Unicode Consortium*, 2016. Disponível em: < http://www.unicode.org/versions/Unicode9.0.0/UnicodeStandard-9.0.pdf>. Acesso em 08 jun. 2017.

WELLS, John. ***An Update on Phonetic Symbols in Unicode***. University College London, Saarbrücken, 2007. Disponível em: <http://www.icphs2007.de/conference/Papers/1357/1357.pdf>. Acesso em 08 jun. 2017.

APÊNDICE A: DOCUMENTO DE VISÃO DE PROJETO

Multikeys – Aplicação para Remapemento de Teclas em *Windows*

FATEC
Faculdade de Tecnologia de Mogi das Cruzes
Tecnologia em Análise e Desenvolvimento de Sistemas | Tecnologia em Agronegócio | Tecnologia em Recursos Humanos

Histórico de Versões

Data	Versão	Descrição	Autor	Revisor
02/05/2017	0.1	Versão preliminar	Rafael Kenji	-
27/05/2017	1.0	Primeira versão	Rafael Kenji	-

Índice

1 Objetivo

Este documento trata das necessidades de negócio, da justificativa do projeto, do entendimento atual das necessidades do cliente, e descreve resumidamente o produto que atenderá a esses requisitos.

O documento alinha as expectativas dos interessados e formaliza o início do projeto, definindo necessidades e funcionalidades de alto nível no programa *Multikeys*.

1.1 Escopo

O escopo deste projeto trata do desenvolvimento de uma aplicação desktop para a criação de *layouts* customizados que possam ser usados em teclados específicos conectados à máquina, através da interceptação e modificação da entrada de usuário com a API do *Windows*. Uma interface gráfica deve existir para facilitar a configuração do programa.

Os *layouts* customizados devem replicar o funcionamento de um *layout* tradicional usado no sistema operacional *Windows*, incluindo teclas modificadoras e teclas de acentuação; tecnologias adicionais, como editores de entrada de texto (IME – *Input Method Editors*) estão fora do escopo do projeto.

1.2 Referências

Este documento faz referência a outras documentações deste projeto:

- Documento de Requisitos: Formaliza os requisitos funcionais e não funcionais do projeto, definindo-os mais a fundo.
- Documento de Regras de Negócio: Formaliza as regras de negócio do projeto.
- Documento de Casos de Uso: Lista todos os casos de uso do projeto, juntamente com cada fluxo de atividades.

2 Necessidades do Projeto

Existem usuários de computador que frequentemente precisam inserir caracteres especiais em seus computadores. Entre eles, podemos citar:

- Matemáticos e professores, que precisam redigir símbolos como “→” (implicação), “Δ” (Delta) e “∈” (pertence a), entre numerosos outros;

- Linguistas que fazem uso frequente do IPA (Alfabeto Fonético Internacional), composto de numerosos caracteres especiais, como “ʔ” (oclusiva glotal) e “ɹ” (aproximante alveolar);

A maioria destes caracteres são de uso em áreas específicas, e o sistema operacional não oferece *layouts* que os contenha para fácil acesso. Por este motivo, é necessário utilizar as funcionalidades dos editores de texto, ou de algum *software* de terceiros, para redigir caracteres especiais.

Uma opção é utilizar um teclado adicional mapeado a estes caracteres especiais.

Através de *software*, é possível estender a funcionalidade dos teclados do usuário, como evidenciado em jogos de computador, e em programas de *scripts* como AutoHotKey. Mesmo assim, os programas existentes para criação e *layouts* customizados sofrem de numerosos problemas, como código fechado, interface não intuitiva e a falta de suporte do sistema operacional para múltiplos teclados conectados ao computador.

Este último problema em particular torna impossível um usuário utilizar múltiplos aparelhos de entrada para diferentes propósitos (por exemplo, programar um teclado para inserir símbolos matemáticos) sem afetar todos os outros teclados conectados ao computador.

O desenvolvimento deste sistema possibilitaria a criação de *layouts* customizados, dispostos de acordo com as preferências pessoais do usuário. A possibilidade de *layouts* serem simultaneamente aplicados a teclados específicos tornaria possível o usuário possuir mais de um dispositivo de entrada, e mapear diferentes *layouts* a estes.

Para os usuários que fazem uso frequente de caracteres especiais, isto significaria fácil acesso a qualquer símbolo representável em *Unicode*; além de usos em áreas especializadas (como em matemática e linguística), o sistema também possibilitaria a entrada de texto usando múltiplos teclados para mais de um idioma. Também seria possível a criação de um teclado de macros, juntamente com uma interface gráfica para que o usuário não precise de conhecimentos prévios para a utilização do programa.

3 Objetivo do Projeto

Desenvolver uma aplicação de *desktop* capaz de:

- Permitir ao usuário utilizar *layouts* customizados, que ofereçam similar funcionalidade aos *layouts* existentes no sistema operacional; isto inclui teclas de acentuação e teclas modificadoras (Shift, Ctrl, Alt), sem a necessidade da instalação de *layouts* no sistema operacional;
- Oferecer a criação fácil e interativa de *layouts* customizados;

- Interceptar, analisar e alterar os sinais de entrada dos teclados do usuário, a fim de simular um *layout* customizado;
- Atribuir diferentes *layouts* a cada teclado conectado ao computador, possibilitando seu uso simultâneo.
- Adicionalmente, atribuir sequências de caracteres e execução de programas pré-determinados a teclas de um teclado.

4 Declaração Preliminar de Escopo

4.1 Descrição

O sistema de criação de *layouts* permitirá ao usuário atribuir qualquer caractere especial, ou sequência destes, assim como a execução de programas, às teclas de um teclado convencional, a fim de estender a funcionalidade dos mesmos. A possibilidade de inserir quaisquer caracteres especiais, enviados a qualquer programa no *desktop* do usuário, trará mais conveniência para os usuários que frequentemente precisam inserir caracteres que não estão presentes nos *layouts* do sistema operacional.

Além da configuração de *layouts*, outra funcionalidade principal do programa é a possibilidade de reconhecer dispositivos de entrada específicos, a fim de responder de maneira diferente a múltiplos teclados conectados ao computador. Uma aplicação possível para esta funcionalidade é a utilização de dois (ou mais) *layouts* de teclado simultaneamente para a escrita de texto em múltiplos idiomas.

4.2 Produtos a Serem Entregues

Os seguintes itens são considerados produtos do projeto:
- Repositório público contendo
- Instalador da aplicação, incluindo a interface gráfica;
- Documentação gerada durante o desenvolvimento do projeto;
- Manual de uso do programa.

4.3 Requisitos

Nesta seção estão listados os requisitos preliminares, descritos em alto nível e sem detalhamento muito aprofundado. Uma descrição mais precisa de cada item está disponível no documento de requisitos.

4.3.1 Requisitos Funcionais

4.3.1.1 Aplicação

A aplicação principal do programa deve ser capaz de simular as funcionalidades de *layouts* de teclado já existentes no sistema operacional. Isto é:

- Mapear sinais de teclas físicas (*scancodes*) a caracteres *Unicode*;
- Suportar vários estados de modificadores, dependendo do estado de cada tecla modificadora (Shift, AltGr e possivelmente outros);
- Suportar teclas de acentuação;

Além disso, o programa possui os seguintes requisitos adicionais, a fim de oferecer as funcionalidades propostas nas seções anteriores:

- Mapear teclas físicas a sequências de caracteres *Unicode* ou sinais simulados de teclas;
- Mapear teclas físicas à abertura de arquivos no sistema, incluindo executáveis;
- Discriminar sinais vindos de diferentes dispositivos de entrada; mais especificamente, distinguir entre sinais recebidos de diferentes teclados, a fim de prover comportamento diferenciado definido pelo usuário;

Embora editores de entrada (IME – *input method editor*) sejam usados para a entrada de certas linguagens, como o japonês, sua implementação não está prevista como um dos requisitos deste sistema, uma vez que as funcionalidades de um editor de entrada vão além do escopo do projeto proposto neste documento.

4.3.1.2 Interface Gráfica

A seguir estão listados os requisitos funcionais referentes à interface gráfica do sistema:

- Visualizar *layouts* customizados, incluindo estados de modificadores, lidos a partir de arquivos de configuração xml.
- Criar, editar, salvar e deletar *layouts* customizados;
- Editar os mapeamentos de teclas físicas dentro de cada *layout*;
- Aplicar *layouts* existentes a teclados conectados ao computador;
- Iniciar e interromper a execução dos *layouts*;
- Alterar a linguagem de exibição.

4.3.2 Requisitos Não-Funcionais

Plataforma: O sistema deve funcionar, com todas as suas funcionalidades, nos sistemas operacionais *Windows* 7 ou mais recente.

Tempo de resposta: Como a aplicação deverá responder à entrada de usuário em tempo real, o tempo de resposta do programa deve ser suficientemente rápida, dado que atrasos perceptíveis na resposta do programa em tempo de execução são indesejáveis ao usuário.

Robustez: No caso de erro, é indesejável que a atividade seja interrompida (por exemplo, mostrando-se uma mensagem de erro em uma janela *popup*). Erros de execução devem ser tratados de maneira que não causem efeitos colaterais, e possibilitem a retomada de atividades o mais rápido possível.

Recursos: Como o programa ficará executando como um processo no plano de fundo (*background process*), é indesejável que este utilize muitos recursos do sistema, ou que bloqueie o uso de quaisquer recursos (salvo os teclados) por outras aplicações. Em particular, vazamentos de memória devem ser eliminados, sobretudo em tempo de execução, uma vez que se assume que o usuário possa usar o sistema durante longos períodos de tempo.

Disponibilidade: Devido à sua natureza orientada a eventos, o programa deverá estar sempre pronto para responder à entrada do usuário. Além disso, o processo de iniciar e terminar a execução de um *layout* customizado não deve requerer interrupções nas outras atividades do usuário (ex. reinicialização do computador), e deve poder acontecer de maneira rápida e a qualquer momento.

Código aberto: Dado que as funcionalidades principais do programa incluem a interceptação, processamento e modificação de entrada do usuário, é esperado que muitos usuários se preocupem com o funcionamento interno do sistema. Levando isto em conta, o código do programa deverá ser publicado abertamente, sob licença livre.

4.3.3 Regras de Negócio

Esta seção descreve o comportamento típico da entrada de teclado; mais especificamente, são detalhadas as características relevantes sobre teclados e *layouts* que devem ser respeitados durante a implementação dos requisitos.

Uma descrição mais detalhada das regras de negócio está disponível no documento de regras de negócio.

4.3.3.1 Layouts Físicos

- Existem diferentes *layouts* físicos a serem apresentados ao usuário para configuração; estes *layouts* físicos são definidos por diferentes padrões, e alteram a quantidade, o posicionamento e o formato das teclas físicas.

4.3.3.2 Teclas Alfanuméricas

- Toda tecla envia um sinal ao ser pressionado e ao ser solto; quando pressionada durante um longo período de tempo, uma tecla continua enviando o mesmo sinal em um intervalo configurável;
- Teclas alfanuméricas são representadas por um caractere *Unicode*, ou por uma sequência de caracteres Unicode que são enviadas para a janela que tem foco.

4.3.3.3 Teclas Modificadoras

- Uma tecla modificadora altera o estado do *layout* até que a tecla seja solta;
- Combinações de teclas modificadoras também devem contar como possíveis estados de *layout*;
- Teclas modificadoras podem ser compostas por mais de uma tecla física, como a tecla modificadora *Shift*, composta pelos scancodes 2a e 36;
- Em *layouts* que utilizam a tecla modificadora *AltGr*, a tecla física *Alt* direita (scancode e0 38) envia o sinal das teclas modificadoras *Ctrl* esquerda (scancode 1d) e Alt esquerda (scancode 38), em sequência, ao invés do sinal normalmente esperado.

4.3.3.4 Teclas de Acentuação

- Uma tecla de acentuação não envia um caractere, mas coloca o teclado em um estado que altera o caractere da próxima tecla de acordo com regras de substituição pré-determinadas; o estado é revertido logo que a próxima tecla é pressionada;
- O teclado não considera combinações de teclas de acentuação; uma tecla de acentuação pressionada enquanto outra está ativa é considerada um candidato para substituição;
- As teclas *Tab*, *Esc*, e teclas modificadoras não são consideradas pela tecla de acentuação atualmente ativa, e não revertem o estado do teclado;

- Uma tecla de acentuação contém uma representação independente que, geralmente, mas não necessariamente, consiste na representação gráfica do sinal de acentuação (como ‘~’);

- Uma sequência inválida (não correspondente a uma substituição) envia a representação independente da tecla de acentuação, seguida pela representação normal (não substituída) da tecla seguinte, ou a representação independente da tecla seguinte se esta for uma tecla de acentuação.

5 Premissas

O sistema operacional a ser suportado é o *Windows*, a partir da versão *Windows* 7. O suporte para XP e Vista não está previsto, mas poderá ocorrer em um momento futuro; suporte para outros sistemas operacionais não será implementado, devido à alta dependência da API do *Windows*.

O programa será escrito como código aberto. Componentes e bibliotecas externas utilizadas pelo sistema não necessariamente seguem esta premissa, mas prefere-se o uso de componentes livres.

6 Influência das Partes Interessadas

Como o público alvo desta aplicação é bastante amplo, espera-se que haja preocupações com o comportamento interno do programa, dado a necessidade da interceptação da entrada do usuário, e sua potencial utilização como *keylogger* ou aplicação similar. Em conta destas preocupações, considera-se uma vantagem publicar abertamente o código do sistema.

O fato do público alvo ser potencialmente amplo também faz com que a disponibilidade do programa em diversas linguagens seja preferido para o desenvolvimento da interface gráfica.

DOCUMENTO DE REQUISITOS

Multikeys – Aplicação para Remapemento de Teclas em *Windows*

FATEC
Faculdade de Tecnologia de Mogi das Cruzes
Tecnologia em Análise e Desenvolvimento de Sistemas | Tecnologia em Agronegócio | Tecnologia em Recursos Humanos

Histórico de Versões

Data	Versão	Descrição	Autor	Revisor
02/05/2017	1.0	Primeira versão	Rafael Kenji	-
24/09/2017	1.1	Adição de novos requisitos	Rafael Kenji	-
01/10/2017	1.2	Revisão de requisitos	Rafael Kenji	-
28/11/2017	1.3	Versão final	Rafael Kenji	-

Índice

1 Introdução

1.1 Propósito deste Documento

O propósito deste documento de requisitos é descrever os requisitos para o software *Multikeys* para remapeamento de teclados e criação de *layouts* customizados. Estes requisitos incluem uma descrição detalhada acerca de funcionalidades, performance, dados e outras características relevantes do sistema.

1.2 Público Alvo

O público alvo do software *Multikeys* consiste em usuários do sistema operacional *Windows* (versão 7 ou mais recente), sem limitações de localização ou linguagem.

1.3 Escopo do Projeto em Desenvolvimento

O escopo do sistema *Multikeys* para remapeamento de teclados e criação de layouts customizados inclui suas funcionalidades distintas, suas potenciais aplicações e limitações.

Suas funcionalidades são:

- Customização de *layouts* de teclado;
- Remapeamento de teclas em teclados específicos;
- Atribuição de caracteres *Unicode* ou macros a teclas específicas;
- Permitir a execução ou abertura de arquivos em resposta à entrada de teclado;
- Replicar das funcionalidades de um *layout* de teclado oferecido pelo sistema operacional, tais como teclas modificadoras e teclas de acentuação.

As potenciais aplicações do sistema *Multikeys* incluem:

- A utilização de dois ou mais *layouts* de teclado simultaneamente, para tradutores, estudantes de língua estrangeira, e outros usuários que fazem uso frequente de layouts em múltiplos idiomas;
- A criação de um teclado de macros para melhorar a produtividade do usuário em softwares que fazem uso frequente de atalhos de teclado ou sequências destes;
- Usar um teclado dedicado à entrada de caracteres especiais em *Unicode* para linguistas, matemáticos e outros profissionais que fazem uso frequente de caracteres *Unicode* menos comuns.

No entanto, o software *Multikeys* não terá as seguintes funcionalidades em consideração durante o seu desenvolvimento:

- Funcionalidades referentes à automatização de entrada em jogos (comumente chamados de *bots*);
- Replicar ou imitar o comportamento de IMEs (*Input Method Editor*), também chamados de processadores de entrada de texto.

1.4 Definições, Acrônimos e Abreviações

Esta seção define os termos usados ao longo deste documento, a fim de esclarecer quaisquer dúvidas sobre o uso de tais termos.

Termo	Definição
API	*Application Programming Interface*: Interface de programação de aplicação.
C++	Linguagem de programação; o uso do termo neste documento se refere especificamente ao Visual C++ da Microsoft.
C#	Linguagem de programação desenvolvida pela Microsoft.
Unicode	Padrão de codificação de texto capaz de representar conteúdo escrito em numerosos sistemas de escritas no mundo, incluindo caracteres especiais como símbolos matemáticos e Emoji.
Scancode	Sinal vindo do teclado que identifica a tecla pressionada.
WinAPI	Abreviação de Windows API. Refere-se à API dos sistemas operacionais da família Windows NT.
WPF	*Windows Presentation Foundation* – um subsistema gráfico da Microsoft para a interface de usuário baseada em janelas.

1.5 Referências

Este documento complementa e faz referência aos seguintes documentos:
Documento de Visão do Sistema *Multikeys*;
Documento de Regras de Negócio do Sistema *Multikeys*.

1.6 Visão Geral do Documento

O restante deste documento contém as seções Requisitos de Interfaces Externas e Requisitos de *Software*.

A seção Requisitos de Interfaces Externas descrevem as tecnologias externas ao sistema que são necessárias para seu funcionamento.

A seção Requisitos do Sistema descreve, em ordem, os requisitos funcionais da interface gráfica do sistema *Multikeys*, os requisitos funcionais da aplicação em segundo plano do sistema *Multikeys*, as interfaces de *hardware* e *software*, os requisitos de dados, e os requisitos não funcionais do sistema Multikeys.

2 Requisitos de Interfaces Externas

Esta seção contém os recursos externos necessários para o funcionamento do sistema Multikeys.

2.1 Interfaces de *Hardware*

Embora o sistema interaja frequentemente com os teclados do usuário, esta comunicação é feita através da *WinAPI*, e por isso não existe a necessidade do uso de dispositivos de entrada específicos.

2.2 Interfaces de *Software*

O sistema faz uso direto das funções da *WinAPI*, requerendo por isso o sistema operacional *Windows*. Devido às funcionalidades específicas utilizadas pelo sistema, a

versão do sistema operacional requerida para o funcionamento do sistema *Multikeys* é o *Windows* 7 ou superior.

Como a comunicação com dispositivos de entrada é realizada através da *WinAPI*, não existem requisitos de *drivers* ou outros programas específicos para o funcionamento do software *Multikeys*.

A fim de minimizar as dependências de software, o subsistema em C++ deverá ser compilado com as bibliotecas redistributáveis VC++ estaticamente, para que não seja necessário que estes pacotes existam na máquina do usuário.

Acerca do subsistema *Multikeys Editor*, a interface gráfica do sistema *Multikeys* será implementada em .NET; a versão 4.5 da plataforma .NET será necessária para o funcionamento do programa. Embora um instalador seja fornecido como um dos entregáveis do projeto, o sistema também deverá funcionar de maneira portável, sem alterar a máquina onde é executado.

3 Requisitos do Sistema

Esta seção contém todos os requisitos funcionais e não funcionais do sistema, assim como descrições de suas funcionalidades.

3.1 Requisitos Funcionais

Esta seção detalha os requisitos funcionais do sistema; os requisitos são divididos entre a interface gráfica e a aplicação em plano de fundo.

Os requisitos referentes à interface gráfica descrevem as funcionalidades de configuração do sistema, para a criação e utilização de *layouts* customizados. Quando estes *layouts* estão em atividade, um processo é aberto no plano de fundo; os requisitos referentes à aplicação são referentes a este processo, que é responsável por interceptar e interpretar a entrada de tecla do usuário.

Os requisitos funcionais listados a seguir estão anotados com as seguintes legendas:

N – Funcionalidade necessária;

D – Funcionalidade desejada;

Grupo 1: Interface Gráfica	
RF1.1 (N)	Visualizar *Layouts*
Detalhes: Visualizar *layouts* customizados, dispostos visualmente de maneira similar ao layout físico de um teclado.	
RF1.2 (N)	Editar *Layouts*
Detalhes: Editar um *layout*, atribuindo a uma tecla física um ou mais caracteres *Unicode*, sequências de pressões de teclas simuladas, ou execuções de arquivos a teclas específicas. A edição de *layouts* também deve possibilitar a definição de teclas modificadoras, *layouts* para cada combinação de modificadores, e teclas de acentuação.	
RF1.3 (N)	Ler e Salvar *Layouts*
Detalhes: O sistema deve ser capaz de criar, abrir, salvar e editar *layouts* customizados armazenados na forma de um arquivo de configuração visíveis ao usuário.	
RF1.4 (N)	Aplicar *Layouts* a Teclados
Detalhes: O usuário deve poder (opcionalmente) associar *layouts* customizados a teclados específicos, podendo ter múltiplos *layouts* associados a teclados diferentes ao mesmo tempo. Um conjunto de teclados customizados deve ser armazenado em um mesmo arquivo de configuração.	
RF1.5 (N)	Iniciar e Interromper Atividade de *Layouts*
Detalhes: O usuário deve poder iniciar e interromper a atividade dos *layouts* customizados em segundo plano; durante a atividade, as teclas serão interpretadas pelo sistema de acordo com os requisitos detalhados no grupo 2: Aplicação.	
RF1.6 (D)	Alterar Idioma
Detalhes: Selecionar o idioma de exibição da interface gráfica.	
RF1.7 (D)	Configurar *Layout* Físico
Detalhes: Alterar o *layout* físico que a interface usa para organizar as teclas disponíveis para mapeamento (padrões ABNT-2, ANSI, ISO, JIS, *Dubeolsik* e possivelmente outros).	
RF1.8 (D)	Configurar *Layout* Lógico
Detalhes: Alterar o *layout* lógico que é usado para visualmente identificar as teclas mostradas na tela.	

Grupo 2: Aplicação	
RF2.1 (N)	Identificar Entrada de Teclado
Detalhes: Interceptar e bloquear entrada de teclado baseado em *scancodes*, e realizar a ação mapeada de acordo com o *layout* ativo; a identificação de entrada deve discriminar entre dispositivos por nome ou porta, e permitir comportamento diferenciado dependendo do dispositivo que enviou o sinal.	
RF2.2 (N)	Simular Entrada de Texto
Detalhes: O sistema deve ser capaz de simular a entrada de texto em *Unicode*.	
RF2.3 (N)	Simular Sinais de Tecla
Detalhes: O sistema deve ser capaz de simular sinais de teclas pressionadas, em resposta às ações do usuário.	
RF2.4 (D)	Abrir Arquivos Executáveis
Detalhes: O sistema deve ser capaz de procurar e executar um arquivo na máquina a partir do nome do arquivo definido no *layout* customizado. Nota-se que executáveis que precisam de permissão de administrador devem pedir por permissão de acordo.	
RF2.5 (N)	Suportar Teclas Modificadoras
Detalhes: Replicar o comportamento usual de teclas modificadoras; referir-se às regras de negócio 3.1 a 3.4.	
RF2.6 (N)	Suportar Teclas de Acentuação
Detalhes: Replicar o comportamento usual de uma tecla de acentuação; referir-se às regras de negócio 4.1 a 4.6.	
RF2.7 (D)	Suportar Teclas de Fixação
Detalhes: Suportar a customização de teclas com comportamento similar ás teclas *Capslock* e *Numlock*.	

3.2 Requisitos Não-Funcionais

Os requisitos não funcionais listados a seguir se referem tanto à interface gráfica quanto ao processo em plano de fundo, exceto quando explicitamente anotado.

RNF001	**Requisitos de *Hardware***
Descrição: O sistema não deve ter requisitos de *hardware* significantes, além daqueles relacionados ao sistema operacional (referir ao requisito RNF002).	
Justificativa: Requisitos de *hardware* adicionais devem ser evitados para não limitar o público alvo em potencial. Além disso, a preocupação com a performance do programa, como detalhado também no requisito RNF3, torna desejável que o sistema não precise de muitos recursos de *hardware* para funcionar.	

RNF002	**Requisitos de *Software***
Descrição: O sistema *Multikeys* deve funcionar no sistema operacional Windows, versões 7, 8.1 e 10.	
Justificativa: As versões mencionadas são usadas por uma grande porção dos usuários do sistema operacional Windows.	
RNF003	**Tempo de Resposta (plano de fundo)**
Descrição: O tempo de resposta do programa executando em segundo plano deve ser curta o suficiente para o usuário não o perceba. Nota-se que atrasos são inevitáveis no caso de uma aplicação definida pelo usuário ser mapeada a uma tecla.	
Justificativa: O programa executando em segundo plano deverá responder à entrada do usuário em tempo real, e atrasos perceptíveis na resposta do programa são indesejáveis para não interromper o *workflow* do usuário.	
RNF004	**Robustez (plano de fundo)**
Descrição: Erros de execução devem ser tratados de maneira que não causem efeitos colaterais, como mensagens de erro em janela *popup*. Dada qualquer falha, a retomada de atividades deve ocorrer o mais rápido possível, de preferência sem alertar o usuário.	
Justificativa: No caso de erro, é indesejável que a atividade seja interrompida. Alertar o usuário sobre erros recuperáveis interromperia seu fluxo normal de atividades, o que não é desejável quando o programa está executando em plano de fundo.	
RNF005	**Uso de Recursos (plano de fundo)**
Descrição: O sistema nunca deve bloquear o uso de recursos do computador pelo usuário, sistema operacional ou outras aplicações, salvo o uso do teclado de acordo com o funcionamento normal do sistema *Multikeys*. Um caso particular é o vazamento de memória, que bloqueia o uso de memória por outras aplicações, e deve ser particularmente evitado, pois o sistema deverá continuar executando durante longos períodos de tempo.	
Justificativa: Como o programa ficará executando no plano de fundo, este não deverá interromper o fluxo normal de atividades fora do sistema, além de suas funcionalidades previstas.	

RNF006	**Disponibilidade (plano de fundo)**
Descrição: Durante sua execução, o programa deve estar sempre pronto para responder à entrada do usuário. Além disso, o processo de iniciar e terminar a execução de um *layout* customizado não deve requerer interrupções nas outras atividades do usuário, como a reinicialização do computador, e deve poder acontecer de maneira rápida e a qualquer momento.	
Justificativa: O software possui uma natureza orientada a eventos, uma vez que durante sua execução, processamento acontece em resposta ao recebimento de mensagens ao invés de chamadas a funções. É necessário que o sistema consiga corretamente responder a uma mensagem em qualquer situação.	
RNF007	**Código Aberto**
Descrição: O código do programa deverá ser publicado abertamente, sob licença livre.	
Justificativa: As funcionalidades do programa incluem a interceptação, processamento e modificação de entrada do usuário. Por isso, é esperado que muitos usuários se preocupem com o funcionamento interno do sistema.	
RNF008	**Internacionalização (GUI)**
Descrição: A interface gráfica deve estar disponível ao menos nas línguas Inglesa e Portuguesa, e de preferência em outros idiomas. A arquitetura da interface gráfica deve ser tal que permita a fácil adição de outros idiomas ao sistema.	
Justificativa: Dado o amplo público alvo em potencial do sistema Multikeys, o sistema deve estar disponível em várias linguagens para alcançar uma quantidade maior de usuários.	
RNF009	**Facilidade de Uso (GUI)**
Descrição: A interface gráfica deve ser razoavelmente fácil de se usar, com uma interface intuitiva e responsiva. O uso do sistema não deve requerer conhecimento prévio em qualquer outro sistema (salvo o sistema operacional) ou linguagem de programação.	
Justificativa: Embora *softwares* similares já existam, poucos apresentam tanto funcionalidades comparáveis às descritas neste documento, quanto uma interface fácil e intuitiva de se usar, que não requeira conhecimento prévio em alguma área específica.	

RNF010	**Arquivos de Configuração**
Descrição: O usuário deve poder salvar seus layouts customizados em arquivos de configuração. Estes arquivos de configuração devem estar disponíveis para o usuário manipular e mover para outro computador.	
Justificativa: A facilidade de uso do sistema inclui a facilidade de manipular os arquivos de configuração; isto também permitirá diferentes usuários compartilharem seus layouts customizados.	
RNF011	**Execução Portável**
Descrição: A execução do programa deve poder ocorrer sem a necessidade de uma instalação. *Layouts* não devem precisar ser instalados na máquina do usuário, e arquivos de configuração não devem depender de quaisquer outros arquivos.	
Justificativa: Usuários que desejam utilizar seus próprios layouts em outros computadores não devem precisar instalar o programa em outras máquinas.	

DOCUMENTO DE REGRAS DE NEGÓCIO

Multikeys – Aplicação para Remapemento de Teclas em *Windows*

FATEC
Faculdade de Tecnologia de Mogi das Cruzes
Tecnologia em Análise e Desenvolvimento de Sistemas | Tecnologia em Agronegócio | Tecnologia em Recursos Humanos

Histórico de Versões

Data	Versão	Descrição	Autor	Revisor
09/05/2017	0.1	Primeira versão, em desenv.	Rafael Kenji	-
27/05/2017	0.2	Segunda versão, em desenv.	Rafael Kenji	-
24/09/2017	1.0	Versão completa.	Rafael Kenji	-
01/10/2017	1.1	Revisão de regras	Rafael Kenji	-
25/11/2017	1.2	Versão final	Rafael Kenji	-

Índice

1 Introdução

1.1 Propósito deste Documento

Este documento de regras de negócio detalha as regras de negócio do sistema Multikeys, referentes a restrições relacionadas ao funcionamento de *layouts* de teclado no sistema operacional *Windows*.

1.2 Referências

Este documento complementa e faz referência aos seguintes documentos:
- Documento de Visão de Projeto;
- Documento de Requisitos.

2 Regras de Negócio

Esta seção detalha as regras e restrições de negócio, que são relacionadas ao funcionamento usual de teclados e que devem ser levados em consideração durante o desenvolvimento do software. Ficam detalhados os requisitos resultantes destas regras para descrever o que o sistema fará para conformar a estas regras.

As regras de negócio listadas nas seções 2.2, 2.3 e 2.4 são referentes ao *workflow* usual da utilização de um teclado. A fim de replicar o comportamento de um do sistema operacional, o comportamento destes deve ser compreendido.

2.1.1 Layouts de Teclado

RN1.1	**Padrões de *Layout* Físico**
Descrição: Existem vários possíveis *layouts* físicos de teclado, que afetam o posicionamento físico das teclas no teclado. Adicionalmente, *layouts* físicos diferentes podem conter quantidades diferentes de teclas, em posições diferentes no teclado, ou possuir formatos e tamanhos diferentes para alguma tecla.	
Requisito de negócio: A interface gráfica deve oferecer a possibilidade de configurar o posicionamento das teclas na tela de edição de *layout*, a fim de corresponder com o *layout* físico do dispositivo do usuário.	

RN1.2	**Padrões de *Layout* Lógico**
Descrição: Um *layout* lógico descreve o mapeamento entre *scancodes* (sinais físicos enviados por cada tecla) e teclas virtuais, que são correspondentes à função de cada tecla; diferentes *layouts* lógicos são utilizados de acordo com a localização do sistema operacional. Os caracteres impressos sobre um dispositivo de teclado correspondem a um *layout* lógico.	
Requisito de negócio: Durante a edição de *layout*, a visualização de um teclado deve anotar cada tecla com um *layout* lógico da escolha do usuário.	

2.2 Teclas Alfanuméricas

Uma tecla alfanumérica é aquela correspondente a um ou mais caracteres; teclas alfanuméricas compõem a maior parte das teclas de um teclado, e não incluem as teclas de função, como as teclas F1 - F12.

RN2.1	***Scancodes***
Descrição: Toda tecla envia um sinal (*scancode*) ao ser pressionado e ao ser solto; quando pressionada durante um longo período de tempo, uma tecla continua enviando o mesmo sinal em um intervalo determinado pelas configurações do sistema operacional.	
Requisito de negócio: O sistema deve ser capaz de detectar sinais de teclado na forma de *scancodes* (ver requisito funcional RF2.1).	
RN2.2	**Representação *Unicode***
Descrição: Teclas alfanuméricas são representadas por um caractere *Unicode*, ou por uma sequência de caracteres *Unicode* que são enviadas para a janela que tem foco. Sistemas operacionais da família *Windows* NT utilizam a codificação UTF-16.	
Requisito de negócio: O sistema deve ser capaz de simular a entrada de caracteres *Unicode*, codificados em UTF-16 (ver requisito funcional RF2.2).	

2.3 Teclas Modificadoras

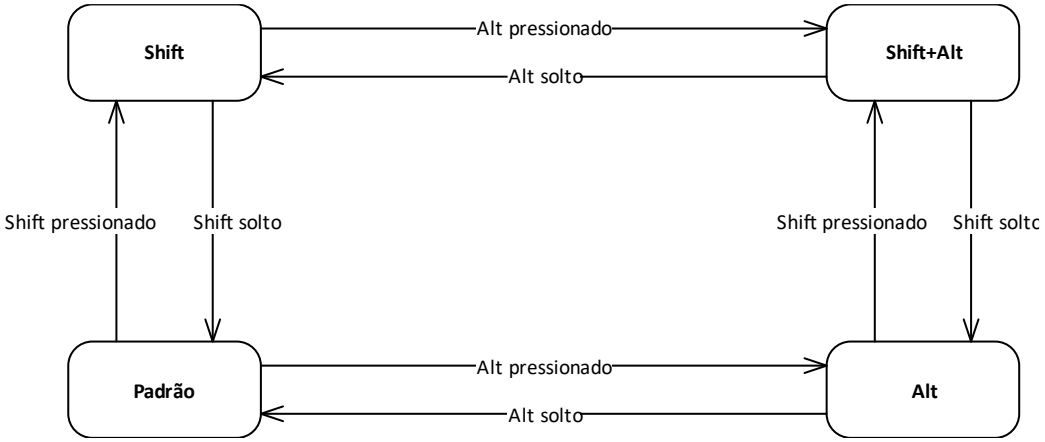
Uma tecla modificadora é uma estratégia para aumentar a quantidade de caracteres ou ações disponíveis no dispositivo de entrada do usuário sem a necessidade de aumentar a quantidade de teclas físicas. *Layouts* de teclado diferentes usam teclas modificadoras diferentes; a tecla *AltGr*, por exemplo, não existe em todos os *layouts*.

RN3.1	**Estado de Tecla Modificadora**
Descrição: Uma tecla modificadora altera o estado do layout até que a tecla seja solta. Combinações de teclas modificadoras também correspondem a estados de tecla modificadora (por exemplo, *Shift* + *Alt*).	
Requisito de negócio: O sistema deve suportar a edição de vários mapas de *scancodes* para as várias combinações de modificadores. Por exemplo, o mapeamento de teclas para o estado *Shift* + *AltGr* deve ser mantido separadamente do mapeamento de teclas para o estado *Shift*.	
RN3.2	**Composição da Tecla Modificadora**
Descrição: Teclas modificadoras podem ser compostas por mais de uma tecla física, como a tecla modificadora *Shift*, composta pelas teclas físicas de *scancodes* 2a e 36. Neste caso, somente uma das teclas físicas precisa estar pressionada para ativar (e manter) o estado de tecla modificadora.	
Requisito de negócio: A representação interna de uma tecla modificadora deve pode conter mais de um *scancode*.	
RN3.3	**Tecla Modificadora *AltGr***
Descrição: Em *layouts* que utilizam a tecla modificadora *AltGr*, a tecla física *Alt* direita (*scancode* e0 38) envia os sinais das teclas *Ctrl* esquerda (*scancode* 1d) e *Alt* esquerda (*scancode* 38) em sequência, ao invés do sinal normalmente esperado. A combinação *Ctrl* + *Alt* tem o mesmo efeito que a tecla *AltGr*.	
Requisito de negócio: O sistema deve ser capaz de corretamente identificar uma tecla *AltGr*, mesmo que esta mande sinais inesperados, e trata-la como a tecla *Alt* direita. Possivelmente inclui saber o *layout* ativo.	
RN3.4	**Tecla Modificadora Fn**
Descrição: A tecla fn ("*function*") está presente em numerosos teclados de tamanho reduzido, geralmente para o acesso das teclas do teclado numérico e funcionalidades adicionais como teclas de mídia. No entanto, a tecla fn é implementada no nível físico, alterando o sinal gerado pelas outras teclas do dispositivo. Isto significa que o sistema operacional não recebe sinais da tecla fn.	
Requisito de negócio: Como a tecla fn é essencialmente invisível ao sistema operacional, não é possível seu mapeamento em qualquer teclado.	

RN3.5	**Independência de Mapas de *Scancodes***
Descrição: Embora o estado de uma tecla modificadora determine qual mapa de *scancodes* é usado, a mudança de estado não altera as teclas modificadoras do teclado. Isto é, teclas como *Shift* e *AltGr* nunca não alteradas devido ao estado de outros modificadores.	
Requisito de negócio: As teclas registradas como modificadores não devem depender do mapa de *scancodes* sendo usado.	

O diagrama de estados na figura 1 ilustra um *layout* com duas teclas modificadoras. Embora os *layouts* de teclado presentes no sistema operacional *Windows* permitam apenas os modificadores *Shift*, *AltGr* e *Ctrl* para a entrada de caracteres, é teoricamente possível permitir um número ilimitado de estados de modificadores.

Figura 1: Diagrama de estados de teclas modificadoras.



Cada estado de modificadores possui um mapa de teclas físicas (*scancodes*) para caracteres; o estado *Shift*, por exemplo, contém caracteres de letras maiúsculas na maioria dos layouts.

Um teclado que usa mais de uma tecla modificadora não precisa necessariamente utilizar todo o espaço disponível; o teclado canadense multilíngue, por exemplo, não possui um mapa de caracteres para o estado *Shift+AltGr*; neste caso, as teclas não têm efeito.

2.4 Teclas de Acentuação

Teclas de acentuação (*dead keys*) possuem uma finalidade similar às teclas modificadoras, a de expandir a quantidade de caracteres disponíveis ao usuário sem

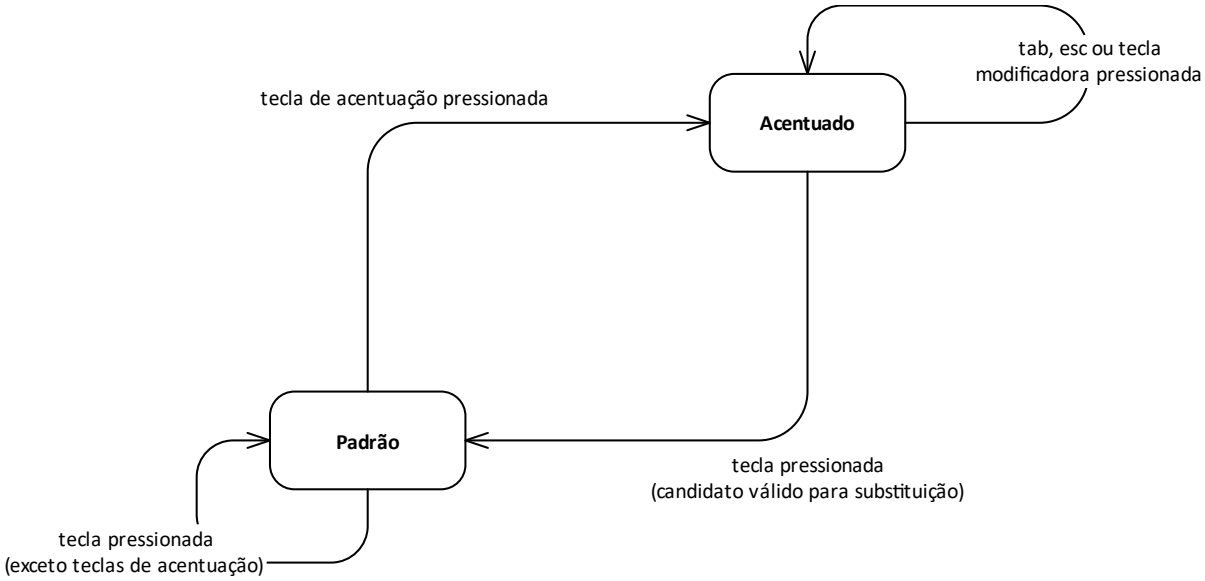
utilizar teclas físicas adicionais. Teclas de acentuação modificam o próximo caractere a ser inserido, colocando o teclado em um estado de tecla de acentuação; por exemplo, a tecla de acentuação '~', quando pressionada, altera o teclado para o estado de tecla de acentuação, causando uma tecla 'a' pressionada a ser substituída pelo caractere 'ã'.

RN4.1	**Estado de Tecla de Acentuação**
Descrição: Uma tecla de acentuação não envia um caractere, mas coloca o teclado em um estado que altera o caractere da próxima tecla de acordo com regras de substituição pré-determinadas; o estado é revertido logo que a próxima tecla é pressionada.	
Requisito de Negócio: O sistema deve suportar a definição de uma tecla de acentuação contendo regras de substituição; além disso, a representação interna do teclado customizado deve suportar os estados	
RN4.2	**Combinações de Teclas de Acentuação**
Descrição: O teclado não considera combinações de teclas de acentuação; uma tecla de acentuação pressionada enquanto outra está ativa é considerada como um candidato para substituição.	
RN4.3	**Teclas Invisíveis para Teclas de Acentuação**
Descrição: As teclas *Tab*, *Esc*, e teclas modificadoras não são consideradas pela tecla de acentuação atualmente ativa como candidatos para substituição, e não revertem o estado do teclado.	
RN4.4	**Representação *Unicode* de uma Tecla de Acentuação**
Descrição: Uma tecla de acentuação contém uma representação independente que, geralmente, mas não necessariamente, consiste na representação gráfica do sinal de acentuação (como '~').	
RN4.5	**Sequência Inválida de Tecla de Acentuação**
Descrição: Uma sequência inválida (não correspondente a uma substituição) envia a representação independente da tecla de acentuação, seguida pela representação normal (não substituída) da tecla seguinte. Se a segunda tecla pressionada for uma tecla de acentuação, sua representação *Unicode* é considerada.	
RN4.6	**Teclas de Acentuação e Estados de Modificadores**
Descrição: O estado de acentuação persiste entre estados de modificadores. Isto é, uma tecla modificadora pressionada enquanto o teclado está em um determinado estado de modificadores continuará válido em outros estados de modificadores.	

O diagrama de estados na figura 2 ilustra o processo de teclas de acentuação. Diferentes dos estados de teclas modificadoras, estes estados não alteram o mapa de

scancodes do *layout*, mas no estado acentuado qualquer tecla pressionada é avaliada juntamente com a tecla modificadora que causou a mudança de estado, possivelmente resultando no envio de um caractere substituto.

Figura 2: **Diagrama de estados de teclas de acentuação.**



2.5 Teclas de Função

Esta seção detalha o funcionamento de teclas não-alfanuméricas, como F1-F12, *Delete*, *Pause*/*Break* e outros.

RN5.1	***Pause*/*Break***
Descrição: A tecla *pause*/*break* gera duas mensagens *RawInput* (*scancodes* 0xe1 0x1d e 0x45) e apenas uma mensagem detectável por um *hook* de teclado (*scancode* 0x45). O *scancode* desta tecla é 0xe1 0x1d 0x45, a única tecla que envia três bytes.	
Requisito de Negócio: O sistema deve identificar este padrão para conseguir corretamente detectar a entrada da tecla *pause*/*break*.	
RN5.2	***Ctrl* + *Pause*/*Break***
Descrição: A tecla *pause*/*break* possui o comportamento peculiar de enviar uma sequência de *scancodes* diferente quando pressionada em combinação com a tecla *Ctrl*. Neste caso, o scancode enviado é 0xe0 0x46.	
Requisito de Negócio: O sistema deve levar em consideração que a tecla *pause*/*break* possui outro *scancode* quando usado em combinação com *Ctrl*.	

RN5.3	***PrintScreen***
Descrição: A tecla *printScreen* pressionada por si só (sem outras teclas pressionadas simultaneamente) produz um *shift* sintético (vide regra de negócio RN5.5 *Shift* Sintético) seguido do *scancode* 0xe0 0x37. Curiosamente, a combinação *Shift* + *PrintScreen* omite o *shift* sintético. A tecla *PrintScreen* possui outra particularidade problemática, a de somente gerar a mensagem de soltura de tecla (*key break*), não acompanhada da pressão de tecla (*key make*).	
Requisito de Negócio: As particularidades da tecla *PrintScreen* devem ser levadas em consideração ao avaliar as mensagens de tecla, incluindo a falta de uma mensagem de pressão de tecla (*key make*).	
RN5.4	***Alt + PrintScreen***
Descrição: A combinação *Alt* + *PrintScreen* corresponde à tecla *System Requirement*, uma tecla comumente omitida dos teclados e que não possui uso padrão. Seu *scancode* é 0x54.	
Requisito de Negócio: O sistema deve levar em consideração que a tecla *PrintScreen* possui outro *scancode* quando usado em combinação com *Alt*.	
RN5.*5*	***Shift* Sintético**
Descrição: Termo original *fake shift* (“shift falso”). Refere-se ao *scancode* 0xe0 0x2a, sintetizado pelo driver em certas situações para manipular o estado do modificador *Shift* de certas maneiras.	
Requisito de Negócio: O sistema deve identificar e corrigir quaisquer situações em que um *shift* sintético poderia vir a alterar o resultado esperado. Cuidado deve ser tomado com as teclas que costumam enviar o sinal do shift sintético.	
RN5.6	***Eisuu* / Alfanumérico**
Descrição: A tecla *Eisuu* (英数), presente no layout JIS, ocupa o mesmo espaço da tecla *CapsLock*. Seu uso enquanto o layout de teclado japonês está ativo envia uma mensagem diferente para o *hook* de teclado.	
Requisito de Negócio: O sistema deve reconhecer que a mensagem alterada corresponde à mesma tecla física que o *CapsLock*.	

RN5.7	**Teclas Coreanas de Linguagem**
Descrição: As teclas *Hanja* (한자, *scancode* = 0xf1) e *Han*/*Yeong*(한/영,*scancode*=0xf2), presentes no layout *Dubeolsik*, enviam somente um *scancode* quando são pressionadas, mas não ao serem soltas. Estas teclas não repetem *scancodes* quando são continuamente pressionadas.	
Requisito de Negócio: O sistema deve reconhecer que os *scancodes* 0xf1 e 0xf2 não enviam sinais ao serem soltos.	

DOCUMENTO DE ARQUITETURA

Multikeys – Aplicação para Remapemento de Teclas em *Windows*

FATEC
Faculdade de Tecnologia de Mogi das Cruzes
Tecnologia em Análise e Desenvolvimento de Sistemas | Tecnologia em Agronegócio | Tecnologia em Recursos Humanos

Histórico de Versões

Data	Versão	Descrição	Autor	Revisor
09/05/2017	0.1	Versão preliminar	Rafael Kenji	-
27/05/2017	1.0	Primeira versão	Rafael Kenji	-
24/09/2017	1.1	Atualização	Rafael Kenji	-
01/10/2017	1.2	Atualização na visão de dados	Rafael Kenji	-
08/10/2017	1.3	Atualização da visão lógica	Rafael Kenji	-
25/11/2017	1.4	Versão final	Rafael Kenji	-

Índice

1 Introdução

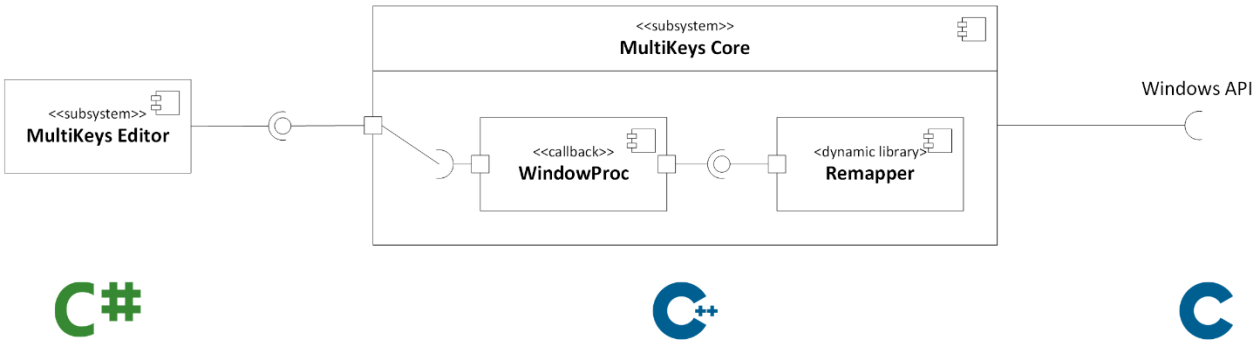
1.1 Propósito Deste Documento

Este documento descreve a arquitetura do sistema *Multikeys*, provendo um número de visões arquiteturais diferentes para ilustrar diferentes aspectos do sistema. Este documento apresenta e justifica as decisões arquiteturais significativas realizadas durante o desenvolvimento do sistema.

2 Representação Arquitetural

O sistema será desenvolvido de acordo com a arquitetura apresentada na Figura 1. Este diagrama separa o projeto em subsistemas, os quais agrupam e demonstram os aspectos do sistema relevantes ao modelo.

Figura 1: Diagrama de componentes dos subsistemas



A arquitetura apresentada provê a separação entre interface gráfica e o núcleo da aplicação como subsistemas. A comunicação entre estes será mínima, através de arquivos de configuração e mensagens do sistema; além disso, somente o núcleo da aplicação fará uso direto da API do *Windows*.

A separação em subsistemas é justificada pelas necessidades diferentes entre a interface gráfica e o programa que realiza os remapeamentos; a interface gráfica existe para prover um uso fácil do sistema, eliminando a necessidade do usuário possuir muito conhecimento específico para sua utilização e configuração.

Esta separação é baseada no conceito de camadas de software, em que uma determinada camada utiliza comente os recursos das camadas abaixo. Neste caso, a

interface gráfica pode iniciar e finalizar a execução do núcleo da aplicação, entre outras atividades. O núcleo da aplicação, por sua vez, utiliza os recursos do sistema operacional através de chamadas de funções da API do *Windows*, sendo também independente de qualquer interface gráfica. Por último, o sistema operacional é obviamente independente do sistema *Multikeys*.

A aplicação principal é dividida em três partes: o processo de janela, o modelo de negócios, e o *hook*, responsáveis por interceptar mensagens de teclado.

A interface gráfica possui a camada de apresentação, a camada de lógica de domínio, e o modelo, consistindo em entidades de domínio.

Para maximizar a modularidade entre os dois componentes, não há código comum entre a aplicação e sua interface gráfica; ao invés disso, os dois componentes realizarão a comunicação através de um arquivo de configuração em xml, que conterá toda a estrutura de um ou mais *layouts* customizados, e também através de mensagens do sistema operacional em tempo de execução.

2.1 Restrições Arquiteturais

Como a aplicação principal deverá ser escrita em C++, certos componentes devem ser implementados de acordo. Por exemplo, as interfaces exibidas na figura 4 serão implementadas como ponteiros para classes puramente abstratas, e os componentes representados como pacotes serão implementados como *namespaces*.

Estas implementações são baseadas nos padrões de projeto da Gangue dos Quatro, cuja obra original oferece exemplos e explicações na linguagem C++.

3 Visão de Casos de Uso

Preliminarmente, o caso de uso principal e mais significativo à arquitetura é a resposta da aplicação a uma entrada de tecla do usuário. Esta atividade ocorre em resposta a cada mensagem de teclado interceptada, e somente ocorre enquanto o programa estiver ativo em plano de fundo.

Além deste, casos de uso de configuração são fundamentais para os objetivos deste projeto – a possibilidade de criar, visualizar e realizar operações em *layouts* customizados, assim como aplicar estes layouts a teclas específicos conectados ao computador.

Descrições detalhadas sobre cada caso de uso, assim como os diagramas de caso de uso, podem ser encontrados no documento de caso de uso deste software. Além

disso, a especificação do caso de uso de interceptação de tecla pode ser encontrado no documento *Especificação de Caso de Uso UC3.1*.

4 Visão de Lógica

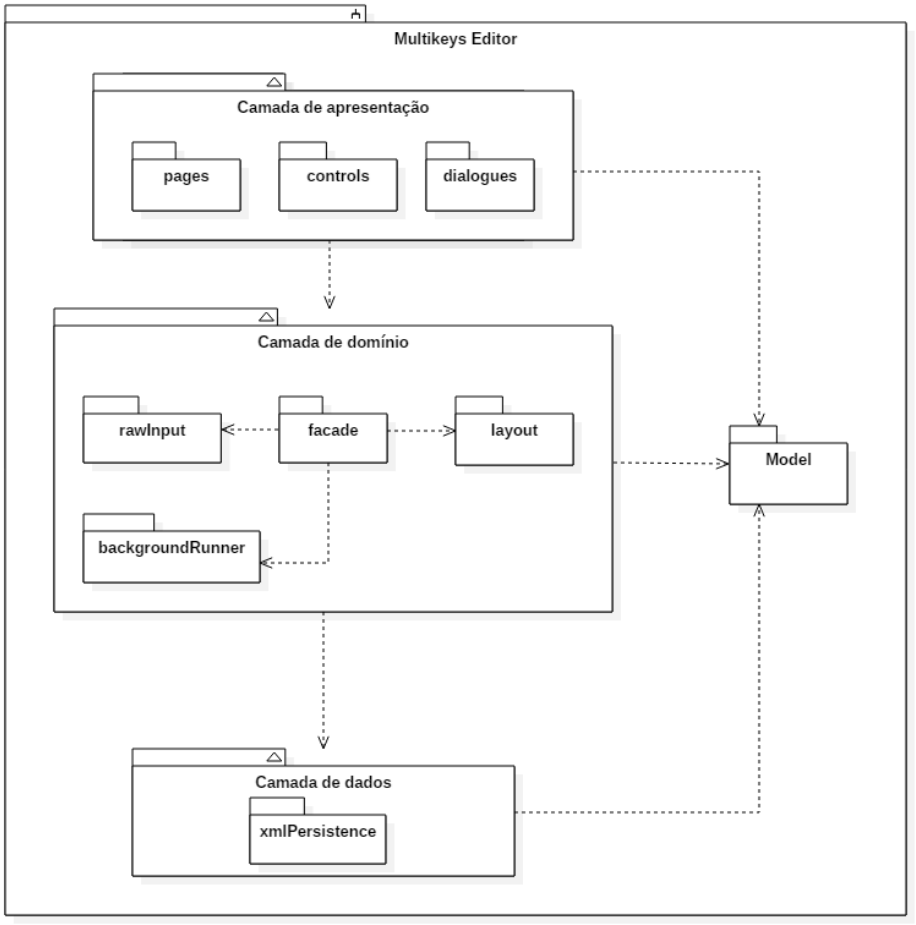
Esta seção detalha os elementos de design da arquitetura do sistema Multikeys, e a organização de seus componentes.

4.1 Interface Gráfica

A interface gráfica é o subsistema do projeto responsável por prover a interação com o usuário final através de janelas no desktop, permitindo a edição de *layouts* customizados. Considerando que o programa será escrito para a plataforma *Windows*, a interface gráfica usará a plataforma .NET, na linguagem C#, e usando a tecnologia WPF (*Windows Presentation Foundation*) para a separação de interface e lógica.

Embora a interface gráfica por si só funcione como a camada de visualização do sistema, este subsistema é ainda dividido nas camadas de apresentação, de domínio e de dados. O motivo para isto é a necessidade de manipular os *layouts* personalizados do usuário na forma de arquivos xml; o modelo de negócios presente neste subsistema existe somente para modelar os dados de um layout, e não possui comportamentos. Adicionalmente, várias funcionalidades e regras de negócio do programa (como padrões de *layouts* físicos e detecção do nome de um teclado) são agrupadas na camada de domínio.

Figura 2: Diagrama de modelo da interface



4.1.1 Camada de Apresentação

Esta camada contém os componentes gráficos, consistindo em janelas XAML, assim como suas classes controladoras que oferecem a lógica de apresentação. Também estarão contidas nessa camada os componentes gráficos customizados em WPF, chamados de *user controls*, e janelas de diálogo customizadas.

4.1.2 Camada de Domínio

Nesta camada ficarão os objetos de domínio, representando a estrutura dos mapeamentos de teclas.

O pacote chamado *backgroundRunner* é responsável pela comunicação com a aplicação principal, assim como pelo gerenciamento de seu ciclo de vida. Uma fachada é fornecida para expor as funcionalidades da camada de negócios à camada de apresentação.

Devido aos requisitos **RF1.7 – Configurar *Layout* Físico** e **RF1.8 – Configurar *Layout* Lógico**, existe um pacode chamado *layout*, responsável pelas informações de diferentes padrões de *layouts* físicos e lógicos (por exemplo, quais *scancodes* existem no padrão físico ABNT-2).

4.1.3 Camada de Dados

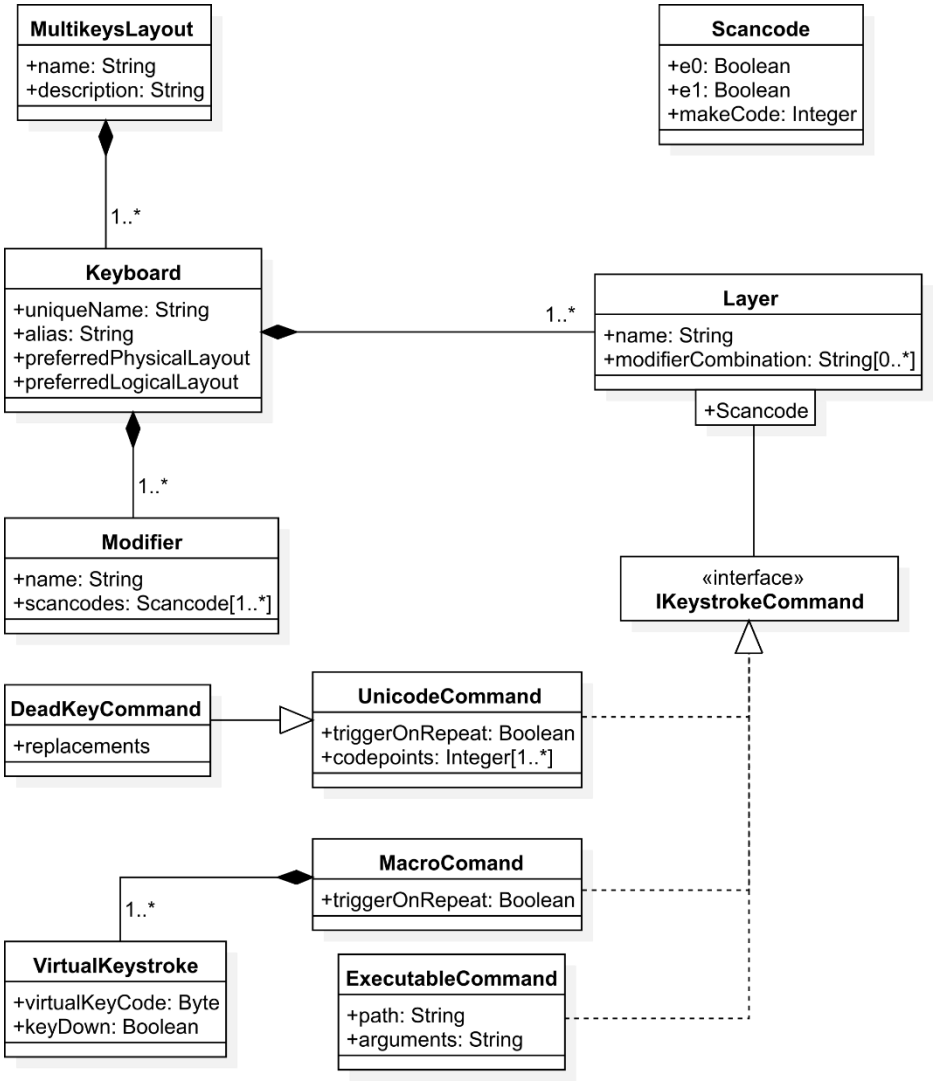
A camada de dados é responsável por ler e escrever arquivos de configuração xml em disco. A persistência dos dados não é feita em um banco de dados, mas sim em arquivos de configuração para que o componente da aplicação possa ler as configurações e funcionar independentemente da interface.

O formato xml foi escolhido devido à organização hierárquica das estruturas de dados; isto é, vários teclados possuem vários níveis cada, e cada nível possui vários mapeamentos cada.

4.1.4 Modelo

O modelo é mostrado separadamente no diagrama por conter entidades que podem ser utilizadas por qualquer camada do sistema. O modelo é uma representação do conteúdo de um arquivo de configuração *Multikeys*; as entidades não possuem comportamentos e consistem simplesmente em classes de dados.

Figura 3: Entidades do modelo Multikeys Editor



Na Figura 3, o diagrama contém as classes contidas no pacote *model*. Estas classes estão dispostas em uma estrutura de árvore, com um objeto *MultikeysLayout* como a raiz. Isto significa que uma instância da classe *MultikeysLayout* contém toda a informação contida em um arquivo de *layout Multikeys*.

4.2 *Multikeys Core*

O subsistema de aplicação, *Multikeys* Core, é o componente do projeto que executará no plano de fundo, recebendo, processando e modificando mensagens do

sistema referentes à entrada do usuário pelo teclado. Este componente é escrito em C++, devido a frequentes interações com a API do *Windows*, e também por conta dos requisitos de performace e resposividade.

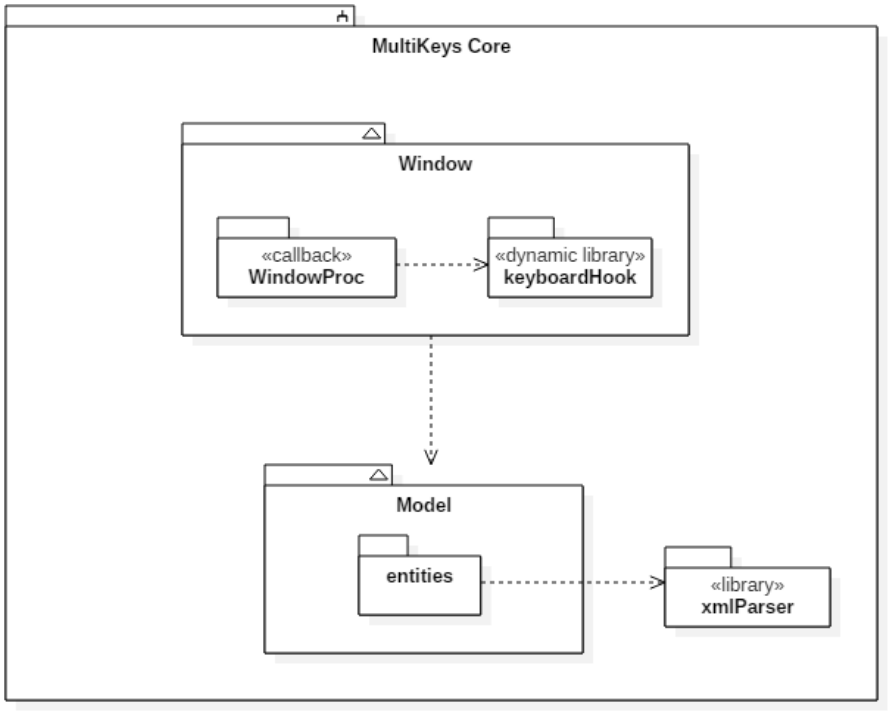
Embora várias das funcionalidades providas por este subsistema possam ser realizadas em alto nível, como através da plataforma .NET, este projeto possui várias necessidades de baixo nível (como a identificação da origem de uma mensagem) que tornam mais apropriado o uso direto da API do *Windows*.

A aplicação terá seu ciclo de vida gerenciado pelo subsistema *Multikeys Editor*, sendo a comunicação entre eles realizada através de mensagens do sistema. Através destas mensagens, o subsistema da aplicação será capaz de carregar mapeamentos a partir de um arquivo de configuração, ou finalizar sua execução. Estes arquivos de configuração também são gerenciados pelo *Multikeys Editor*.

A aplicação terá uma arquitetura movida por eventos, na qual um procedimento de janela (*window procedure*) registra *hooks* através da API do *Windows* para receber mensagens do sistema referentes à entrada de teclado.

É importante notar que em *Windows* existem os *serviços*, programas similares a *daemons* em *Linux* e *agentes* em *MacOS*/*OS X* que executam em plano de fundo, sem interação direta com o usuário. No entanto, a instalação de serviços em uma máquina requer direitos de administrador, enquanto aplicações de janela (mesmo sem janela visível) podem ser executados livremente, oferecendo uma vantagem do ponto de vista da usabilidade.

Figura 4: Diagrama de modelo da aplicação



4.2.1 Window Procedure

A *window procedure*, às vezes chamada de *WindowProc*, é a função de *callback* que processa as mensagens enviadas a uma janela (mesmo que esta não seja visível). É necessário haver um *window procedure* na aplicação para que o programa possa interceptar e processar dados vindos do(s) teclado(s) do usuário.

Dois mecanismos de interceptação são registrados para este procedimento: uma usando a API *Raw Input* provida pelo sistema operacional para a obtenção de dados em baixo nível, e um *keyboard hook*, também provida pela API do sistema, que deve ser instalada em uma biblioteca dinâmica à parte para interceptar mensagens globais. Tal biblioteca dinâmica é representada na figura 3 como o módulo *keyboardHook*.

4.2.2 Camada de Negócios

A camada de negócios fica responsável por manter um modelo interno mutável de cada teclado do usuário, atualizando-os conforme entradas de teclado são recebidas pelo sistema. Este modelo interno também contém todos os mapeamentos do usuário,

carregados do arquivo de configurações, e mantém comandos correspondentes a cada um deles (os quais enviam sinais fabricados ao serem executados).

4.2.3 Xerces

A biblioteca responsável por ler arquivos xml e transformá-los em uma estrutura em árvore será o *Xerces*, desenvolvido pela *Apache*; A aplicação fará uso desta biblioteca para carregar os mapeamentos do arquivo de configuração para o modelo interno.

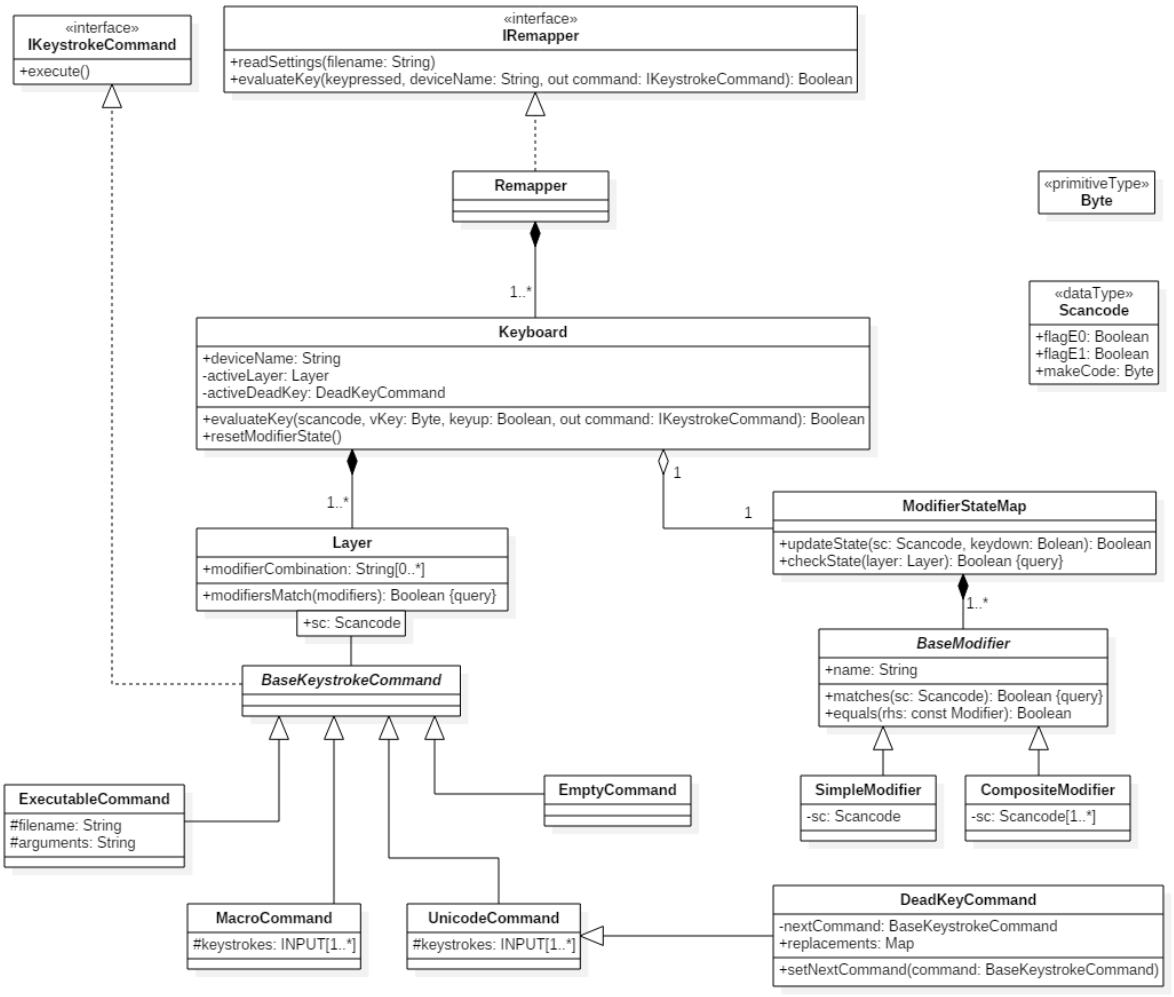
4.2.3.1 Entidades

Este pacote conterá o modelo interno dos teclados do usuário, e expõe suas funcionalidades através de interfaces acessíveis pelo procedimento de janela. Um objeto do tipo *IRemapper* será instanciado e usado pelo *window procedure*; este objeto ficará responsável por receber informações sobre cada tecla pressionada pelo usuário, e retornar um *ICommand* correspondente à ação mapeada.

O encapsulamento do modelo se dará através de funções expostas em um cabeçalho, responsáveis por instanciar e destruir um objeto derivado de *IRemapper* através de ponteiros. Devido à linguagem, interfaces são implementadas com ponteiros para classes puramente abstratas.

A escolha de usar funções para instanciar e destruir uma interface busca facilitar futuras modificações feitas no pacote de entidades.

Figura 5: Diagrama de classe do modelo interno da aplicação

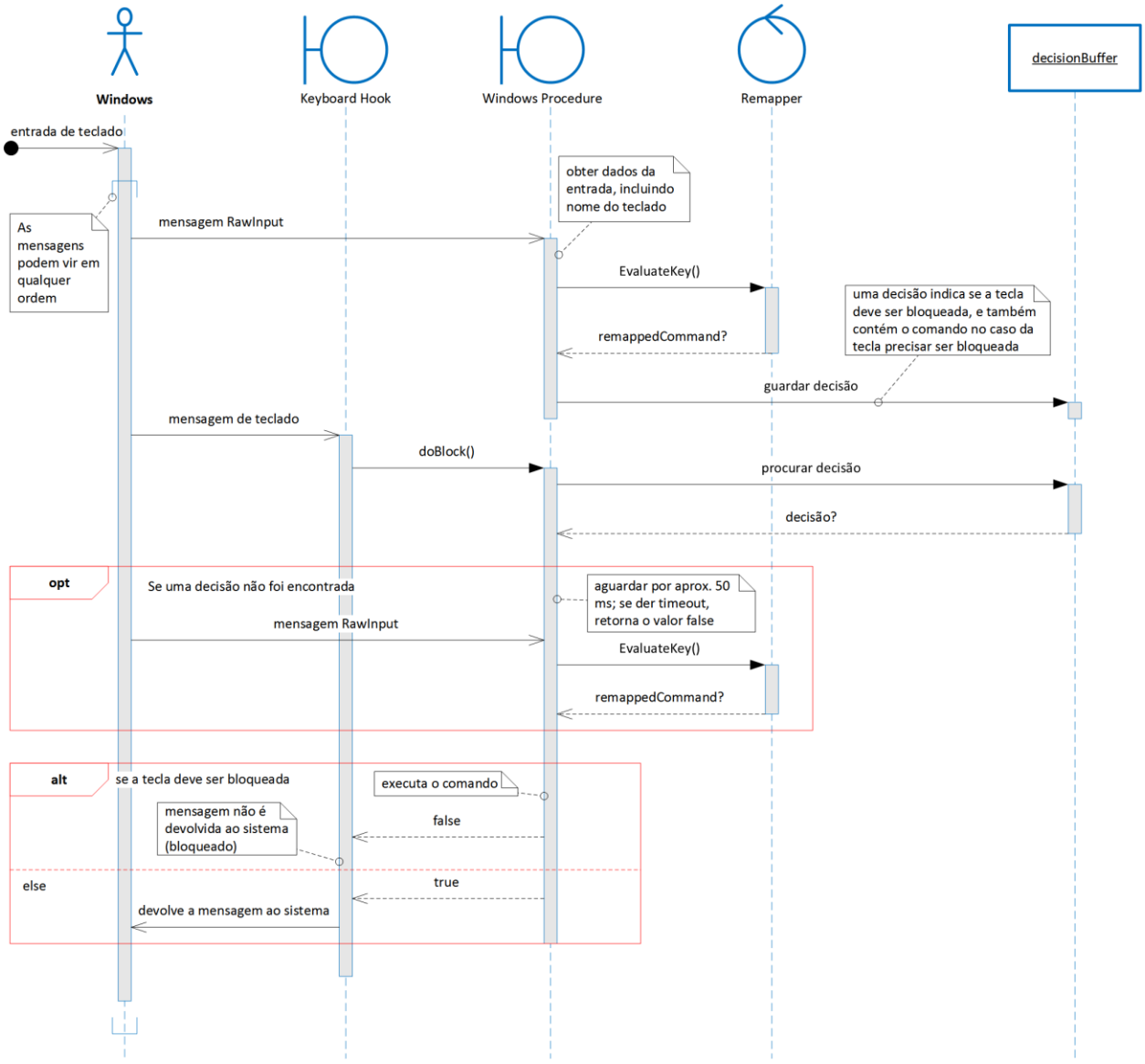


As interfaces *IRemapper* e *IKeystrokeCommand* são expostas e usadas no procedimento de janela. Todos os objetos necessários para o funcionamento da aplicação são instanciados no momento da leitura do arquivo de configuração, que é feito através do método *IRemapper*::*readSettings*().

4.3 Realização de Casos de Uso

A Figura 6 contém um diagrama de sequência que descreve o processamento que ocorre no procedimento de janela do processo em plano de fundo em resposta à entrada de teclado. O processamento ocorre na camada "*Window*" no diagrama de pacotes apresentado na Figura 4.

Figura 6: Diagrama de sequência do módulo *Window* (subsistema *Multikeys Core*)



Neste diagrama, o ponto de interrogação é usado para representar objetos anuláveis (que em C++ são implementados como ponteiros que podem assumir o valor nulo para representar um certo estado). Esta notação não é parte da linguagem UML, mas foi empregada para representar um comportamento relevante à sequência.

O objeto *decisionBuffer* é uma fila contendo decisões referentes ao bloqueio da tecla em questão. O nome da classe é omitido porque os detalhes de sua implementação

não são relevantes para a arquitetura. Cada decisão na fila contém também uma referência para o comando a ser executado.

5 Visão de Implantação

O sistema não trabalhará em rede, e sua instalação se dará através de um arquivo instalador (.msi ou .exe), a fim de facilitar o uso pelo usuário. O sistema poderá ser executado em qualquer computador usando um sistema operacional da família *Windows* NT, versão *Windows* 7, 8.1 ou 10.

Uma funcionalidade desejada para o programa é a possibilidade de se realizar uma instalação portável, que possa ser facilmente utilizada em mais de um computador.

Isto também significa que não há alvo específico para testes; especificações de *hardware* são definidas nos requisitos não funcionais do projeto *Multikeys*.

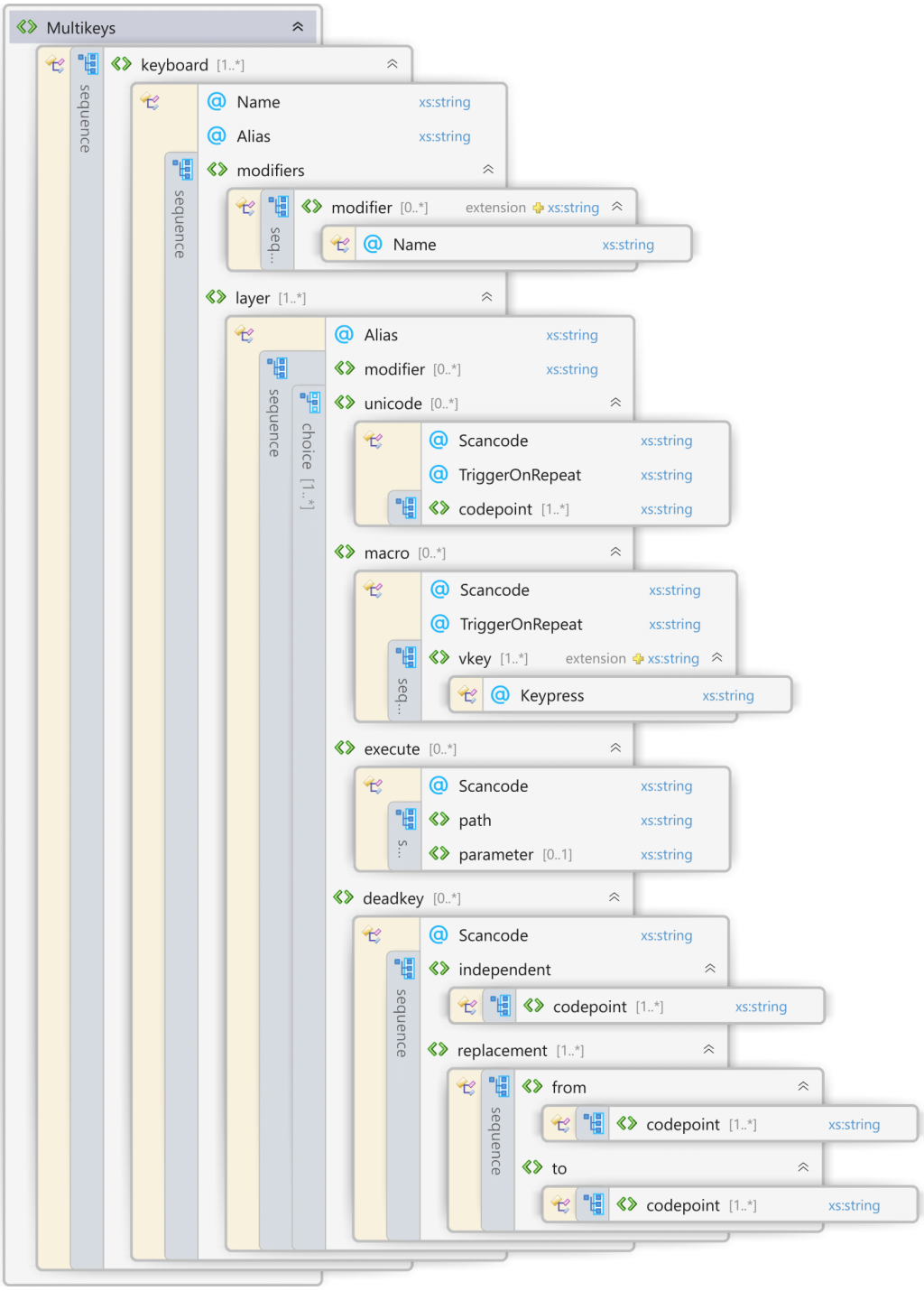
6 Visão de Dados

Os dados a serem persistidos entre usos desta aplicação consistem nos *layouts* customizados. Estes *layouts* contém informações sobre os diferentes modificadores, os níveis do teclado (referentes aos estados de modificadores), ações associadas a cada tecla física, entre outros.

A aplicação *Multikeys* não usará um sistema gerenciador de banco de dados (SGBD). Esta decisão se justifica pelo fato da aplicação não beneficiar das principais vantagens do uso de um SGBD, relacional ou não, como armazenamento de grande volume de dados e confidencialidade. Ao invés disso, serão usados arquivos XML para a representação de cada *layout*, com um esquema em XSD para a definição formal do vocabulário e estrutura dos arquivos.

A Figura 7 contém um modelo de dados do sistema *Multikeys*, gerado no visualizador de XMLs do *Visual Studio*.

Figura 7: Estrutura de dados do arquivo de configuração xml.



7 Tamanho e Performance

De acordo com os requisitos não funcionais do projeto *Multikeys*, é importante que o sistema não apresente atrasos perceptíveis durante a sua utilização em plano de fundo. Na situação ideal, os dados utilizados pelo processo em plano de fundo (consistindo em todos os mapeamentos de um *layout* customizado) devem ser carregados em memória, sem excesso de alocação dinâmica em tempo de execução. O objetivo é que todo o processamento realizado pelo sistema seja sempre rápido o suficiente para que não seja perceptível pelo usuário.

8 Qualidade

A arquitetura apresentada atende aos requisitos listados no documentos de requisitos. A separação em subsistemas significa que cada parte do sistema pode ser desenvolvida com foco nos requisitos relevantes; a interface é desenvolvida com foco na usabilidade, de maneira apropriada para usuários sem treinamento especial, e o módulo do processo em plano de fundo é desenvolvido com foco na performance e resposividade.

DOCUMENTO DE CASOS DE USO

Multikeys – Aplicação para Remapemento de Teclas em *Windows*

FATEC
Faculdade de Tecnologia de Mogi das Cruzes
Tecnologia em Análise e Desenvolvimento de Sistemas | Tecnologia em Agronegócio | Tecnologia em Recursos Humanos

Histórico de Versões

Data	Versão	Descrição	Autor	Revisor
09/05/2017	0.1	Versão preliminar	Rafael Kenji	-
27/05/2017	1.0	Primeira versão	Rafael Kenji	-
21/10/2017	1.1	Finalização dos casos de uso	Rafael Kenji	-
25/11/2017	1.2	Versão final	Rafael Kenji	-

Índice

1 Introdução

1.1 Propósito Deste Documento

Este documento lista de forma resumida todos os casos de uso do sistema *Multikeys* que não estão descritos em seus próprios documentos de Especificação de Caso de Uso.

1.2 Descrição do Sistema

O sistema *Multikeys* para remapeamento de teclas em *Windows* permite ao usuário criar *layouts* personalizados, atribuindo qualquer caractere *Unicode* ou combinação de teclas a cada tecla física de seu teclado. Além disso, o usuário pode definir estes *layouts* distinguindo entre dois ou mais teclados físicos, o que normalmente não é possível no sistema operacional *Windows*, que trata múltiplos dispositivos de teclado da mesma maneira.

2 Casos de Uso

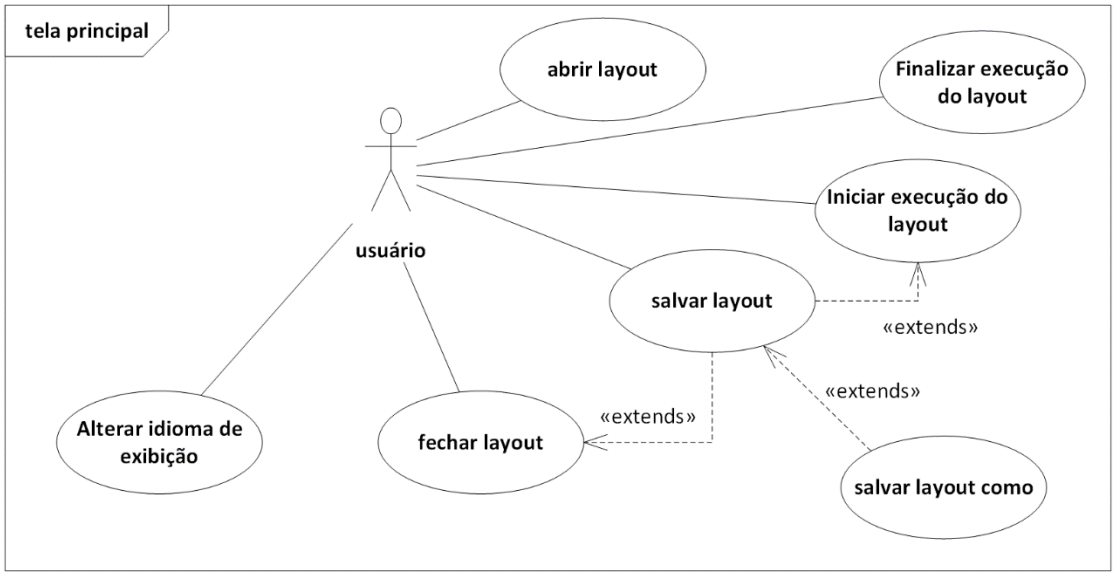
2.1 Subsistema *Multikeys Editor*

Este subsistema é voltado à configuração de *layouts* customizados pelo cliente. É através da interface gráfica que o usuário manipula *layouts* e escolhe os mapeamentos entre teclas físicas (*scancodes*) e ações, sejam elas o envio de um caractere *Unicode*, a simulação de sinais de teclas simulados, ou a execução de um determinado arquivo em disco.

2.2 Tela Principal

A Figura 1 contém um diagrama de casos de uso ilustrando as atividades que o usuário pode tomar a partir da tela inicial do sistema.

Figura 1: Diagrama de casos de uso da tela principal



2.2.1 Abrir Layout

Código: UC1.1

Descrição: Usuário possui um arquivo de *layout* que deseja abrir e visualizar / editar usando o sistema.

Pré-condições: Nenhum

Pós-condições: *Layout* está aberto e pronto para ser editado na tela.

Extensões: Nenhum

Fluxo Principal:

1. Usuário clica no botão para abrir um *layout*.
2. Sistema apresenta uma tela para seleção de arquivo.
3. Usuário navega até o arquivo desejado e o seleciona.
4. Sistema abre e lê o arquivo de configuração.
5. Sistema mostra o conteúdo do arquivo na tela de edição de *layout*.
6. Sistema armazena o nome do arquivo atualmente sendo editado.

Fluxos Alternativos:

1a. Já existe um *layout* aberto e em edição.

1a.1. Sistema informa que as alterações serão perdidas, e pergunta que o usuário deseja continuar.

1a.2. Se o usuário deseja continuar, o fluxo continua para o passo 2; se não, o fluxo é interrompido.

4a. Sistema não consegue abrir o arquivo selecionado.

4a1. Sistema informa ao usuário que o arquivo selecionado é inválido ou está corrompido.

2.2.2 Salvar Layout

Código: UC1.2
Descrição: Usuário deseja salvar as alterações feitas em um arquivo em disco.
Pré-condições: Existe um *layout* aberto.
Pós-condições: *Layout* do usuário foi salvo em disco no local requisitado.
Extensões: UC1.3 Salvar *Layout* Como; UC1.4 Fechar *Layout*
Fluxo Principal:
1. Usuário dá o comando de salvar um *layout* (barra de ferramentas ou atalho).
2. Sistema procura o arquivo em disco, de acordo com o caminho que foi armazenado.

Fluxos Alternativos:
2a. O *layout* ainda não existe em disco (foi criado nesta sessão).

2a.1. O fluxo é enviado para o passo 2 do caso de uso 2.2.3 Salvar Layout Como.

2b. Sistema não consegue salvar o *layout* aberto, possivelmente porque este está aberto em outra aplicação.

2b1. Sistema informa o usuário do erro ocorrido.

2c. O arquivo em disco não foi encontrado, possivelmente porque este não existe mais em disco.

2c.1. O sistema informa o usuário que o arquivo não foi encontrado, e o pergunta se ele deseja salvar um novo *layout*.

2c.2. Se o usuário deseja salvar um novo *layout*, o fluxo é enviado para o passo 2 do caso de uso 2.2.3 Salvar Layout Como.

2.2.3 Salvar Layout Como

Código: UC1.3
Descrição: Usuário deseja salvar um *layout* que ainda não está salvo em disco, ou deseja salvar o *layout* sendo trabalhado em um novo arquivo.
Pré-condições: Existe um *layout* aberto.
Pós-condições: *Layout* do usuário foi salvo em disco no local requisitado.
Extensões: Nenhum

Fluxo Principal:

1. Usuário dá o comando de salvar um *layout* como (barra de ferramentas ou atalho).
2. Sistema apresenta a janela para a escolha do nome e local do arquivo.
3. Usuário escolhe e confirma o nome e local do arquivo.
4. Sistema cria um novo arquivo contendo o *layout* aberto.
5. Sistema passa a usar o novo arquivo como o *layout* aberto.

Fluxos Alternativos:

4a. Já existe um arquivo de mesmo nome no local requisitado.

4a1. Sistema pergunta se o usuário deseja substituir o arquivo existente.

4a2. Usuário responde e o sistema reage de acordo.

2.2.4 Fechar Layout

Código: UC1.4

Descrição: Usuário fecha um *layout* aberto.

Pré-condições: Existe um *layout* aberto.

Pós-condições: *Layout* aberto foi fechado.

Extensões: UC1.2 Salvar *Layout*

Fluxo Principal:

1. Usuário dá o comando de fechar o *layout* (barra de ferramentas ou atalho).
2. Sistema avisa o usuário de que mudanças não salvas serão perdidas, e pede confirmação.
3. Sistema fecha o *layout* sendo editado, e deixa de usar o caminho ao *layout* como o *layout* atualmente aberto.

Fluxos Alternativos:

2a. Usuário não confirma o fechamento do *layout* sendo editado.

2a.1. O fluxo é interrompido.

2.2.5 Inicializar execução do Layout

Código: UC1.5

Descrição: Usuário inicia o processo em plano de fundo (subsistema *Multikeys Core*) para a execução do *layout*.

Pré-condições: O processo em plano de fundo não está sendo executado.

Pós-condições: O processo em plano de fundo é iniciado.

Extensões: UC1.2 Salvar *Layout*.

Fluxo Principal:

1. Usuário dá o comando de iniciar execução do *layout*

2. Sistema inicia o processo em plano de fundo, passando a este o caminho ao *layout* sendo editado.

Fluxos Alternativos:

1a. O *layout* sendo editado possui alterações não salvas, e, portanto, o arquivo em disco não está sincronizado com o conteúdo mostrado na tela.

1a.1. O sistema informa o usuário de que existem alterações não salvas, e pergunta se ele deseja salvar o *layout*.

1a.2. Se o usuário deseja salvar o *layout*, o caso de uso UC1.2. Salvar *Layout* é executado a partir do passo 2, e depois retorna a este passo.

1a.3. O fluxo retorna ao passo 2 do fluxo principal.

2.2.6 Finalizar execução do Layout

Código: UC1.6

Descrição: Usuário finaliza o processo em plano de fundo (subsistema *Multikeys Core*) para a execução do *layout*.

Pré-condições: O processo em plano de fundo está em execução.

Pós-condições: Nenhum

Extensões: Nenhum

Fluxo Principal:

1. Usuário dá o comando de parar execução do *layout*

2. Sistema envia o sinal de finalização para o processo em plano de fundo.

Fluxos Alternativos:

2a. O processo já foi finalizado por alguma outra maneira.

2a.1. O processo é considerado como finalizado, e o fluxo é interrompido.

2b. O processo falhou em finalizar, ou deixou de responder.

2b.1. O processo é finalizado forçadamente, e considerado como finalizado.

2.2.7 Alterar Idioma de Exibição

Código: UC1.7

Descrição: Usuário deseja alterar o idioma de exibição da interface gráfica.

Pré-condições: Nenhum

Pós-condições: A página é recarregada na linguagem escolhida.

Extensões: Nenhum.

Fluxo Principal:

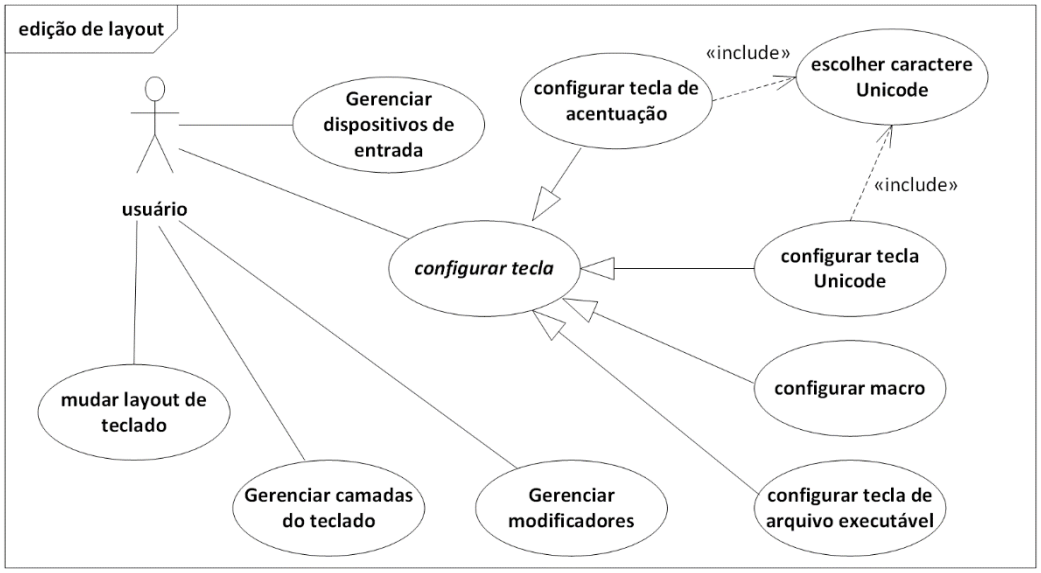
1. Usuário dá o comando de alterar a interface para determinado idioma
2. Sistema apresenta a lista de idiomas suportados.
3. Usuário seleciona o idioma desejado e confirma.
4. Sistema recarrega os controles da tela de acordo com o idioma selecionado, e armazena o novo idioma nas preferências do usuário.

Fluxos Alternativos:

3a. Usuário não deseja mudar o idioma.

3a1. O fluxo é interrompido, e o idioma é mantido o mesmo.

Figura 2: Diagrama de casos de uso de edição de *layout*



2.2.8 Configurar Tecla

Código: UC2.1

Descrição: Caso de uso abstrato; usuário quer associar uma tecla física em um determinado teclado a uma ação.

Pré-condições: *Layout* está aberto.

Pós-condições: A tecla física selecionada foi mapeada a uma ação.

Extensões: Nenhum

Fluxo Principal:

1. Usuário seleciona uma determinada tecla para edição.

Fluxos Alternativos:

2.2.9 Escolher Caractere Unicode

Código: UC2.2.
Descrição: Usuário quer associar uma tecla física a um ou mais símbolos Unicode.
Pré-condições: *Layout* está aberto.
Pós-condições: Tecla física foi mapeada a um ou mais símbolos *Unicode*.
Extensões: Nenhum.
Fluxo Principal:
1. Usuário seleciona uma determinada tecla para edição.
2. **Include**: Caso de uso UC2.6 Escolher Caractere *Unicode*.
3. Sistema associa a tecla aos caracteres.
Fluxos Alternativos:
2a. Usuário cancela a atividade.
2a1. Sistema fecha a janela de entrada de caracteres; a tecla não é mapeada a símbolos *Unicode*.

2.2.10 Configurar Tecla de Acentuação

Código: UC2.3
Descrição: Usuário quer configurar uma tecla física como uma tecla de acentuação.
Pré-condições: *Layout* está aberto.
Pós-condições: Tecla física foi mapeada a uma tecla de acentuação.
Extensões: Nenhum.
Fluxo Principal:
1. Usuário seleciona uma determinada tecla para edição.
2. Sistema mostra uma janela para a entrada dos dados sobre uma tecla de acentuação.
3. Usuário entra os dados de uma tecla de acentuação (caracteres independentes e caracteres para substituição). **Include**: Caso de uso UC2.6 Escolher Caractere *Unicode*.
4. Sistema associa a tecla aos dados informados.
Fluxos Alternativos:
3a. Usuário cancela a atividade.
3a1. Sistema fecha a janela da entrada de dados; a tecla não é mapeada a uma tecla de acentuação.

2.2.11 Configurar Macro

Código: UC2.4
Descrição: Usuário quer configurar uma tecla física com uma sequência de sinais simulados.
Pré-condições: *Layout* está aberto.
Pós-condições: Tecla física foi mapeada a uma sequência de sinais simulados.
Extensões: ø
Fluxo Principal:
1. Usuário seleciona uma determinada tecla para edição.
2. Sistema mostra uma janela para a seleção de comandos de teclas virtuais.
3. Usuário monta a sequência de teclas virtuais.
4. Usuário dá um nome ao macro (para mostrar visualmente na tela)
5. Sistema associa a tecla aos dados informados.
Fluxos Alternativos:
3a. Usuário cancela a atividade.
3a1. Sistema fecha a janela da entrada de dados; a tecla não é mapeada a uma tecla de acentuação.

2.2.12 Configurar Tecla de Arquivo Executável

Código: UC2.5
Descrição: Usuário quer configurar uma tecla física para executar algum arquivo em disco.
Pré-condições: *Layout* está aberto.
Pós-condições: Tecla física foi mapeada a um arquivo executável.
Extensões: Nenhum.
Fluxo Principal:
1. Usuário seleciona uma determinada tecla para edição.
2. Sistema mostra uma janela para a seleção de um arquivo em disco.
3. Usuário seleciona um arquivo executável.
4. Sistema oferece a possibilidade de abrir o arquivo executável com parâmetros.
5. Usuário insere os parâmetros para o arquivo executável.
6. Sistema associa a tecla ao arquivo em disco e aos parâmetros informados.
Fluxos Alternativos:
3a. Usuário não seleciona um arquivo.

3a1. A atividade é cancelada; a tecla não é associada a uma ação.

4a. Usuário informa que não deseja abrir o arquivo executável com parâmetros.

4a1. Fluxo pula para o passo 6.

2.2.13 Escolher Caractere Unicode

Código: UC2.6

Descrição: O sistema deve oferecer ao usuário alguma maneira de inserir um ou mais caracteres *Unicode*, apresentando um sumário sobre estes caracteres.

Pré-condições: Nenhum.

Pós-condições: Nenhum.

Extensões: Nenhum.

Fluxo Principal:

1. Sistema apresenta ao usuário uma tela para a entrada de texto em *Unicode*, contendo um campo para a entrada de caracteres e um painel mostrando um sumário dos caracteres inseridos (código de ponto, nome *Unicode* e outras informações).

2. Usuário insere os caracteres desejados.

3. Sistema atualiza o painel com os detalhes do(s) caractere(s) inseridos. O usuário pode continuar editando os caracteres inseridos.

4. Usuário confirma o texto inserido.

Fluxos Alternativos: Nenhum.

2.2.14 Mudar Layout de Teclado

A representação de um teclado que é mostrada na tela depende do *layout* físico para determinar o posicionamento das teclas. Além disso, o controle de teclado anota cada tecla com a letra impressa de acordo com um *layout* lógico. O usuário deve ser capaz de configurar qual *layout* físico e qual *layout* lógico cada teclado utiliza.

2.2.14.1 Mudar Layout Físico

Código: UC2.7.1

Descrição: Se o *layout* físico de algum dos teclados do usuário não é representado corretamente pelo arranjo visual apresentado na tela, ele deve poder mudar o arranjo visual dos controles na tela para melhor representar o posicionamento físico das teclas em seu teclado (vide regra de negócio RN1.1).

Pré-condições: *Layout* está aberto.

Pós-condições: Nenhum.

Extensões: Nenhum.

Fluxo Principal:

1. Usuário seleciona o comando de editar o *layout* de um dos teclados presentes no layout aberto.

2. Sistema apresenta uma lista de padrões físicos para a escolha do usuário.

3. Usuário escolhe um padrão físico.

4. Sistema passa a mostrar aquele teclado usando o posicionamento de teclas físicas de acordo com o padrão físico selecionado.

Fluxos Alternativos:

3a. Usuário cancela a ação.

3a1. Sistema cancela a atividade; o teclado não é associado a um novo *layout* físico.

2.2.14.2 Mudar Layout Lógico

Código: UC2.7.2

Descrição: Se as anotações apresentadas na tela não correspondem às letras impressas no teclado físico do usuário, ele deve poder mudar as anotações usadas pelo controle de teclado para corresponder ao seu dispositivo.

Pré-condições: *Layout* está aberto.

Pós-condições: Nenhum.

Extensões: Nenhum.

Fluxo Principal:

1. Usuário seleciona o comando de editar o *layout* de um dos teclados presentes no *layout* aberto.

2. Sistema apresenta uma lista de *layouts* lógicos, indicando o *layout* físico mais apropriado para cada opção (Ex.: *layout* lógico ISO – DE / *layout* físico ISO).

3.Usuário escolhe um *layout* lógico.

4. Sistema passa a mostrar o teclado em questão anotando as teclas de acordo com o *layout* lógico escolhido.

Fluxos Alternativos:

3a. Usuário cancela a ação.

3a1. Sistema cancela a atividade; o teclado continua usando as mesmas anotações.

2.2.15 Gerenciar Teclados

2.2.15.1 Adicionar Teclado

Código: UC2.8.1
Descrição: Usuário deseja configurar um novo teclado no sistema para mapear suas teclas físicas.
Pré-condições: Nenhum
Pós-condições: Teclado inserido passa a estar disponível para mapeamentos.
Extensões: Nenhum
Fluxo Principal:
1. Usuário realiza o comando para adicionar um novo teclado.
2. Sistema adiciona um controle de teclado vazio no final da lista de teclados.
(Nota: Após esta ação, o usuário deve configurar o teclado.)
Fluxos Alternativos: Nenhum.

2.2.15.2 Editar Teclado

Código: UC2.8.2
Descrição: Usuário deseja alterar informações sobre um teclado que já existe no *layout* aberto.
Pré-condições: *Layout* está aberto e contém ao menos um teclado configurado.
Pós-condições: Nenhum
Extensões: Nenhum
Fluxo Principal:
1. Usuário clica no ícone para editar um teclado já configurado no sistema.
2. Sistema apresenta a tela para o preenchimento de informações sobre o teclado, preenchido com as informações existentes. Esta tela deve incluir também a opção de detectar automaticamente o nome do teclado do usuário.
3. Usuário altera as informações sobre o *layout*.
4. Sistema salva as informações.
Fluxos Alternativos:
3a. Usuário cancela a operação.
3a1. Fluxo é interrompido; o sistema não salva as alterações.

2.2.15.3 Excluir Teclado

Código: UC2.8.3

Descrição: Usuário remove um teclado configurado do *layout*; o *layout* em questão passa a não identificar mais o teclado que foi excluído.

Pré-condições: *Layout* está aberto e contém ao menos um teclado configurado.

Pós-condições: Nenhum.

Extensões: Nenhum.

Fluxo Principal:

1. Usuário clica na opção de remover um teclado do *layout*.
2. Sistema pede confirmação, e também oferece a possibilidade de dissociar o *layout* existente ao teclado físico sem a necessidade de deletar.
3. Usuário confirma a exclusão do teclado.
4. Sistema remove os mapeamentos que estavam associados ao teclado excluído.

Fluxos Alternativos:

3a. Usuário não confirma a exclusão do teclado.

3a1. Fluxo é interrompido; a configuração do teclado não é removida do sistema.

3b. Usuário decide dissociar o dispositivo do *layout* sem deletar os dados.

3b1. Sistema remove as informações sobre o teclado físico do *layout*; neste caso, os mapeamentos que estavam associados ao teclado excluído ficam inativos.

2.2.16 Gerenciar Modificadores

2.2.16.1 Adicionar Tecla Modificadora

Código: UC2.9.1

Descrição: Usuário quer adicionar um modificador a um determinado teclado no *layout* sendo trabalhado.

Pré-condições: *Layout* está aberto e tem ao menos um teclado.

Pós-condições: Nenhum.

Extensões: Nenhum.

Fluxo Principal:

1. Usuário dá o comando de registrar uma determinada tecla como modificador.
2. Sistema apresenta a tela de edição de modificadores.
3. Usuário dá o nome ao modificador, ou adiciona esta tecla a um modificador já existente (referir à regra de negócio RN3.2. Composição da Tecla Modificadora para informações sobre modificadores com mais de uma tecla física).
4. Sistema registra o novo modificador no teclado.

Fluxos Alternativos:

1a. A tecla selecionada pelo usuário possui mapeamentos em outras camadas, e registrá-la como tecla modificadora causaria estes mapeamentos a serem perdidos (de acordo com a regra de negócio 3.5. Independência de Mapas de *Scancodes*, tais teclas são consideradas como modificadoras independentemente da camada).

1a.1. Sistema informa ao usuário de que mapeamentos em outras camadas serão perdidos, e pede confirmação para continuar.

1a.2. Se o usuário decide continuar, o fluxo continua ao passo 2. Se não, o fluxo é interrompido e a tecla não é registrada como modificador.

3a. Usuário cancela a atividade.

3a1. Fluxo é interrompido; o modificador não é salvo.

3b. Usuário associa o modificador a uma ou mais teclas físicas que já estão mapeadas a alguma ação.

3b1. Sistema informa o usuário de que esta ação não pode ser realizada, e volta ao passo 2, sem desfazer as informações que o usuário já havia inserido.

2.2.16.2 Editar Tecla Modificadora

Código: UC2.9.2

Descrição: Usuário edita um modificador que já está cadastrado em algum teclado.

Pré-condições: *Layout* está aberto e tem ao menos um teclado, o qual tem ao menos um modificador.

Pós-condições: Nenhum

Extensões: Nenhum

Fluxo Principal:

1. Usuário dá o comando de editar um modificador já existente e já associado a algum determinado teclado.
2. Sistema apresenta a tela de edição de modificadores.
3. Usuário altera os dados da tecla modificadora.
4. Sistema salva as alterações realizadas ao modificador.

Fluxos Alternativos:

3a. Usuário cancela a atividade.

3a1. Fluxo é interrompido; as alterações não são salvas.

2.2.16.3 Excluir Tecla Modificadora

Código: UC2.9.3

Descrição: Usuário quer remover um modificador de algum determinado teclado.

Pré-condições: *Layout* está aberto e tem ao menos um teclado, o qual tem ao menos um modificador.

Pós-condições: Nenhum.

Extensões: Nenhum.

Fluxo Principal:

1. Usuário dá o comando de excluir uma tecla física que está registrada como tecla modificadora.
2. Sistema pede a confirmação do usuário, avisando-o também de que as camadas que dependem desta tecla modificadora se tornarão inativas.
3. Usuário confirma a exclusão.
4. Sistema deixa de considerar a tecla física como uma tecla modificadora.

Fluxos Alternativos:

3a. Usuário não confirma a exclusão.

3a1. Fluxo é interrompido. O modificador não é excluído.

3a. Sistema determina que a remoção da tecla física resulta em um modificador que não está associado a nenhuma tecla física.

3a1. A confirmação de exclusão deve informar ao usuário de que as camadas que dependem deste modificador serão removidas.

3a2. O fluxo continua normalmente no passo 4.

2.2.17 Gerenciar Camadas do Teclado

As camadas de um teclado são diferentes mapeamentos de teclas físicas para ações que são acessadas usando combinações de teclas modificadoras. Por exemplo, pressionar a tecla *Shift* em *layouts* convencionais acessa uma camada adicional, tipicamente contendo caracteres maiúsculos.

Camadas devem ser mostradas no mesmo controle na tela, e o usuário deve ser capaz de navegar entre elas selecionando a combinação de teclas modificadoras. Os casos de uso nesta seção explicam em detalhes o caso de uso representado no diagrama na imagem 2 como Gerenciar Camadas do Teclado.

2.2.17.1 Adicionar Camada

Código: UC2.10.1

Descrição: Usuário quer adicionar uma nova camada em um certo teclado. Esta camada precisa de uma certa combinação de modificadores presentes no mesmo teclado para ser ativado.

Pré-condições: Existe ao menos uma tecla modificadora configurada no teclado onde a camada será incluída.

Pós-condições: Nenhum.

Extensões: Nenhum.

Fluxo Principal:

1. Usuário pressiona a combinação correta de teclas modificadoras no controle de teclado para acessar a camada correta.
2. Sistema mostra a camada, sem nenhum comando mapeado.
3. Usuário pode mapear esta nova camada.

(Nota: O sistema instancia uma camada quando esta é necessária, sem informar o usuário).

Fluxos Alternativos: Nenhum.

2.2.17.2 Editar Camada

Código: UC2.10.2

Descrição: Usuário quer alterar alguma informação sobre uma camada de um determinado teclado no *layout* sendo trabalhado.

Pré-condições: Existe ao menos uma tecla modificadora configurada no *layout* atual.

Pós-condições: Nenhum.

Extensões: Nenhum.

Fluxo Principal:

1. Usuário acessa uma determinada camada.
2. Sistema mostra uma janela ao lado contendo um sumário da camada selecionada, que pode ser editado.
3. Usuário altera os dados necessários.
4. Sistema salva os dados na camada que foi aberto.

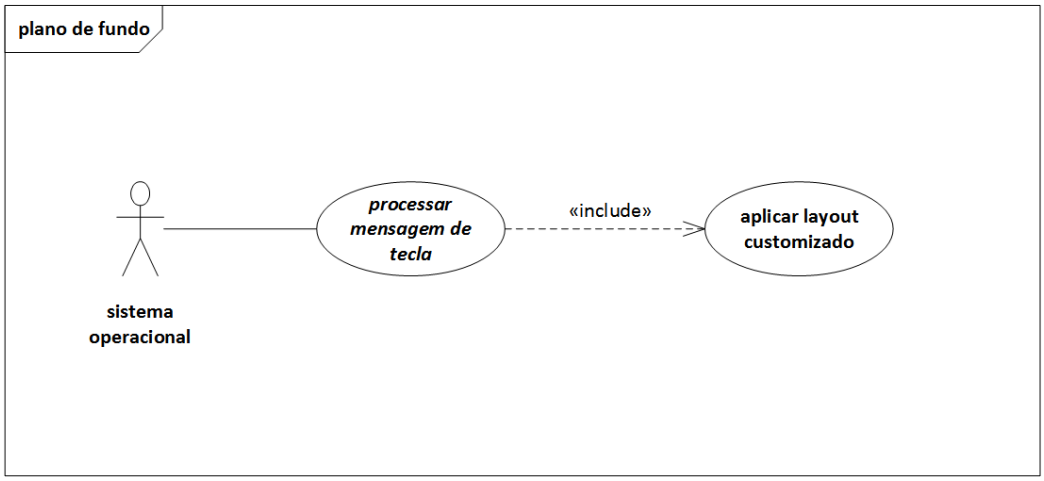
Fluxos Alternativos: nenhum.

2.3 Subsistema *MultiKeys Core*

Este subsistema é responsável pela execução dos *layouts* definidos pelo usuário em plano de fundo. Embora o usuário não interaja com este subsistema através de uma

interface gráfica, ele envia sinais de teclado que são recebidos e processados pelo sistema.

Figura 3: Diagrama de casos de uso do processo em plano de fundo.



2.3.1 Processar Mensagem de Tecla

A especificação deste caso de uso se encontra no documento *Especificação de Caso de Uso UC3.1*.

2.3.2 Aplicar Layout Customizado

A especificação deste caso de uso se encontra no documento *Especificação de Caso de Uso UC3.2*.

ESPECIFICAÇÃO DE CASO DE USO

Multikeys – Aplicação para Remapemento de Teclas em *Windows*

Caso de Uso 3.1: Processar Mensagem de Tecla

FATEC
Faculdade de Tecnologia de Mogi das Cruzes
Tecnologia em Análise e Desenvolvimento de Sistemas | Tecnologia em Agronegócio | Tecnologia em Recursos Humanos

Histórico de Versões

Data	Versão	Descrição	Autor	Revisor
09/05/2017	0.1	Versão preliminar	Rafael Kenji	-
27/05/2017	1.0	Primeira versão	Rafael Kenji	-
20/10/2017	1.1	Atualização	Rafael Kenji	-
28/11/2017	1.2	Versão final	Rafael Kenji	-

Índice

1 Nome do Caso de Uso

UC3.1 – Processar Mensagem de Tecla

2 Objetivo

Esta especificação de caso de uso tem como objetivo prover uma solução computacional capaz de interceptar e identificar sinais de tecla recebidos pelo sistema operacional, e substituir ações de acordo com mapeamentos customizados pelo usuário.

3 Descrição

Uma entrada de teclado vinda do usuário é interpretada pelo *driver* de dispositivo, gerando uma mensagem que é processada pelo sistema operacional. A mensagem é interceptada pela aplicação *Multikeys* em plano de fundo, que extrai as informações sobre a tecla pressionada. A interceptação é realizada primeiro através da API *Raw Input*, para a obtenção de dados de baixo nível, e novamente através de um *Hook* de teclado, para a manipulação da mensagem e o possível bloqueio desta.

4 Requisitos

4.1 Requisitos Funcionais

RF2.1 – Identificar Entrada de Teclado

4.2 Requisitos Não-Funcionais

RNF003 – Tempo de Resposta
RNF004 – Robustez
RNF005 – Uso de Recursos
RNF006 – Disponibilidade

4.3 Requisitos de Negócio

RN2.1 – *Scancodes*
RN3.3 – Tecla Modificadora *AltGr*
RN5.1 – *Pause*/*Break*
RN5.2 – *Ctrl* + *Pause*/*Break*
RN5.3 – *PrintScreen*
RN5.4 – *Alt* + *PrintScreen*
RN5.5 – *Shift* Sintético
RN5.6 – *Eisuu* / Alfanumérico

5 Tipo de Caso de Uso

Este caso de uso é concreto, iniciado diretamente pelo ator.

6 Atores

6.1 Sistema Operacional

Neste caso de uso, o sistema operacional *Windows* é considerado o único ator. Embora o usuário seja responsável por causar as mensagens de tecla, ele não interage diretamente com o subsistema *Multikeys Core*, pelo fato deste ser um processo em plano de fundo.

A interação é realizada através da API do *Windows*. O sistema operacional envia mensagens *RawInput* para receptores cadastrados, e as mensagens de entrada de teclado são recebidas por qualquer sistema que faça parte da chamada *Hook Chain*. Através desta mesma *hook chain* ocorre a devolução de mensagens ao sistema operacional. Adicionalmente, ações adicionais, como o envio de um caractere *Unicode*, também ocorre através de chamadas de função da API do *Windows*.

Como o usuário nunca fica ciente destas transações, ele é excluído como ator deste caso de uso.

7 Pré-Condições

7.1 Plano de Fundo

O processo em plano de fundo do sistema *Multikeys* deve estar em execução.

7.2 Mensagem de Teclado

O sistema operacional (*Windows*) recebeu uma mensagem originada do dispositivo de teclado.

8 Fluxo de Eventos

8.1 Fluxo Principal

1. *Windows* envia uma mensagem *RawInput* para o subsistema *Multikeys Core*.

2. O sistema extrai os dados desta mensagem, incluindo o nome do teclado.

3. Sistema chama o método do objeto *Remapper* que avalia se existe ação mapeada a esta entrada de tecla, de acordo com o layout customizado do usuário.

Include: O processamento realizado pelo objeto *Remapper* é descrito pelo caso de uso UC3.2 Aplicar *Layout* Customizado.

4. Um comando é recebido pelo sistema (que pode não existir caso a mensagem não corresponda a uma tecla mapeada pelo usuário), juntamente com a decisão se a tecla deve ou não ser bloqueada.

Nota: Uma decisão é composta por uma *flag* que indica se a entrada do usuário deve ser bloqueada ou não, e um comando caso a entrada deva ser bloqueada e substituída por outra ação; este comando pode realizar diversas ações, como enviar um caractere *Unicode* ou simular outro sinal de tecla. Como a API *RawInput* não consegue bloquear a entrada de usuário, esta decisão é armazenada para ser lida em outro momento. Além disso, a decisão deve ocorrer através da API *RawInput* porque *hooks* têm informação limitada sobre a entrada de tecla.

5. A decisão é armazenada para ser recuperada posteriormente.

6. Windows envia uma mensagem de teclado, que chega ao sistema *Multikeys* através de um *hook* que existe na *hook chain*.

7. O *hook* chama o processo *Multikeys Core* perguntando se a entrada interceptada deve ser bloqueada.

Nota: Um *hook* deve existir em uma biblioteca separada para que seu efeito seja global (isto é, para que possa receber entradas de teclado direcionadas a outras janelas

e processos). A chamada de método ocorre através do sistema de mensagens do *Windows*.

8. O sistema procura uma decisão armazenada correspondente á entrada recebida.

9. O sistema recupera a decisão, executando o comando se este existir; neste momento, as decisões armazenadas são removidas.

10. O sistema retorna a decisão ao *hook*, que devolve a mensagem ao sistema somente se a tecla não deve ser bloqueada.

8.2 Fluxos Alternativos

8.2.1 Mensagem de Tecla de Acentuação do Sistema Operacional

Mensagens que são recebidas pelo sistema operacional são interpretadas juntamente com o estado das teclas modificadoras *Shift*, *Alt*, *Ctrl* e *Windows*. No caso destas teclas, seu bloqueio não é suficiente para que estes não sejam considerados durante o recebimento de outra tecla, interferindo nas mensagens simuladas.

A resolução deste problema consiste em injetar mensagens de soltura de tecla logo após uma tecla modificadora bloqueada ser recebida.

4a. A partir dos dados extraídos, a mensagem é detectada como uma tecla de acentuação do sistema operacional (*Shift*, *Alt*, *Ctrl* ou *Windows*); além disso, foi determinado que esta mensagem deve ser bloqueada.

4a.1. Uma mensagem representando a soltura da tecla correspondente é simulada.

4a.2. O fluxo continua para o passo 5.

8.2.2 Mensagem Injetada

Uma das possíveis ações mapeadas a uma tecla é a simulação de outras teclas. A simulação de entrada de teclado pode causar o programa a interceptar suas próprias mensagens, causando *loops* infinitos.

É necessário que o *hook* identifique mensagens injetadas, para que não as bloqueie, e não avalie mensagens simuladas pelo sistema Multikeys.

6a. *Hook* intercepta uma mensagem identificada como injetada pelo sistema MultiKeys.

Nota: O sistema é responsável por adicionar alguma informação a mensagens injetadas de forma que esta seja identificável como tal.

6a.1. *Hook* ignora a tecla injetada, sem bloquear (mensagens não são enviadas ao processo *Multikeys Core* para avaliação).

8.2.3 Mensagem Pause/Break

A tecla *pause*/*break* é única em gerar um scancode de três *bytes* de comprimento (0xe1 0x1d 0x45). Uma única mensagem de teclado é interceptada pelo *hook* contendo o scancode 0x45, e duas mensagens são recebidas pela API *RawInput* (vide regra de negócio RN5.1 *Pause*/*Break*).

7a. A mensagem interceptada é identificada como um *Pause*/*Break*.

Nota: Esta checagem não deve considerar a tecla *Ctrl* + *Pause*/*Break*, que possui um *scancode* diferente, de acordo com a regra de negócio RN5.2. Esta combinação não precisa de tratamento diferenciado.

7a.1. O sistema deve procurar pela decisão para a tecla 0xe1 0x1d. A segunda mensagem *Raw Input* é ignorada, e não fará efeito.

7a.2. O fluxo retorna para o passo 8 do fluxo principal.

8.2.4 Mensagem PrintScreen

A tecla *PrintScreen* envia um *shift* sintético, e também possui outro *scancode* quando pressionado juntamente com *Alt*. Porém, o maior problema é que pressionar a tecla *PrintScreen* causa uam mensagem de soltura de tecla (*key break*), não acompanhada da pressão de tecla (*key make*). Este comportamento deve ser corrigido, pois ações são mapeadas a pressões de tecla.

7b. A mensagem interceptada é identificada como um *PrintScreen break*.

Nota: Esta checagem deve levar em consideração que o *scancode* é diferente quando a tecla é pressionada juntamente com *Alt*.

7b.1. Uma cópia da mensagem recebida deve ser simulada, com a exceção de que o clone contém uma mensagem *make* ao invés de *break*. Esta mensagem não pode ser filtrada como mensagem injetada.

7b.2. O fluxo é suspenso até que a mesagem seja interpretada, de acordo com o fluxo normal deste caso de uso.

7b.3. O fluxo retorna para o passo 8 do fluxo principal.

8.2.5 Ordem de Mensagens Incorreta

Mensagens vindas do sistema operacional podem chegar em qualquer ordem. Embora mensagens recebidas pela API *RawInput* quase sempre cheguem antes daquelas interceptadas pelo *hook*, é possível que o inverso ocorra. Neste caso, o fluxo principal é iniciado no passo 6, fazendo com que não haja decisão adequada armazenada na fila de decisões; se isto ocorre, o fluxo alternativo a seguir é realizado a partir do passo 8:

8a. Decisão não foi encontrada.

8a.1. O sistema entra em um *loop* por aprox. 50ms, aguardando uma mensagem *RawInput* atrasada.

8a.2. Ao receber a mensagem, são realizados os passos 2 a 4 do fluxo principal (extrair dados da mensagem, avaliar tecla e obter decisão de bloqueio).

8a.3. Ocorre uma tomada de decisão:

- Se a mensagem recebida pela API *RawInput* realmente corresponde à mensagem recebida pelo *hook*, prossegue-se ao próximo passo.

- Se a mensagem não corresponde à mensagem recebida (é originada de outra tecla), o sistema armazena a decisão na fila e retorna ao passo 8a.1 para aguardar a mensagem correta. O temporizador não é reiniciado.

8a.4. O fluxo de eventos retorna ao fluxo principal no passo 10.

8.2.6 Mensagem AltGr

A tecla *AltGr* tem um comportamento diferente das outras teclas (vide regra de negócio RN3.3 Tecla Modificadora *AltGr*). Uma mensagem recebida pelo *hook* contendo uma tecla *Control* esquerda é acompanhada por uma mensagem *RawInput* atrasada contendo o *Alt* direito. Uma segunda mensagem *hook* é recebida logo após contendo o *Alt* direito.

8a.1a. Se uma mensagem correspondente ao *Ctrl* esquerdo não encontrou sua decisão correspondente, o sistema deve ao invés esperar por uma mensagem atrasada que corresponda á tecla *Alt* direita.

8a.1a.1. Ao receber a mensagem *Alt*, uma *flag* é ativada para ignorar a próxima mensagem *hook* de A*lt* direito.

8a.1a.2. O fluxo é redirecionado ao passo 8a.2, para a avaliação da mensagem *RawInput* corresponde à tecla *Alt* direita.

8.3 Fluxos de Exceção

8.3.1 Timeout

O sistema entra em um *loop* limitado por tempo no passo 8a.1. Se o *loop* é finalizado por tempo, o fluxo alternativo a seguir é realizado a partir do passo 8.a.1:

8a.1a. *Loop* deu timeout sem receber uma mensagem *RawInput*.

8a.1a.1. O sistema devolve uma resposta ao *hook*, informando que a tecla não deve ser bloqueada.

9 Pós-Condições

9.1 Disponibilidade

Após o processamento da entrada de tecla, o sistema deve ficar pronto para receber outra mensagem de tecla imediatamente.

10 Referências

Para detalhes sobre os requisitos do sistema *Multikeys*, veja o Documento de Requisitos Funcionais e Não-Funcionais, e o Documento de Regras de Negócio.

Detalhes sobre outros casos de uso podem ser encontrados no Documento de Caso de Uso, e o caso de uso UC3.2. na Especificação de Caso de Uso 3.2. Detalhes sobre implementações estão disponíveis do Documento de Arquitetura.

ESPECIFICAÇÃO DE CASO DE USO

Multikeys – Aplicação para Remapemento de Teclas em *Windows*

Caso de Uso 3.2: Aplicar Layout Customizado

FATEC
Faculdade de Tecnologia de Mogi das Cruzes
Tecnologia em Análise e Desenvolvimento de Sistemas | Tecnologia em Agronegócio | Tecnologia em Recursos Humanos

Histórico de Versões

Data	Versão	Descrição	Autor	Revisor
27/05/2017	1.0	Primeira versão	Rafael Kenji	-
21/10/2017	1.1	Revisão	Rafael Kenji	-
28/11/2017	1.2	Versão final	Rafael Kenji	-

Índice

1 Nome do Caso de Uso

UC3.2 – Aplicar *Layout* Customizado

2 Objetivo

Esta especificação de caso de uso tem como objetivo prover uma solução computacional capaz de checar os dados de uma tecla interceptada pelo sistema de acordo com um *layout* definido pelo usuário, e identificar que ação deve ser realizada em resposta à mensagem recebida.

3 Descrição

Depois que o sistema *Multikeys* intercepta uma mensagem vinda do usuário, é necessário checar o modelo interno do *layout* ativo para saber qual ação deve ser realizada. O sistema deve ser capaz de interpretar teclas modificadoras, teclas de acentuação e outras funcionalidades típicas de um *layout* tradicional.

A tarefa de encontrar o comando mapeado a uma tecla é responsablidade do módulo chamado *Remapper*.

No caso de existir uma ação mapeada à tecla, a ação a ser realizada é retornada na forma de um comando, que utiliza a API do *Windows* para simular sinais de tecla, enviar caracteres *Unicode*, e possivelmente outras ações.

4 Requisitos

4.1 Requisitos Funcionais

RF2.2 – Simular Entrada de Texto
RF2.3 – Simular Sinais de Tecla
RF2.4 – Abrir Arquivos Executáveis
RF2.5 – Suportar Teclas Modificadoras
RF2.6 – Suportar Teclas de Acentuação

4.2 Requisitos Não-Funcionais

RNF003 – Tempo de Resposta
RNF004 – Robustez
RNF005 – Uso de Recursos
RNF006 – Disponibilidade

4.3 Requisitos de Negócio

RN2.1 – *Scancodes*
RN2.2 – Representação *Unicode*
RN3.1 – Estado de Tecla Modificadora
RN3.2 – Composição de Tecla Modificadora
RN4.1 – Estado de Tecla de Acentuação
RN4.2 – Combinações de Teclas de Acentuação
RN4.3 – Teclas Invisíveis para Teclas de Acentuação
RN4.4 – Representação *Unicode* de uma Tecla de Acentuação
RN4.5 – Sequência Inválida de Tecla de Acentuação
RN4.6 – Reclas de Acentuação e Estados de Modificadores

5 Tipo de Caso de Uso

Este caso de uso é incluído pelo caso de uso UC3.1. Portanto, este caso de uso é concreto, iniciado sob as mesmas circumstâncias que o caso de uso UC3.1.

6 Atores

6.1 Sistema Operacional

Neste caso de uso, o sistema operacional *Windows* é considerado o único ator. Embora o usuário seja responsável por causar as mensagens de tecla, ele não interage diretamente com o subsistema *Multikeys Core*, pelo fato deste ser um processo em plano de fundo.

A interação é realizada através da API do *Windows*. O sistema operacional envia mensagens *RawInput* para receptores cadastrados, e as mensagens de entrada de teclado são recebidas por qualquer sistema que faça parte da chamada *Hook Chain*.

Através desta mesma *hook chain* ocorre a devolução de mensagens ao sistema operacional. Adicionalmente, ações adicionais, como o envio de um caractere *Unicode*, também ocorre através de chamadas de função da API do Windows.

Como o usuário nunca fica ciente destas transações, ele é excluído como ator deste caso de uso.

7 Pré-Condições

7.1 Plano de Fundo

O processo em plano de fundo do sistema *Multikeys* deve estar em execução.

7.2 Chamada de Função

O sistema *Multikeys* está requisitando o comando que corresponde a uma tecla interceptada. O sistema já possui informações sobre a tecla, incluindo o *scancode* e o nome do teclado que originou o sinal.

8 Fluxo de Eventos

8.1 Fluxo Principal

Este processamento é incluído entre os passos 3 e 4 do fluxo principal no caso de uso UC3.1: Processar Mensagem de Tecla.

1. As informações sobre a tecla pressionada chegam ao modelo interno do sistema *Multikeys*.

2. O sistema procura o teclado (no modelo) que possui nome igual ao teclado que originou o sinal, e envia os dados da tecla para avaliação.

3. A tecla é avaliada pela classe de teclado e enviada para a camada de teclado ativa para avaliação.

4. O comando correspondente ao *scancode* da tecla recebida é recuperado pela camada de teclado correta.

5. O *Remapper* retorna o comando mapeado á tecla.

8.2 Fluxos Alternativos

8.2.1 Teclado Configurado para Responder a Todas as Mensagens

É possível configurar, dentro de um *layout*, um teclado cujos remapeamentos se apliquem a todos os dispositivos de teclado do usuário. A utilização pretendida desta funcionalidade é permitir ao usuário prover um conjunto de remapeamentos aplicáveis a todos os dispositivos de teclado não configurados.

2a. O sistema encontra um objeto de teclado configurado para responder a qualquer dispositivo de teclado.

2a.1. *Remapper* considera o teclado encontrado como o apropriado para receber a mensagem para avaliação.

2a.2. O fluxo continua para o passo 3.

8.2.2 Teclado não Utilizado pelo Layout

O usuário pode normalmente utilizar um teclado conectado ao computador que não está mapeado no sistema *Multikeys*. Neste caso, o sistema deve dexar a mensagem passar normalmente, sem executar qualquer ação.

2b. O sistema não encontra o teclado no *layout* ativo.

2b.1. *Remapper* retorna que não há comando mapeado a esta tecla.

8.2.3 Ausência de Camada Apropriada

Os mapeamentos entre teclas físicas e ações são organizados em camadas, correspondentes às combinações de teclas modificadoras. Se o teclado encontrado não possui uma camada com uma combinação de modificadores correspondente aos modificadores atualmente pressionados pelo usuário, a mensagem deve ser processada pelo sistema operacional normalmente, sem executar qualquer ação remapeada.

3a. A classe de teclado possui um nível atualmente ativo nulo.

3a.1. Remapper retorna que não há comando mapeado a esta tecla.

8.2.4 Tecla Modificadora

Se a tecla recebida é uma tecla modificadora, o modelo deve ser atualizado de acordo.

3b. A classe de teclado constata que a tecla interceptada está registrada como um modificador no teclado de onde originou.

3b.1. A classe de teclado atualiza seu estado interno de modificadores, para refletir a combinação de modificadores atualmente pressionados pelo usuário.

3b.2. A classe de teclado procura entre as camadas cadastradas, qual destas corresponde à combinação de modificadores atualmente ativos.

3b.3. A camada encontrada passa a ser usada como a camada ativa do teclado. Na possibilidade da combinação de modificadores não corresponder a nenhum nível cadastrado, uma referência nula é utilizada para representar a ausência de camada apropriada.

8.2.5 Tecla de Acentuação Ativa

Se o teclado em questão está em um estado de tecla de acentuação (porque uma tecla de acentuação foi pressionada anteriormente), qualquer comando encontrado deve passar pela tecla de acentuação lembrada pelo teclado. Mesmo que um comando não seja encontrado, a tecla de acentuação deve ser notificada da próxima pressão de tecla no mesmo teclado. A tecla retornada é a tecla de acentuação armazenada anteriormente, que decide sua ação de acordo com a nova mensagem recebida.

4a. O teclado em questão possui uma tecla de acentuação ativa (cada teclado mantém uma referência à tecla de acentuação ativa, que é nula quando não há tecla de acentuação ativa).

4a.1. A tecla de acentuação recebe a tecla pressionada, junto com o comando se algum foi encontrado.

4a.2. O comando de tecla de acentuação confere o comando recebido na sua lista interna de substituições para determinar o seu algoritmo de execução.

4a.3. *Remapper* retorna a tecla de acentuação modificada.

8.2.6 Tecla Recebida não Está Mapeada do Nível Ativo

Caso a tecla pressionada não possua nenhum mapeamento no contexto atual, considerando teclado de origem e combinação de modificadores, o sistema deve deixar a mensagem passar normalmente, sem executar qualquer ação.

4b. O sistema não encontra um comando apropriado no nível ativo.

4b.1. *Remapper* retorna que não há comando mapeado a esta tecla.

8.2.7 Comando Mapeado Corresponde a uma Tecla de Acentuação

O usuário pode definir teclas como teclas de acentuação, que possuem além de sua representação *Unicode*, uma lista de substituições aplicáveis para a tecla a ser pressionada a seguir.

Nota: Este fluxo alternativo só é checado após o fluxo alternativo 4a. Portanto, no caso de duas teclas de acentuação pressionadas em sequência, o fluxo alternativo 4a é realizado, e não este.

4c. O sistema encontrou um comando de tecla de acentuação.

4c.1. Ao invés de retornar o comando, o modelo interno do teclado em questão é atualizado para armazenar a tecla de acentuação.

4c.2. *Remapper* retorna um comando vazio.

8.3 Fluxos de Exceção

Este caso de uso não prevê fluxos de exceção para erros específicos porque o fluxo não deve ser interrompido em nenhum momento, e todas as possibilidades devem ser cobertas pelos fluxos alternativos. Quaisquer erros inesperados devem resultar no *Remapper* retornando que um comando não foi encontrado, sem interromper processos.

9 Pós-Condições

9.1 Estado Atualizado

Caso a mensagem recebida corresponda a qualquer tipo de atualização no modelo interno do *layout*, como uma tecla de acentuação, o modelo deve responder á próxima chamada já com o seu estado atualizado. Em outras palavras, o estado interno do modelo deve persistir entre chamadas.

10 Referências

Para detalhes sobre os requisitos do sistema *Multikeys*, veja o Documento de Requisitos Funcionais e Não-Funcionais, e o Documento de Regras de Negócio.

Detalhes sobre outros casos de uso podem ser encontrados no Documento de Caso de Uso, e o caso de uso UC3.1, que inclui este caso de uso, é documentado na Especificação de Caso de Uso 3.1. O documento de Arquitetura possui detalhes sobre a implementação da classe Remapper e sua estrutura interna.