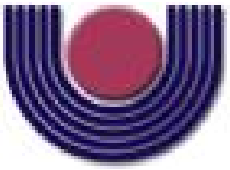

Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Especificação e Implementação de um Leitor de Tela

Cleiton Fiatkoski Balansin

CASCAVEL
2011

CLEITON FIATKOSKI BALANSIN

ESPECIFICAÇÃO E IMPLEMENTAÇÃO DE UM LEITOR DE TELA

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Dr. Jorge Bidarra

CASCAVEL
2011

CLEITON FIATKOSKI BALANSIN

ESPECIFICAÇÃO E IMPLEMENTAÇÃO DE UM LEITOR DE TELA

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Dr. Jorge Bidarra (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Clodis Boscarioli
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Marcio Seiji Oyamada
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 16 de novembro de 2011

AGRADECIMENTOS

Agradeço a minha família pelo incentivo, paciência e compreensão. Aos meus professores pelo empenho no ensino e pelo conhecimento passado. Aos meus amigos pelos momentos que passamos juntos.

Lista de Figuras

Lista de Abreviaturas e Siglas

AT-SPI	*Assistive Technology Service Provider Interface*
GNOME	*GNU Network Object Model Environment*
KDE	*K Desktop Environment*
GUI	*Graphical User Interface*
CLI	*Command Line Interface*
DOS	*Disk Operating System*
CORBA	*Common Object Request Broker Architecture*
API	*Application Programming Interface*
JAWS	*Job Access With Speech*
NVDA	*NonVisual Desktop Access*
LGPL	*Lesser General Public License*
HTML	*HyperText Markup Language*

Sumário

Resumo

Um leitor de tela é uma tecnologia assistiva cujo objetivo principal é auxiliar pessoas com deficiência visual na utilização de um computador. Sua função é transformar textos/legendas que aparecem nas telas dos computadores em voz, criando assim uma interação por áudio com o usuário. O leitor de tela especificado e desenvolvido neste trabalho foi implementado na linguagem de programação Python, sob a forma de um *script*, para ser executado no sistema operacional Linux, distribuição Ubuntu. Com a utilização da interface de acessibilidade AT-SPI, que prove recursos de acessibilidade ao sistema Linux, o leitor busca as informações textuais na tela através dos eventos que são capturados junto a movimentação do usuário sobre o sistema. Com isso, o leitor de tela pode executar a leitura do texto através do sintetizador de voz eSpeak, responsável por transformar o texto em fala. A especificação da interface AT-SPI possui uma implementação na linguagem Python, a biblioteca pyatspi, que é utilizada pelo leitor de tela no processamento de captura dos eventos do sistema, a fim de obter os textos da tela que representam esses eventos. O objetivo deste trabalho é criar um leitor que funcione como um módulo externo para que outros aplicativos que desejam utilizar a funcionalidade de leitura de tela possam fazê-lo de uma forma simples. Apesar disso, o leitor de tela também funciona em modo *stand-alone*, ou seja, os usuários também podem executar apenas o leitor de tela, sem a nessecidade de um aplicativo.

Palavras-chave: Leitor de Tela, Sintetizador de Voz, Síntese de Voz.

Capítulo 1

Introdução

Os computadores ao longo dos anos ganharam destaque na sociedade, porém algumas pessoas não conseguem usufruir os benefícios criados por eles. As pessoas em questão são aquelas que sofrem de algum tipo de deficiência. Um ramo da computação estuda meios para que essas pessoas possam ter acesso aos computadores como qualquer outra. Os meios para garantir que uma pessoa com baixa visão, cega, surda ou com qualquer outra deficiência tenha acesso ao computador ganham o nome de Tecnologias Assistivas [Cook e Hussey 2001].

O leitor de tela se enquadra na categoria de tecnologias assistivas, já que seu propósito é criar uma maneira de auxiliar os usuários com deficiência visual na iteração com o computador criando uma interface de áudio. Com isso, o usuário não precisa de um esforço demasiado na identificação dos textos presentes na tela do computador.

Quando a leitura de tela é executada ela evita que o usuário se esforce na identificação dos textos da tela do computador, com isso, o leitor de tela retarda a fadiga visual que o usuário enfrenta por forçar a visão constantemente. Além de ser um recurso importante na utilização do computador por pessoas com baixa visão, um leitor de tela também pode desempenhar um papel importante na vida de pessoas cegas, pois é um dos poucos recursos que possibilita o uso do computador por essas pessoas.

Este trabalho se concentra em desenvolver um leitor de tela para Linux, mais especificamente para ambiente GNOME [Project 2011] do sistema operacional Ubuntu [Canonical 2011]. O trabalho é focado na especificação e implementação do leitor de tela na forma de *script* Python [Foundation 2011], para isso utilizou-se uma interface provedora de recursos de acessibilidade chamada AT-SPI [Foundation 2008], além disso, vários outros recursos computacionais são utilizados para dar suporte ao *script* de leitura de tela. Todos os recursos utilizados e construídos

no desenvolvimento são demonstrados durante o texto. O *script* desenvolvido cria uma interface de áudio com o usuário, lendo os principais componentes gráficos do ambiente GNOME (botões, ícones, menus, etc.) para que o usuário se localize em sua navegação pelo sistema.

Para tornar este projeto realidade, fizemos uso de uma interface de acessibilidade (AT-SPI). Esta interface possui uma biblioteca em Python, chamada pyatspi [Diggs 2011], que possibilitou a criação de um algoritmo genérico que evita uma comunicação específica com o sistema ou com os aplicativos que executam sobre ele, sendo assim, para capturar os textos da tela do computador só necessitamos executar a comunicação com a biblioteca pyatspi de forma genérica para todos os aplicativos do sistema, já que, a pyatspi é que se encarrega de buscar as informações de todos os aplicativos que estão executando no sistema.

A troca de informações que ocorre do leitor de tela para a biblioteca pyatspi acontece via chamadas de funções da biblioteca e a troca de informações que ocorre da pyatspi para o leitor acontece por meio de um loop da pyatspi que executa as funções do leitor quando os eventos registrados pelo próprio leitor ocorrem no sistema. Quando a pyatspi executa uma função do leitor, ela também manda como parâmetro o evento ocorrido e é a partir de processamentos sobre este evento que o leitor identifica o texto da tela a ser lido.

Como o texto a ser lido é obtido através dos eventos precisamos conhecer tais eventos, para isto, utilizamos o *software* Accerciser [Isaacson 2011] que auxiliou na compreensão das estruturas utilizadas pela pyatspi e de como funciona a geração de eventos pela biblioteca. Depois de analisar os eventos no Accerciser foram selecionados alguns desses eventos que cobrem a maioria das ações dos usuários na navegação sobre o sistema operacional. Executando a leitura dos textos relativos a esses eventos selecionados acreditamos que o leitor estabelece uma iteração satisfatória com o usuário. Após este processamento, o texto é obtido e precisa ser lido ao usuário. Para esta tarefa utilizamos o *software* eSpeak [eSpeak 2011] que é um sintetizador de voz com suporte a vários idiomas e sistemas operacionais. Há algum tempo o eSpeak já consta como aplicativo padrão nas instalações do Linux Ubuntu, o que facilitou o desenvolvimento do leitor de tela utilizando este sintetizador.

1.1 Organização do trabalho

O trabalho esta organizado em 5 capítulos, a saber:

1. Capítulo introdutório, contendo a descrição, os objetivos e a organização do trabalho;

2. Capítulo que discute sobre os leitores de tela, seus objetivos e características. O capítulo apresenta uma divisão entre os tipos diferentes de leitores de tela, além de uma sucinta discussão sobre quatro leitores de tela disponíveis atualmente no mercado;

3. Apresenta uma descrição sobre como o processamento de voz é utilizado pelos leitores de tela. Também descreve as principais técnicas para criação de voz artificial;

4. Capítulo que relata toda a fase de desenvolvimento do leitor de tela. Neste capítulo é apresentado todos os recursos utilizados no desenvolvimento, assim como, o relato de como e para que foram utilizados. Uma seção do capítulo é destinada a explicar os problemas enfrentados durante todo o projeto e outra é destinada a explicar o processo de incorporação do leitor de tela por outros aplicativos;

5. Apresenta a análise dos resultados através de uma comparação com o leitor de tela do Orca, uma discussão sobre os trabalhos futuros que podem envolver este projeto, e a conclusão geral deste trabalho.

Capítulo 2

Leitor de Tela

Neste capítulo, descrevemos as características comuns entre os leitores de tela, assim como suas definições e os principais objetivos de um leitor de tela. Outra seção deste capítulo aborda o estágio atual do desenvolvimento de *software* nesta área. Os leitores de tela serão classificados em quatro categorias de acordo com suas características de execução e, por fim, será realizada uma descrição de quatro leitores de tela disponíveis no mercado atualmente.

2.1 Introdução

Um leitor de tela é um recurso computacional que, a partir da identificação de textos na tela do computador (ou outro dispositivo eletrônico), apresenta para o usuário uma saída desse texto de forma oralizada [Blenkhorn e Evans 2000], [Raj 1998]. O funcionamento de um leitor de tela é baseado na monitoração das ações que o usuário executa sobre o sistema operacional. Um leitor de tela será capaz de ler toda ação cujo evento tem associado a si uma descrição textual.

Os leitores de tela são uma tecnologia assistiva útil para usuários com deficiência visual e também para as pessoas com a chamada "vista cansada" (idosos). Por meio dos leitores de tela, esses grupos de pessoas passam a ter acesso a computadores, à Internet e a outras mídias com mais facilidade [Raj 1998]. A filosofia de um leitor de tela deve se basear em possibilitar ao seu usuário, no nosso caso um deficiente visual, uma solução para que este tenha condições de acessar o ambiente computacional como qualquer outro usuário com visão normal [Thatcher 1994]. Mesmo com algum resíduo visual, muitas pessoas com baixa visão dependem de outros auxílios, além de um ampliador de tela, assim sendo, os leitores de tela desempenham um papel importante junto a esses usuários.

Os leitores de tela se diferenciam por suas características, sistemas que são suportados, desempenho, comunicação com outros *softwares*, qualidade da voz, etc. Mas, no momento da leitura, de um modo geral, todos tentam transmitir ao usuário o que se passa na tela do computador. Embora os leitores de tela sejam construídos para ler os textos exibidos em tela do computador, nem todas as informações são passíveis de leitura. O principal obstáculo quanto à dificuldade dos leitores em passar a informação da tela para o usuário é o uso de imagens pelos programas. A dificuldade está em como executar a leitura de um artifício visual. A solução geralmente adotada é a leitura de uma descrição que está anexada de alguma forma a imagem, recurso que é muito utilizado nas páginas HTML. Além desse obstáculo, outros podem ser citados como as diferentes bibliotecas para desenho dos componentes gráficos e a falta de bom senso dos desenvolvedores em implementar os recursos de acessibilidade em suas aplicações.

Dentre as principais características de um leitor de tela, uma das mais importantes é a voz, já que a qualidade da dicção da voz é primordial para o relacionamento com o usuário. Tal recurso é de responsabilidade do sintetizador de voz. Claro que, quanto mais semelhante à voz humana melhor. Turing (1950) diz que um computador inteligente é aquele que pode enganar alguém se passando por um humano. No caso dos leitores de tela, a solução não é deixar o computador mais inteligente, mas deixá-lo mais amigável, tentando imitar perfeitamente a voz humana, porém, para ser entendida, a voz não precisa ser perfeita. Em alguns casos, os usuários mais experientes utilizam a síntese de voz em uma velocidade de leitura muito alta, inatingível por um ser humano.

De um ponto de vista financeiro, um dos grandes obstáculos para a aquisição de um leitor de tela é o seu alto custo. Segundo Parente (2006), a explicação para isto deve-se ao fato de que uma pessoa com visão normal, em tese, nunca utilizará este tipo de aplicativo, logo a procura por esse *software* é limitada, acarretando o aumento do seu custo.

Para se ter uma idéia, o preço do *software* JAWS[1] para consumidores dos Estados Unidos é de 895 dólares, para a sua versão *Standard*, e de 1095 dólares para a sua versão *Professional*. O mesmo acontece com o Window-Eye[2], cujo preço de mercado também fica em 895 dólares. Felizmente, não existem apenas soluções comerciais, e o mercado oferece soluções em *software*

[1] http://www.freedomscientific.com/products/fs/jaws-product-page.asp
[2] http://www.gwmicro.com/Window-Eyes/

livre que se equiparam com os aplicativos comerciais[3].

2.2 Objetivos e Aplicações de um Leitor de Tela

O principal objetivo deste tipo de recurso é auxiliar, através de informações oralizadas, as pessoas com deficiência visual que desejam utilizar o computador. No caso das pessoas cegas, elas necessitam obrigatoriamente deste mecanismo para interagir com um computador, já que não têm a possibilidade da interação com o ambiente gráfico dos sistemas. No caso das pessoas com deficiência visual menos severa, o leitor de tela também auxilia minimizando o esforço demandado para o reconhecimento de textos existentes na tela.

Embora o objetivo principal de um leitor de tela seja auxiliar pessoas com deficiência visual, na verdade, eles também são recursos úteis para outros objetivos. Por exemplo, pode ser usado como leitor de textos[4], ou como leitor de e-mails[5], ou para ajudar uma pessoa que teve algum tipo de paralisia a se comunicar[6], ou ainda, auxiliar na interação com pessoas em um terminal de atendimento, além de várias outras utilidades.

2.3 O Estágio Atual no Desenvolvimento de Leitores de Tela

Atualmente, existe uma tendência tecnológica na qual os leitores de telas para ambientes gráficos trabalham em conjunto com interfaces de acessibilidade, como a AT-SPI [Foundation 2008], o Java Access Bridge [Oracle 2011] e o Microsoft Active Accessibility (MSAA) [Microsoft 2011]. A grande vantagem é que estas interfaces auxiliam o processo de monitoramento e captura de textos nos sistemas operacionais e nas aplicações. O objetivo destas ferramentas é evitar que o desenvolvedor construa módulos para inúmeras aplicações, a fim de buscar em cada uma delas as informações que precisam ser lidas. Esta solução possibilita

[3]Os valores dos aplicativos JAWS e Window-Eye são referentes ao mês de agosto de 2011.

[4]O Loquendo (http://www.loquendo.com/en/products/text-to-speech/), o NaturalReader 10.0 (www.naturalreaders.com/index.htm) e o Ultra Hal Text-to-Speech Reader (zabaware.com/reader/) são apenas alguns exemplos de leitores de texto gratuitos que podem executar a leitura de textos. A diferença entre estes tipos de leitores e um leitor de tela, é que estes apenas falam o texto que foi direcionado para eles, portanto não funcionam puramente como uma tecnologia assistiva.

[5]O aplicativo E-mail Talker (http://www.scosoft.com/what-is-email-talker.htm) é um exemplo de leitor de texto que trabalha com o email, lendo os emails da caixa de entrada e avisando por voz quando estes são recebidos.

[6]Como no caso do físico e cosmólogo britânico Stephen Hawking, que se comunica através de um sintetizador de voz, pois seu corpo é quase todo paralisado (http://www.hawking.org.uk/index.php?option=com_content&view=article&id=51&Itemid=55).

que o desenvolvedor crie um módulo que se comunica com a AT-SPI, por exemplo, e então a AT-SPI tem a responsabilidade de se comunicar e obter informações do sistema operacional e das aplicações, retirando esse esforço do desenvolvedor do leitor.

Os leitores de tela estão distribuídos em vários tipos de arquiteturas. No caso da Web (onde o leitor pode ser desenvolvido em qualquer tipo de sistema operacional), existem poucas soluções, porém alguns leitores de tela convencionais conseguem manusear o conteúdo Web, assim como vários navegadores se preocupam com a acessibilidade, tornando esta área menos crítica. Já nos sistemas operacionais Windows e Linux existe uma maior quantidade de soluções, com o Windows se destacando com um grande número de soluções, sendo que nas versões mais atuais desses sistemas eles já contam com leitores de tela em suas instalações padrões[7].

2.4 Tipos de leitores de telas

Segundo Raj (2008) os leitores de tela podem ser classificados em três tipos: CLI (*Command Line Interface*), GUI (*Graphical User Interface*) e *Web-based*, além desses três tipos, inserimos também, nesta seção, a descrição dos sistemas chamados *self-voicing*. Os três primeiros tipos se dividem principalmente quanto ao ambiente de execução dos leitores. Cada um desses tipos também define, parcialmente, as técnicas utilizadas no desenvolvimento de leitores de tela.

Os aplicativos do tipo *self-voicing*, por sua vez, podem ser executados em qualquer tipo de ambiente. Os *self-voicing* são capazes de produzir sons por si mesmos, sem a necessidade de um leitor de tela.

Cada um desses tipos de leitores têm suas características particulares, mas todos seguem o padrão de transformar os textos da tela do computador em voz, mesmo que o processo para um tipo de leitor seja totalmente diferente do outro.

Já podemos adiantar que o leitor de tela desenvolvido nesse trabalho se encaixa no tipo GUI e subtipo *Accessibility API*, com a explicação de cada tipo e com as seções seguintes contendo as informações sobre o leitor, a categorização neste tipo de leitor fica mais clara.

Na sequência, são apresentados as principais características dos tipos de leitores de tela.

[7] O Windows 7 possui o Narrator que é um leitor de tela simples, enquanto que o Linux Ubuntu, por exemplo, já há algum tempo, traz o Orca em sua instalação padrão.

2.4.1 Tipo CLI

As interfaces de linhas de comandos (CLI) foram umas das primeiras formas de interação com o usuário, sendo utilizadas nos primeiros sistemas operacionais. O CLI é um tipo de interface somente texto, que aguarda o usuário digitar um comando e então o executa, e mostra os resultados também em modo texto. Os exemplos mais conhecidos desses tipos de sistemas são o MS-DOS e o terminal do Linux. Como todos os programas rodavam nesta interface, um leitor de tela para este sistema devia de alguma forma mapear as entradas e as ações do usuário neste sistema, a fim de transformá-las em voz. Para executar a leitura, o leitor necessita mapear diretamente os caracteres lidos para um *buffer* na memória, assim como a posição do cursor no texto. A partir das ações do usuário e com base no *buffer*, o leitor executa a leitura [Raj 1998]. Os exemplos de leitores que executam sobre um sistema CLI, ou implementam um CLI internamente são o JAWS e o DOSVOX.

2.4.2 Tipo GUI

Os sistemas operacionais evoluíram a ponto de criarem a chamada interface gráfica do usuário (GUI), que pode ser encarada como a parte visual dos sistemas operacionais. O ambiente gráfico dos sistemas operacionais é o *software* responsável pela interface gráfica do usuário, a fim de tornar possível a utilização do computador por meios gráficos, desenhando na tela botões, janelas, ícones, etc.

Como exemplos de ambientes gráficos atuais, temos o Windows Aero, o GNOME e o KDE. De um modo geral, a utilização desse recurso melhorou muito a interação do sistema com a maioria dos usuários, mas também, tornou mais complexo o processamento dos leitores de tela e o cotidiano dos usuários de leitores. Segundo Parente (2006) a simples transformação dos elementos da tela para um fluxo de fala não resulta em uma demonstração de áudio efetiva do sistema. Em vez disso, o leitor deste ambiente obriga os usuários a pensar e interagir com uma aplicação em termos de conceitos gráficos, que faz pouco ou nenhum sentido no domínio do áudio.

Em relação aos leitores de tela, a dificuldade está em que a GUI é a composição de elementos gráficos desenhados na tela, e sendo assim, não existe uma representação puramente textual do conteúdo da tela [Raj 1998].

O leitor de tela nesse tipo de sistema é um aplicativo que se comunica com todos os outros aplicativos do sistema, não podendo interromper o fluxo de execução natural do sistema, nem dos aplicativos. Neste conceito, a abordagem destes leitores de tela que executam "por baixo" do sistema, permitindo o acesso a uma aplicação sem ter uma cooperação direta é uma vantagem, porém, também é considerado seu principal defeito [Raman 1996].

Na tentativa de solucionar os problemas computacionais impostos pelos ambientes gráficos, os desenvolvedores de leitores de tela utilizam duas principais técnicas para executar a leitura. Uma breve descrição caracterizando as técnicas chamadas de *Off-screen models* e *Accessibility APIs* são mostradas a seguir.

Off-screen models

Nesta técnica, cria-se um modelo fora da tela (em memória), que de alguma maneira representa o sistema e seu ambiente gráfico. Uma maneira de fazer este modelo é monitorar as mensagens trocadas entre aplicações e sistema operacional. Por exemplo, interceptando uma mensagem para desenho de um botão, o leitor consegue representá-lo na memória e quando necessário ler o seu conteúdo. O desperdício de memória aqui é evidente, já que o conteúdo da tela deve de alguma maneira estar representado na memória, e o que não é tão evidente é a dificuldade que se tem em monitorar os sistemas e suas aplicações, a fim de fazer o mapeamento em memória que represente corretamente o sistema [Raj 1998].

Accessibility API

Como dito, as interfaces gráficas trouxeram problemas para os desenvolvedores de leitores de tela. Por este motivo, os desenvolvimentos das APIs de acessibilidade começaram a ser uma solução para esta área. A partir daí, criou-se uma maneira diferente de interagir com os sistemas, a fim de obter as informações textuais acerca destes. Os desenvolvedores que utilizam essas APIs não necessitam criar um modelo fora da tela do sistema, por que as informações que eles necessitam dos componentes gráficos (menus, ícones, botões, etc.) estão disponíveis pelos comandos da própria API. Estas APIs devem ser suportadas por todos os aplicativos que rodam no sistema, dessa maneira, o desenvolvedor pode criar um código mais genérico (não se preocupando com as especificidades de cada aplicativo) e enxuto (não criando métodos para cada aplicativo).

Por outro lado, se um aplicativo não suporta a API que o desenvolvedor estiver usando, o desenvolvimento deve levar em conta maneiras para tratar esta dificuldade, por que aplicativos que não suportam a API simplesmente não vão dar suas informações. Uma das opções é criar um modelo fora da tela para o aplicativo que não suporta a API, ou ainda, optar por implementar outra API que o aplicativo tenha suporte.

2.4.3 Tipo *Web-based*

Os leitores de tela baseados em Web são soluções voltadas para atender usuários que desejam navegar por páginas na Internet por meio de um navegador ou sobre outro aplicativo que roda sobre a Web. Essa área é relativamente nova quando comparado aos outras áreas do desenvolvimento de leitores de tela, tendo como uma das principais motivações prover acessibilidade a sites da Internet em computadores públicos onde não é possível a instalação de programas [Raj 1998]. Um exemplo de *software* nesta área é o WebAnywhere[8], desenvolvido pela *University of Washington* que é um leitor de telas *web-based* que funciona diretamente no navegador, necessitando apenas que o usuário acesse a página do mesmo.

2.4.4 Tipo *Self-voicing*

Enquanto que nos três casos anteriores, o processamento de leitura de tela é feito por um aplicativo externo aos demais aplicativos, no *self-voicing* esse processamento ocorre dentro da própria aplicação que deseja adicionar leitura em suas funções. Devido ao próprio tipo e características, a maioria desses sistemas são dos tipos CLI e *web-based*.

Esses sistemas eliminam a necessidade do uso de um leitor de tela, pois são capazes de gerar fala internamente, para realizar a interação com o usuário. No entanto a fala gerada só tem relação às ações executadas no aplicativo, e as ações externas ainda necessitam de um leitor de tela.

Em alguns casos, como o sistema consegue gerar a fala, e o desenvolvedor tem o conhecimento de toda a execução de seu *software*, existe a tentativa de implementar várias aplicações (como editor de texto, navegador Web, leitor de e-mails, etc.) no mesmo sistema, todos com recurso de fala, construindo assim, uma aplicação parecida com um sistema operacional, onde

[8]http://webanywhere.cs.washington.edu/

o objetivo é suprir todas as necessidades do usuário através de uma interface de som. Este tipo de desenvolvimento ocorre, por exemplo, no DOSVOX.

Em outros casos, um *software* pode trazer consigo uma extensão para produzir a voz, caso que se tornou uma tendência entre os navegadores que desenvolveram uma extensão (ou um *plug-in*) para produzir a leitura das páginas HTML, o que também elimina a necessidade de um leitor de tela. Caso da extensão de voz do Opera[9] e do Fire Vox[10], que é uma extensão de leitura de tela para o navegador Firefox[11].

2.5 Trabalhos Correlatos

Nesta seção são apresentados alguns dos principais leitores de telas disponíveis para uso atualmente. A escolha destes *softwares* se deve principalmente pela maior quantidade de usuários, assim como melhores recursos e documentações.

2.5.1 Dosvox

O DOSVOX[12] é um sistema desenvolvido desde 1993 para várias versões do Windows e pode ser classificado como pertencente ao tipo CLI, já que manipula uma interface de texto semelhante ao MS-DOS.

O sistema vem acompanhado de várias ferramentas, entre elas, um leitor e editor de texto, um impressor e formatador Braille, alguns programas para internet como Correio Eletrônico, Acesso a Homepages, Telnet e FTP, além de jogos, e muitos outros que somam um total de 80 aplicativos.

O sistema pode ser adquirido gratuitamente pelo próprio site do Intervox baixando algum dos instaladores disponíveis. As últimas versões disponibilizadas são a 4.2 (experimental) que tem 340 *megabytes* e a 4.1 (estável) que tem 112 *megabytes*.

O DOSVOX ainda conta com uma versão para Linux, chamada LINVOX[13], que não é mais que o próprio DOSVOX emulado pelo programa Wine, em uma versão do Kurumin. O LINVOX

[9] http://www.opera.com
[10] http://www.firevox.clcworld.net/about.html
[11] http://www.mozilla.com/pt-BR/firefox/
[12] O texto encontrado nesta seção é baseado no texto contido no site da Intervox na página do próprio DOSVOX (http://intervox.nce.ufrj.br/dosvox/index.htm).
[13] http://equipe.nce.ufrj.br/gabriel/linvox/

ainda pode ser instalado manualmente pelo usuário.

A comunicação na interface de usuário do DOSVOX é mais simples que em outros leitores. Ao contrário dos outros programas, o DOSVOX não apenas lê o que está escrito na tela, mas estabelece uma interação mais amigável com o usuário, ficando evidente em alguns pontos da execução, onde o sistema fala às possíveis ações que o usuário poderá executar naquele ponto com o intuito de ambientar o usuário ao sistema. Este tipo de ajuda, o usuário dificilmente terá em um leitor de tela para ambientes gráficos, pois em qualquer ponto da execução, o sistema é incapaz de prever os próximos passos dos usuários, já que esses passos podem atingir um número bastante grande nas interfaces gráficas.

A leitura depende da ferramenta que o usuário esteja utilizando, mas sempre é satisfatória, lendo praticamente todos os textos que aparecem na tela e auxiliando o usuário nos passos que ele pode executar (oferecendo ajuda em todos os aplicativos, por exemplo).

2.5.2 Orca

O aplicativo Orca[14] é um sistema que além de um leitor de tela conta com outras funções, como ampliação de tela e braile. Pertence ao tipo de leitores de tela GUI, pois executa na interface GNOME do Linux. Ganhou um maior reconhecimento entre os usuários, no momento que o sistema operacional Linux Ubuntu adotou-o como leitor de tela padrão. Assim sendo, todas as versões do sistema operacional vinham acompanhados do Orca, pronto para uso.

É um *software* livre (licença LGPL) e pode ser baixado gratuitamente da página do GNOME. Além do *download* do instalador, a página contém uma descrição sobre o processo de instalação do Orca e dos programas auxiliares que devem ser instalados para o Orca funcionar corretamente.

É um programa de código aberto que suporta a interface AT-SPI, ou seja, é projetado para trabalhar com aplicações e kits de ferramentas que também a suportem. Foi desenvolvido com a linguagem de programação Python e utiliza o sintetizador de voz eSpeak, ou qualquer outro configurável pelo usuário. Como o Orca é parte da plataforma GNOME, ele também tem versões nos sistemas operacionais Open Solaris, Fedora, e Ubuntu.

Uma das vantagens desse sistema é o usuário poder customizar a aplicação, através das

[14]O texto contido nesta seção é uma adaptação dos textos das páginas do Orca no site do GNOME (http://live.gnome.org/Orca.pt_BR e http://live.gnome.org/Orca).

funcionalidades de leitura de tela, ampliação e braile para atender as suas necessidades.

A leitura de tela do Orca consegue atender as necessidades dos usuários de uma maneira bem satisfatória, sendo que um dos únicos problemas encontrados na execução do *software* é que em alguns momentos o sistema cria uma fila de eventos e acaba demorando para falar todos os eventos desta fila. Outro pequeno problema verificado que ocorre poucas vezes é que o sistema executa a fala de algum componente que não é visível ao usuário, isto pode confundir a navegação no sistema.

2.5.3 NVDA

O NVDA[15] (*NonVisual Desktop Access*) é um leitor de tela livre de código aberto para o sistema operacional Windows XP (ou versão posterior). Também é capaz de prover a função braile e para executar a síntese de voz utiliza o *software* eSpeak. O *software* inclui síntese para 20 idiomas e conta com versões para instalação e para viagem (o aplicativo pode ser carregado no *pen-drive* e ser executado em qualquer computador com Windows).

Na página do NVDA pode-se baixar as versões de instalação e de viagem gratuitamente, mas caso o usuário deseje, existe uma maneira para fazer doações ao projeto na própria página. Tanto a versão de instalação como a de viagem tem cerca de 12 *megabytes*.

É implementado utilizando a linguagem de programação Python e para se comunicar com o sistema utiliza vários métodos como a implementações das APIs como MSAA (Microsoft Active Accessibility)[Microsoft 2011], IAccessible2 [Foundation 2009], Java Access Bridge [Oracle 2011], além das interfaces específicas de cada aplicativo.

O NVDA possui uma ótima leitura de tela, lendo praticamente todas as ações do usuário no sistema, além de ler os textos da tela apenas com a movimentação do mouse, colocando-o sobre o texto a ser lido.

[15] O texto encontrado nesta seção é um resumo dos textos encontrados na página do NVDA (http://www.nvda-project.org/) e de [Sonza, Mello e Moro 2009].

2.5.4 JAWS

JAWS[16] (Job Access With Speech) [17] é um leitor de tela comercial para uso no Windows, que também funciona no DOS. É desenvolvido pela Freedom Scientific, uma empresa que comercializa produtos voltados para pessoas com deficiência visual.

Em sua página pode-se encontrar um *link* para aquisição do JAWS, onde a sua versão *Standard* custa cerca de 895 dólares e a versão *Professional* custa cerca de 1.095 dólares. Na mesma página é possível encontrar algumas versões de demonstração do aplicativo, que podem ser baixadas gratuitamente, essas versões tem cerca de 100 *megabytes*.

Praticamente todas as versões do Windows são suportadas pelo *software* (desde a versão 95 até o 7). A voz é sintetizada pelo *software* próprio Eloquency, mas o sistema também pode ser alterado para utilizar um sintetizador externo. Com o sintetizador próprio pode-se optar por vários idiomas, incluindo o português.

O JAWS interage com o sistema Windows ditando às ações do usuário. Conta com uma série de teclas de atalho que auxiliam o usuário a manusear o computador. Para aumentar a interação com o usuário, o sistema ainda permite que o usuário configure o leitor de acordo com o aplicativo que ele estiver utilizando. Também permite que o usuário mude a intensidade da leitura (os tipos de leitura são: Realçado, Tudo ou Nenhum). Uma funcionalidade interessante é o auxilio por meio de voz no momento da instalação do sistema.

2.5.5 Considerações Finais

Como em muitos outros casos, escolher o melhor *software* da área não é uma tarefa fácil, e neste caso a escolha depende de alguns fatores além das características e funcionalidades dos próprios leitores. Por exemplo, se um usuário necessita usar certo sistema operacional obviamente os leitores de tela de outros sistemas já serão descartados em sua escolha.

O DOSVOX é o mais aconselhável para as pessoas que tiveram pouco ou nenhum contato com o computador, pois a interação com o usuário no aplicativo é a melhor entre os leitores descritos, já que ele trabalha com uma interface mais simples, porém mais interativa. O DOSVOX cria a sua própria interface de linha de comando, diferente dos outros leitores que executam so-

[16] A descrição contida nesta seção é um resumo das informações do JAWS contido em [Sonza, Mello e Moro 2009].

[17] http://www.freedomscientific.com/products/fs/jaws-product-page.asp

bre a interface gráfica de um sistema operacional, assim o DOSVOX não necessita que o usuário tenha conhecimento sobre um sistema operacional específico, mas acaba obrigando o usuário a aprender e utilizar a sua interface.

O NVDA por sua vez é o ideal para usuários do ambiente Windows, já que é gratuito (diferente do JAWS) e tem uma qualidade semelhante a outros aplicativos comerciais. Já o Orca é o indicado para usuários do Linux, a leitura contem alguns inconvenientes, mas o mesmo conta com uma série de funcionalidades como braile e ampliação de tela que podem em alguns casos ajudar o usuário ou em outros casos apenas o atrapalhar se este desejar apenas a leitura de tela.

Apesar de citarmos os principais leitores de tela, o leitor desenvolvido não concorre diretamente com eles, pois cada um desses leitores executa sobre um sistema diferente. O Orca é que trabalha no mesmo ambiente que o leitor aqui desenvolvido, porém o Orca é praticamente uma suíte de aplicativos de acessibilidade (ampliador, braille, leitor, etc.). A ideia do leitor desenvolvido é a criação de uma forma mais simples para que usuários e principalmente outros aplicativos possam fazer uso de um leitor de tela, que diferente do Orca, não possui nenhuma outra funcionalidade que não seja a leitura de tela.

Capítulo 3

Processamento de voz em Leitores de Tela

O processamento de voz consiste em vários tipos de aplicações que de alguma forma processam a fala, dentre essas aplicações se encontram o processamento para reconhecimento de voz e o da síntese de voz [Ramachandran e Mammone 1995]. Na construção de um leitor de tela é indispensável a implementação ou a adoção de um sistema de síntese de voz, pois este tem o papel de criar a voz que será utilizada na leitura dos textos.

A seguir, serão apresentadas as principais características da síntese e do reconhecimento de voz. Vale a ressaltar que mesmo contendo uma seção de reconhecimento de voz, não nos estenderemos nesse assunto, uma vez que não estamos trabalhando com este recurso, apresentamos aqui, apenas para contextualizá-lo na área mostrando a diferença das técnicas, tirando assim, qualquer dúvida que possa haver acerca destas duas técnicas. E quanto a síntese de voz, o nosso sistema também não implementa esta técnica, mas utiliza de um *software* externo para executar a síntese, o sintetizador de voz eSpeak [eSpeak 2011].

3.1 Síntese de Voz

A síntese de voz pode ser entendida como o processo artificial de transformar informações textuais em enunciados orais. Um sistema de síntese de voz converte linguagem normal em fala, outros sistemas transformam representações linguísticas, como transcrições fonéticas em fala [Allen, Hunnicutt e Klatt 1987].

3.1.1 Histórico

O primeiro indício de tentativa de reproduzir a fala humana aconteceu no ano de 1779 em St. Petersburg com o professor russo Christian Kratzenstein explicando as diferenças fisiológicas entre as vogais, e mostrando um equipamento construído para a produção artificial destas vogais [Lemetty 1999]. A partir daí, vários outros conseguiram até certo ponto reproduzir a voz humana, atualmente o processo de síntese de voz é mais conhecido pelos *softwares* sintetizadores de voz capazes de gerar sons muito próximos aos da fala de uma pessoa.

Talvez o momento em que a síntese de voz, assim como o reconhecimento de voz, ganhou maior atenção do público foi quando Arthur C. Clarke em 1968 escreveu o livro "2001: Uma Odisseia no Espaço", que foi transformado em filme pelas mãos de Stanley Kubrick se tornando um marco na ficção científica. Na obra, HAL 9000 é um computador com inteligência artificial bastante avançada, capaz de conversar com os tripulantes de sua nave, identificando os dizeres destes e falando naturalmente como um humano. A partir daí, a visão de um computador inteligente e a visão de relacionamento homem-máquina passam a ser relacionadas ao reconhecimento e a síntese da voz humana. No futuro, essa nova visão de relacionamento entre homem e máquina seria importantíssima para as tecnologias assistivas.

No início dos anos 80 surgiram os primeiros sintetizadores comerciais [Castro et al. 2004] e hoje os sintetizadores estão ainda mais em destaque, como no caso do sistema operacional móvel Android que adicionou suporte para síntese de voz na sua versão 1.6 [Trivi 2011].

3.1.2 Metodologias de síntese

O processo de sintetização é dividido em duas partes básicas: o Processador de Linguagem Natural e o Processador de Sinais Digitais.

O Processador de Linguagem Natural (frequentemente chamado de normalização de texto, pré-processamento, ou *tokenization*) recebe uma mensagem textual e a converte numa série de símbolos fonéticos [Schroeder 2001], [Körting, Costa e Silva 2005]. Enquanto que o Processador de Sinais Digitais recebe a saída do módulo anterior, ou seja, a representação simbólica linguística ou a representação fonética e a partir daí é responsável pela formação do som [Lemetty 1999].

Existem sub-funções que devem ser realizadas pelo processador de linguagem natural, são

elas: Pré-processador, Analisador Contextual, Transformador Sintático-Prosódico e o Conversor letra-fonema. Alguns sistemas podem diferir sobre essas sub-funções, mas o habitual é que todos assemelham-se, expandindo alguma função ou unindo duas em uma. O pré-processador é responsável por transformar números, abreviações e tudo mais em texto por extenso. O analisador deve considerar cada palavra e identificar o seu contexto para possibilitar, se for o caso, a mudança na entonação. O transformador separa frases e contextos examinando a estrutura do texto, e finalmente, o conversor letra-fonema determina a transcrição fonética de cada palavra [Körting, Costa e Silva 2005]. Lemmetty (1999), classifica os sintetizadores em três grupos de acordo com a sua metodologia. Eles são a Síntese por Concatenação, a Síntese Articulatória e a Síntese por Formantes.

Quando se executa arquivos de som pré-gravados, temos a Síntese por Concatenação, que atinge atualmente a maior naturalidade e inteligibilidade. O sistema trabalha com uma base de dados com os sons pré-gravados onde cada um é associado a um fonema, e então os sons são concatenados para gerar a fala [Castro et al. 2004]. Este método provavelmente é a maneira mais fácil de produzir fala, no entanto a síntese gera apenas uma voz e exige muito mais capacidade de memória que os outros métodos [Lemetty 1999].

Existem três tipos de síntese por concatenação: por domínios específicos, por seleção de unidade e por difones (combinação de dois fones). A síntese por domínios específicos possui um pequeno conjunto de palavras para um tema específico. A síntese por seleção de unidades utiliza-se de um banco de dados formado por fonemas, difones, meios-fonemas, frases, etc. para determinar o melhor ajuste para a frase a ser sintetizada. A síntese por difones gera a fala através de um banco de dados que contem todos os possíveis difones do idioma [Parente 2009].

Nesses três tipos de síntese são utilizados sons pré-gravados, o que altera a qualidade da voz é a duração desses sons, nisto a síntese por difones é a mais prejudicada, já os outros tipos tiram proveito dessa característica, pois podem ter frases inteiras gravadas. Por outro lado, a síntese por difones é a mais flexível, pois como os sons gravados são menores, eles são capazes de produzir uma quantidade maior de palavras.

Outra técnica é a Síntese Articulatória, que tenta simular o aparelho vocal humano tentando modelar os órgãos da fala (como cordas vocais e língua, por exemplo) e os processos que ocorrem nesses órgãos [Lemetty 1999]. Potencialmente este é o método mais satisfatório

para produzir uma voz sintética de alta qualidade, porém ainda não atingiu o mesmo nível de sucesso das outras técnicas [Donovan 1996], além disso, o seu custo computacional é muito elevado [Körting, Costa e Silva 2005] e não recebe a mesma atenção dos métodos que utilizam sinais.

A última técnica combina frequências utilizando um conjunto de ressonadores, geralmente ligados em paralelo, ou em cascata, ou principalmente em uma combinação desses [Lemetty 1999]. Tais ressonadores processam a entrada formando uma saída equivalente, que é semelhante a um som vocal. Somando todas as saídas provenientes de vários tipos de entradas diferentes, produz-se um som semelhante a um fonema. Devido ao fato desta técnica modelar apenas a fonte de som e as frequências formadoras é chamada de Síntese por Formantes [Castro et al. 2004]. Um formante é constituído por picos de intensidade em uma certa forma de onda. É possível utilizar esses picos para distinguir as ondas de fala entre si. A síntese por formantes se utiliza dessa característica das ondas para obter formas sintéticas aplicando a elas certas regras de produção [Parente 2009]. Tal técnica provavelmente é a mais utilizada nos últimos anos, sendo uma de suas vantagens, a produção de uma infinidade maior de sons que torna este método muito mais flexível que os outros [Lemetty 1999].

3.1.3 Sintetizador de Voz

Como explicado anteriormente a síntese de voz é o processo de produção artificial da voz humana e um sistema computacional construído para tal finalidade é denominado sintetizador de voz.

O aspecto mais relevante em um sintetizador de voz é a qualidade da dicção de voz. Esta qualidade de fala de um sintetizador está ligada a percepção que temos do som gerado. A qualidade depende principalmente da metodologia utilizada na síntese, no caso de um sintetizador que usa fonemas pré-gravados, a qualidade dependerá do tamanho destes, ou seja, caso os fonemas gravados sejam pequenos a dicção será prejudicada, porém se frases inteiras são gravadas a qualidade aumenta, mas nem todos os sintetizadores trabalham com fonemas gravados. A gravação de frases prontas em leitores de tela para ambientes gráficos, não é muito utilizada, pois a utilização do sistema é muito diferente para cada usuário e os leitores de tela nesses sistemas não passam de um *software* que executa "por baixo" do sistema não interferindo na utilização

do mesmo. Outros leitores, sim, têm uma interface própria e fazem o uso de frases gravadas, o que, como dito, melhora muito a compreensão. Na seção seguinte, pode-se encontrar mais detalhes sobre as metodologias de geração de fala pelos sintetizadores de voz.

Um ponto fraco dos sintetizadores em geral é que eles são produzidos para o público americano, logo contam com uma dicção de voz na língua inglesa, porém alternativas em outras línguas existem e alguns sintetizadores contam com um conjunto de fonemas gravados em outras línguas, mas muitas vezes sem a mesma qualidade que o original em inglês. Outra característica referente à fala dos sintetizadores é a voz mecanizada que pode identificar um sintetizador facilmente.

Muitos sistemas operacionais têm incluído capacidade de síntese de voz desde o início da década de 1980, porém a qualidade da maioria dos *softwares* ainda é baixa, se levarmos em conta a proximidade com a voz humana, no entanto é possível decifrar facilmente o que o computador está "falando". O Ubuntu já há algum tempo, traz no seu sistema um sintetizador de voz chamado eSpeak sendo este o sintetizador escolhido para fazer parte do leitor de tela aqui desenvolvido.

Atualmente, existem sintetizadores de voz para basicamente todos os sistemas computacionais, e o seu principal foco é a produção de fala para os leitores de tela que utilizam dela para a interação com o usuário.

Entre as principais características a se melhorar nos sistemas de síntese de voz, Braga (2007) destaca a integração de emoções, o estilo, a atitude, e o aperfeiçoamento da naturalidade da voz sintética. Em outras palavras, as melhorias que devem ser realizadas são uma tentativa de fazer a voz ficar mais agradável ao ouvido humano, já que a maioria dos sintetizadores ainda tem aquela voz mecanizada que não é muito agradável.

O Uso de Sintetizadores

O uso de sintetizadores de voz tem sido um recurso bastante explorado na implementação dos mais diversos tipos de aplicações, especialmente em sistemas de auto-atendimento, tais como *call centers* e caixas eletrônicos[1]. Sintetizadores de voz também têm sido utilizados na

[1]O aplicativo de síntese de voz CPqD Texto Fala [CPqD 2011], foi criado justamente para ser utilizado por caixas eletrônicos, gerando fala com uma boa qualidade de uma forma flexível aos sistemas através de arquivos de texto.

produção de livros falados ou para fins de ensino de língua estrangeira (onde um texto é repetido inúmeras vezes, com o intuito de que o aluno escute e repita a pronúncia corretamente). Talvez a utilização mais popular dos sintetizadores de voz seja em sistemas de navegação, como os aparelhos GPS, que funcionam como um co-piloto, dizendo onde e quando o motorista deve fazer uma conversão, por exemplo. Outro ponto, onde a cada dia os sintetizadores ganham mais espaço é nos terminais de atendimento, em bancos ou pontos turísticos, a fim de fornecer informações para os turistas ou clientes em forma de voz [Coelho et al. 2004].

3.2 Reconhecimento de voz

No reconhecimento de voz, o processo a ser executado é o "inverso" do da síntese de voz. Em outras palavras, a partir de um sinal de voz identificam-se as unidades que fazem sentido e que permitam a construção de estruturas lógicas em uma determinada língua [Braga 2007].

As pesquisas relacionadas ao reconhecimento da fala tiveram seu início no final dos anos 40 e com os rápidos avanços na computação, vêem alcançando inúmeros progressos [Englund 2004]. A utilização de sistemas com reconhecimento de voz tem sido cada vez mais intensificada. A seguir demonstramos alguns exemplos de sistemas de reconhecimento de voz.

Recentemente, o site YouTube [2] disponibilizou para os seus usuários um novo recurso. O site colocou em seu player de vídeo um botão para ativar a transcrição de áudio, a partir do reconhecimento de fala. Segundo o próprio YouTube, "Transcrever áudio é um serviço experimental que usa tecnologias de reconhecimento de fala do Google para fornecer legendas automatizadas para o vídeo".

Outro exemplo é o navegador Opera [3]. Através de comandos de voz permite que o usuário interaja com o navegador. Os comandos são muito variados, permitindo ao usuário comandar o funcionamento do sistema por meio da voz. Por exemplo, o usuário pode falar "*New Page*" e uma nova página vazia se abrirá, também pode ser dito "*Open Link*" para que o *link* selecionado se abra. Além do reconhecedor de voz o Opera também conta com um sintetizador de voz, no caso do usuário ditar o comando de voz "*Speak*", o texto selecionado será lido [4].

Estes tipos de comandos de voz também são utilizados por muitos outros aplicativos, como

[2] www.youtube.com
[3] www.opera.com.br
[4] http://help.opera.com/Windows/11.00/en/voice.html

jogos e em muitas plataformas, como o celular, por exemplo. Em alguns celulares ditando o nome do contato, a discagem para este contato é feita automaticamente sem a necessidade de digitar os números. Em jogos, o mais comum é que os comandos de fala façam alguma ação no jogo, por exemplo, mandar seus companheiros (Inteligência Artificial do jogo) atacarem o inimigo naquele instante.

Outras utilidades de um reconhecedor de texto podem estar presentes em equipamentos que permitem comandos, como automóveis e celulares e em terminais de atendimento, como aqueles presentes em bancos, pontos turísticos, shoppings e aeroportos. A técnica também pode ser utilizada para o aprendizado de línguas estrangeiras, verificando a pronuncia do aluno [Coelho et al. 2004].

Quanto ao desenvolvimento de um reconhecedor de voz, uma das tarefas mais importantes é a comparação entre padrões. Existem alguns métodos conhecidos para tratar essa tarefa que são utilizados no processo de reconhecimento, são eles a Variação Temporal Dinâmica (*Dynamic Time Warping*), os Modelos Ocultos de Markov (*Hidden Markov Models*) e as Redes Neurais Artificiais [Valiati 2000]. Esses métodos já foram utilizados em testes e podem ser usados tanto individualmente como em conjunto no processamento de reconhecimento de fala [Deller, Proakis e Hansen 2000], [Rabiner e Schafer 1978].

Um sistema de reconhecimento de voz tem muitos problemas para tratar durante o seu desenvolvimento como diferentes interlocutores, palavras sensíveis ao contexto, além da entonação e do timbre da fala, que podem mudar a interpretação e a linguagem escrita, que necessita de pontuação, que não é tão aparente na fala. Um meio para solucionar vários desses problemas é a utilização de técnicas de inteligência artificial, principalmente de modelos estatísticos que podem ser aplicados, por exemplo, para desambiguação e incremento na qualidade do reconhecimento.

Capítulo 4

Especificação e Implementação do Leitor de Tela

Os principais aspectos envolvidos na fase de desenvolvimento do leitor de tela são descritas neste capítulo, desde os requisitos de *software* exigidos até as questões relacionadas a implementação do leitor de tela.

A principal tarefa a ser desenvolvida neste trabalho é a construção de um leitor de tela para o ambiente GNOME do Linux Ubuntu. Este fato, já define previamente algumas linguagens, ferramentas e aplicativos a ser utilizadas, pois exige a necessidade de compatibilidade com este sistema. Logo, isto levou a escolha dos principais recursos utilizados neste projeto, como o Accerciser (*software* para Linux), o eSpeak (*software* presente nas distribuições Ubuntu), o Python (linguagem de programação suportada por sistemas Linux), a AT-SPI (desenvolvida pelo GNOME), etc. Todos os recursos, devidamente compatíveis com o sistema foram utilizados para garantir que os requisitos básicos fossem atendidos no produto final ou durante a elaboração do leitor.

4.1 Requisitos de *Software*

Os principais requisitos que o leitor deve satisfazer como já foi dito, são executar sobre o ambiente GNOME do sistema operacional Ubuntu e possuir as principais funcionalidades existentes nos leitores de tela semelhantes que atualmente trabalham com ambientes gráficos. Dentro do primeiro requisito ainda, podemos citar a possibilidade de fazer com que qualquer outro *software* possa utilizar o leitor desenvolvido como uma ferramenta externa de leitura de tela, comportamento semelhante a um *plug-in*.

Essas primeiras características que o leitor de tela deve atender já o encaixam na categoria GUI de leitores de tela, ou seja, ele deve executar sobre uma interface gráfica de usuário. A categoria GUI tem duas divisões, sendo que o leitor aqui desenvolvido se encaixa na divisão Accessibility APIs, pois utiliza uma interface de acessibilidade (AT-SPI).

Em relação aos elementos que o leitor deve ler estão os textos dos componentes gráficos que são exibidos na tela do sistema, esses componentes podem ser ícones, menus, itens de menus, botões, janelas, digitação de textos, etc. Além daquelas palavras existentes na tela, o leitor também pode falar outras palavras, sempre pensando na usabilidade e acessibilidade do usuário. O caso mais comum e de fácil compreensão dessa questão é quando se apaga uma letra ou palavra em um editor de texto, o leitor poderia ler apenas a letra deletada ou nem isso, porém é comum que o leitor adicione a palavra "delete" antes do texto apagado em sua fala, como no caso do nosso leitor.

Além dos requisitos funcionais podemos citar o requisito não-funcional desempenho, que é importantíssimo no desenvolvimento deste leitor de tela. O desempenho do algoritmo do leitor de tela deve ser rápido o bastante para não interromper o fluxo natural de execução de um ambiente gráfico.

A meta de um leitor de tela é conseguir interpretar a maior quantidade possível dos componentes gráficos para que todo o texto em tela resulte em fala. O leitor desenvolvido é capaz de interpretar poucos componentes através dos eventos[1] gerados, porém este número de componentes interpretados já acarreta um bom relacionamento com o usuário.

Um aspecto que fez o número de eventos monitorados cair, foi que alguns desses eventos traziam problemas para a execução normal da aplicação, logo tiveram que ser excluídos, caso contrário prejudicariam a usabilidade do sistema. Esse problema acontece, por exemplo, com eventos que demonstram os textos em tela apenas em algumas vezes que são gerados, e nas outras tantas vezes o conteúdo mostrado não tem sentido com o que está em tela. Outros eventos, no entanto, podem trazer alguns pequenos problemas, mas foram mantidos devido a sua importância. Neste caso, podemos citar o evento de inserção de textos, que é chamado quando algum texto é inserido na tela. O problema é que alguns aplicativos, além dos usuários também geram esse evento, e em algumas vezes a inserção do texto não está visível na tela para o usuário.

[1] Apesar de termos que reproduzir os textos dos componentes gráficos, este texto é obtido através dos eventos que os componentes geram.

No sentido de tornar o leitor um sistema que pode ser acessado por outros aplicativos, a linguagem Python surgiu como uma solução, pois é uma linguagem que pode ser facilmente adaptável a outras linguagens. O fato de executar sobre um interpretador pode ser útil em alguns casos, por exemplo, um programa escrito em C pode chamar o *script* Python através da função *system()*, executando o interpretador Python juntamente com o *script* do leitor. Outra forma de um programa em C executar o script Python é implementar a biblioteca "python.h"[2], que é a maneira com a qual o xLupa executa a leitura de tela, com o código em Python embutido no código C nativo do aplicativo.

4.2 Implementação

Esta seção engloba todos os aspectos da implementação do leitor de tela. A primeira parte da seção trata das linguagens, bibliotecas, aplicativos e interfaces utilizadas durante todo o desenvolvimento do leitor de tela. Em cada tópico abaixo, pode-se encontrar a motivação para o uso de tal recurso, assim como, a sua importância dentro do projeto e uma sucinta descrição sobre a ferramenta. Após esta apresentação das ferramentas utilizadas, o texto se foca na parte de implementação do sistema de leitura de tela, então, todos os aspectos da implementação são descritos desde os primeiros passos até a conclusão do aplicativo.

4.2.1 Python

Python é uma linguagem de programação interpretada e orientada a objeto, que pode ser utilizada no desenvolvimento de *software* livre ou comercial[3]. É utilizada como linguagem de desenvolvimento primário ou como uma linguagem auxiliar por outras aplicações que usam linguagens e ferramentas diferentes. Possui a vantagem de poder se integrar fortemente com C, C++, ou Java, e com objetos COM e .NET [4].

Python é a linguagem de programação utilizada para implementar a leitura da tela sendo usada para executar a comunicação com a interface AT-SPI através da biblioteca pyatspi. Ela recebe os eventos gerados nessa comunicação, para então chamar o eSpeak com uma frase específica para a execução da leitura.

[2]Biblioteca para embutir código Python dentro de código C, Para maiores detalhes consulte [Matthew 2002]
[3]Exemplos de aplicações que usam o Python: Blender, YUM, GIMP, BitTorrent, Dropbox, Morpheus.
[4]www.python.org/about/

A utilização desta linguagem no projeto se deve principalmente ao fato da existência de uma implementação da interface AT-SPI (a biblioteca pyatspi), que permite o uso dos recursos de acessibilidade presentes no ambiente GNOME. Outro ponto importante é a integração com outras linguagens, além disso, o Python também está presente na maioria das distribuições Linux, podendo ser facilmente acessado via terminal pelo comando "python". Porém, uma das principais motivações que levaram a escolha desta linguagem é a possibilidade da criação de um *script*.

Em termos gerais, o Python nos possibilitou o uso da pyatspi, assim nos limitamos a executar os comandos da AT-SPI, não necessitando implementar nenhum módulo individual para executar uma comunicação com qualquer programa. Outra vantagem da utilização do Python é o código produzido ser enxuto e facilmente legível.

4.2.2 AT-SPI

A AT-SPI (*Assistive Technology Service Provider Interface*, ou em português: Interface do Provedor de Serviços de Tecnologia Assistiva) é uma API de acessibilidade desenvolvida pelo GNOME, como um meio facilitador para desenvolvedores proverem recursos de acessibilidades em suas aplicações [Foundation 2008]. Ela também pode ser usada para automatizar testes de interfaces gráficas de usuário. Utilizando a AT-SPI, *softwares* como leitores de tela ou ampliadores permitem que pessoas com deficiência interajam com as aplicações [Accessibility 2011].

Os programas que são compatíveis com a AT-SPI fornecem informações relevantes de sua interface com o usuário para a API. Para garantir a compatibilidade entre o aplicativo e a AT-SPI existem várias possibilidades que o programador pode optar em realizar para deixar o seu aplicativo mais acessível.

O ATK (*Accessibility Toolkit*) é um pacote de desenvolvimento que fornece uma interface de acessibilidade para ser implementada por outras ferramentas[5]. O GAIL é uma dessas ferramentas, que implementa as interfaces de acessibilidade do ATK[6]. Por sua vez, o GTK+ é uma biblioteca para a criação de componentes gráficos (janelas, botões, etc.), que utiliza o GAIL para fornecer recursos de acessibilidade para as interfaces criadas pela mesma[7]. Logo, como

[5]http://developer.gnome.org/atk/
[6]http://developer.gnome.org/accessibility-devel-guide/stable/dev-start-5.html.en
[7]http://live.gnome.org/GAP/AtkGuide/Gtk

o ATK tem uma ponte de comunicação com a AT-SPI, qualquer *software* desenvolvido com o ATK é uma das possibilidades de implementar os recursos da AT-SPI. Então, desenvolver o *software* com a biblioteca GTK+, possibilita-o de utilizar os recursos de acessibilidade presentes na AT-SPI, já que o GTK+ é desenvolvido com base no ATK através do GAIL [8].

Caso o aplicativo não utilize a GTK+, para criar sua interface gráfica, ele pode possibilitar a comunicação com a AT-SPI implementando diretamente ou por outra biblioteca os recursos da ATK. Esse é o caso da suíte Mozilla, que implementa uma biblioteca chamada nsIAccessible[9] para poder contar com a compatibilidade com o ATK (isto pode ser visto na Figura 4.1).

Outra ponte de comunicação semelhante a que existe entre a AT-SPI e o ATK é o *Java Accessibility Framework*, que garante a compatibilidade entre os *softwares* desenvolvidos com Java e a AT-SPI [Schmi 2011].

A Figura 4.1 (adaptada de [Lee 2008]), mostra onde se encontra e atua a AT-SPI num esquema global juntamente com várias outros recursos. Podemos inclusive, ver como o GTK+, o GAIL e o ATK estão distribuídos nesse esquema.

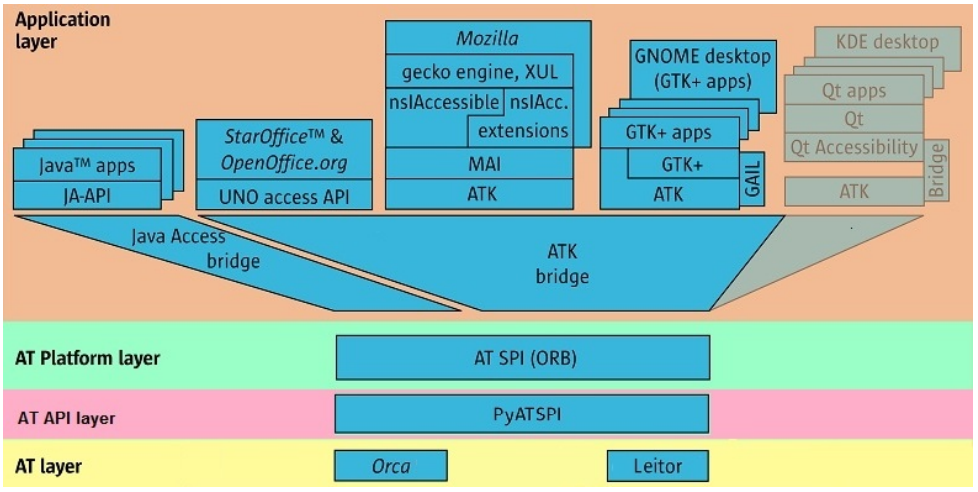

Figura 4.1: Esquema Global do AT-SPI

Note que a AT-SPI é o meio de ligação entre os aplicativos de acessibilidade, na figura representados pelo Orca e pelo Leitor, e as camadas de aplicação, mas ela não atua diretamente sobre os aplicativos do sistema e sim sobre outras camadas de pontes, como a *Java Access Bridge* e a *ATK Bridge*, que garantem a compatibilidade dos aplicativos com a AT-SPI. Essas

[8] Para maiores detalhes sobre esses recursos acesse o site http://live.gnome.org/GAP/AtkGuide/Gtk.
[9] https://developer.mozilla.org/en/nsIAccessible

pontes são responsáveis por buscar as informações dos aplicativos que estão rodando no sistema e repassar tais informações para a AT-SPI. Então a AT-SPI pode, através dessas informações, notificar os aplicativos de acessibilidade[10].

A Figura 4.1 é dividida em quatro camadas. A primeira "*Application layer*", é a camada de aplicativos comuns que executam no sistema operacional, e as suas respectivas bibliotecas em que são desenvolvidos. Essas bibliotecas têm pontes de comunicação com a AT-SPI que são responsáveis por passar as informações da interface gráfica dos aplicativos para a AT-SPI. Essa comunicação que ocorre nesta primeira camada acontece em CORBA[11], já a comunicação que ocorre entre as camadas "*AT API layer*" e "*AT layer*" acontece via chamadas e passagem de parâmetros das funções da biblioteca pyatspi.

O principal recurso da AT-SPI utilizado na implementação do leitor é a capacidade de monitoração das ações do usuário no sistema. Através dessa monitoração temos conhecimento sobre os eventos gerados pelo usuário, assim como as informações dos programas que este está utilizando. A AT-SPI é simplesmente uma especificação de uma interface. Em nosso caso, optou-se pela implementação da biblioteca pyatspi escrita na linguagem Python.

4.2.3 pyatspi

A pyatspi foi criada para ser um invólucro oficial em Python para a AT-SPI. Agora, ela faz parte do módulo AT-SPI do GNOME para qualquer tecnologia assistiva ou para ferramentas de testes automatizados. A principal função da pyatspi, advinda da especificação da AT-SPI, é retirar o esforço do programador em ter que criar módulos específicos para cada *software* que esteja executando no sistema, assim o programador pode criar um módulo genérico para tratar todos os aplicativos [Diggs 2011].

Para o desenvolvimento da pyatspi, o GNOME considerou vários objetivos, alguns destes acabaram contribuindo para este projeto, entre os quais: o fornecimento de um pacote leve para Python, a tentativa de manter a nomenclatura semelhante à AT-SPI, a ocultação da camada

[10] Além da AT-SPI, existem várias outras APIs de acessibilidade disponíveis para os vários sistemas operacionais. Alguns exemplos dessas APIs são o *Microsoft Active Accessibility* (MSAA) [Microsoft 2011], o *Microsoft UI Automation* [Microsoft 2011], o *Java Access Bridge* [Oracle 2011], a *Apple Accessibility API* [Aple 2011] e a *IAccessible2* [Foundation 2009].

[11] CORBA (*Common Object Request Broker Architecture*) é uma especificação para que objetos de sistemas distribuídos possam se comunicar, independentemente das suas diferenças como arquiteturas, linguagens de programação, sistema em que estejam executando, etc.

CORBA, e a criação de um invólucro único que pode ser reutilizável em todas as tecnologias de apoio ou de testes [Diggs 2011].

Atualmente, os principais *softwares* que utilizam a pyatspi são o Orca, o Accerciser, o LSR e o Dogtail[12].

Com a pyatspi, foi implementado um módulo que é responsável pela captura dos eventos gerados pelo usuário durante a utilização do sistema operacional e de seus aplicativos. Este módulo monitora o sistema e chama as funções do leitor quando algum evento de interesse do leitor de tela ocorre. No caso de algum evento ocorrer, existe um tipo de tratamento diferente para cada evento, onde se fará um processamento, fazendo com que o texto adquirido através do evento seja lido para o usuário. Este recurso é o principal entre todos utilizados no projeto, pois permite a monitoração de todo o sistema. Caso não existisse este recurso, o nosso projeto seria muito menos viável, pois a implementação se basearia em outras interfaces semelhantes, porém muito menos abrangentes, ou ainda a implementação se basearia na técnica *off-screen model* para garantir a leitura de tela, o que no mínimo, diminuiria a compatibilidade com os vários aplicativos do sistema, além de aumentar o tempo necessário para o desenvolvimento.

A questão da escolha da biblioteca em Python em vez da biblioteca em C, pode ser explicada devido ao fato da biblioteca em C constar como depreciada[13], enquanto a pyatspi é tida como oficial pelo próprio GNOME [Bessa 2011].

4.2.4 Accerciser

O Accerciser é um explorador de acessibilidade para o ambiente GNOME. É desenvolvido em Python, e utiliza a AT-SPI para inspecionar e controlar *widgets* (botões, janelas, menus, etc.), permitindo a verificação de um aplicativo, a fim de demonstrar se ele está fornecendo informações corretas para as tecnologias assistivas e para os *frameworks* de testes automatizados [Isaacson 2011].

Este recurso foi utilizado para procurar entre todos os eventos gerados pela pyatspi, os que seriam úteis no desenvolvimento da leitura de tela. Com o Accerciser, podemos explorar as funcionalidades e as estruturas de dados da pyatspi (já que este utiliza-a), e também para testar

[12]Os *softwares* Accerciser, LSR e Dogtail podem ser encontrados nas páginas http://live.gnome.org/Accerciser, http://live.gnome.org/LSR e http://people.redhat.com/zcerza/dogtail/ respectivamente.

[13]Quando uma biblioteca deixa de ser atualizada ou não possui mais suporte por parte de seus desenvolvedores ela é definida como depreciada.

alguns códigos que futuramente seriam parte do sistema de leitura de tela.

O Accerciser foi o recurso mais utilizado durante a etapa inicial do desenvolvimento. E a funcionalidade mais utilizada do Accerciser foi o "Monitor de Eventos". Com ele, selecionamos dentre o total, alguns eventos e continuamos a executar as ações no computador como um usuário normal faria. Com isso, os eventos escolhidos eram monitorados e a tela do Accerciser demonstrava os eventos que eram registrados pela AT-SPI, assim como a sua estrutura de dados.

Com as informações sobre cada evento que obtemos através do "Monitor de Eventos" pode-se concluir rapidamente que a maioria dos eventos seriam descartados, como os eventos advindos do *mouse* que não contém informações relevantes para a leitura (no entanto, podem ser interessantes no desenvolvimento de outras tecnologias assistivas como um ampliador de tela). Verificando cada um dos eventos disponíveis, pode-se ter a ideia dos eventos que seriam úteis para um leitor de tela, e com estes eventos escolhidos, foi iniciado o processo de desenvolvimento de códigos para teste, a fim de verificar se os eventos previamente selecionados realmente seriam úteis, ou seja, se a partir deles era possível chegar aos textos que são exibidos em tela.

As funcionalidades presentes no Accerciser implementam vários recursos da pyatspi, como a monitoração de eventos e o mapeamento das aplicações que o sistema está executando. Com a sua utilização tivemos a perspectiva do que conseguiríamos fazer a partir da implementação dos mesmos recursos da pyatspi. Sem o uso do Accerciser, as implementações dos vários recursos da pyatspi teriam que ser codificados. Por exemplo, o "Monitor de Eventos" monitora e imprime os eventos (com a sua estrutura de dados) que selecionamos apenas em alguns cliques. Para construir uma funcionalidade semelhante somente para testar cada evento disponível na pyatspi, teríamos que gerar muito código, além de perder tempo executando, testando, corrigindo falhas, etc.

Em vários outros pontos o Accerciser auxilia os desenvolvedores, por exemplo, na questão de inserção de texto. Acontece que a inserção de texto nos aplicativos acarreta dois eventos que seriam monitorados pelo nosso algoritmo, um pela própria inserção de texto e outro pelo deslocamento do cursor. Uma ação provocar dois eventos era um problema para ser tratado na implementação, pois cada ação do usuário deve se transformar em um evento para ser lido, mas como já se sabia que este fato iria acontecer, uma solução foi sendo pensada desde o momento da descoberta desse contratempo até o momento da implementação. Deste modo, a solução já

estava prevista no momento da implementação evitando dificuldades durante essa etapa.

O resultado em nosso projeto é que o Accerciser contribuiu para otimizar o processo de desenvolvimento do código do leitor. No momento de desenvolver o código final do leitor, já se possuía grandes informações sobre a implementação, além de pedaços de códigos que certamente fariam parte do algoritmo final. Com essa visão geral do código final e alguns trechos já prontos, o resultado final foi obtido com mais facilidade.

4.2.5 eSpeak

O eSpeak é um *software* sintetizador de voz com as vantagens de possuir o código aberto, suportar vários idiomas, e ser desenvolvido para vários sistemas operacionais. Várias outras vantagens para o uso do eSpeak, como a sua adoção pelo Ubuntu como *software* embarcado no sistema operacional, por exemplo, que foram as principais característica para a escolha da adoção deste sintetizador no leitor de tela, além é da dicção de voz do mesmo que é uma das melhores disponíveis atualmente no mercado. O tamanho incrivelmente pequeno do *software*, apenas 1.4 *megabytes*, é outro ponto a se destacar [eSpeak 2011].

Para executar um teste rápido com o eSpeak, basta no Ubuntu, abrir o terminal e digitar algo como “espeak ’olá mundo!”’, e será lido “olá mundo!”. Mas a leitura ocorrerá com um sotaque inglês, pois a linguagem é a padrão do *software*. Para executar a leitura com o idioma português basta reescrever o comando da seguinte forma: “espeak ’olá mundo!’ -v brazil”, e então o idioma português do Brasil é utilizado para executar a leitura. Existem vários outros parâmetros além do -v, como -s (velocidade), -a (amplitude), etc. Para mais informações sobre estes parâmetros use o comando “espeak –help”.

Com respeito à utilidade e qualidade do *software* não há o que se discutir, o *software* atendeu todas as nossas expectativas. A dicção de voz no idioma brasileiro não é perfeita, mas é muito satisfatória, tanto que é utilizado por muitos outros *softwares* da área. Apesar das tentativas de mudança, a execução do eSpeak ocorre de forma direta, sendo executado por um comando de terminal onde é passado como parâmetro o texto que deve ser lido.

4.2.6 Modelagem do Leitor de Tela

A Figura 4.2 a seguir, ilustra a comunicação do leitor de tela com outras entidades envolvidas no processamento de captura de textos e geração da fala.

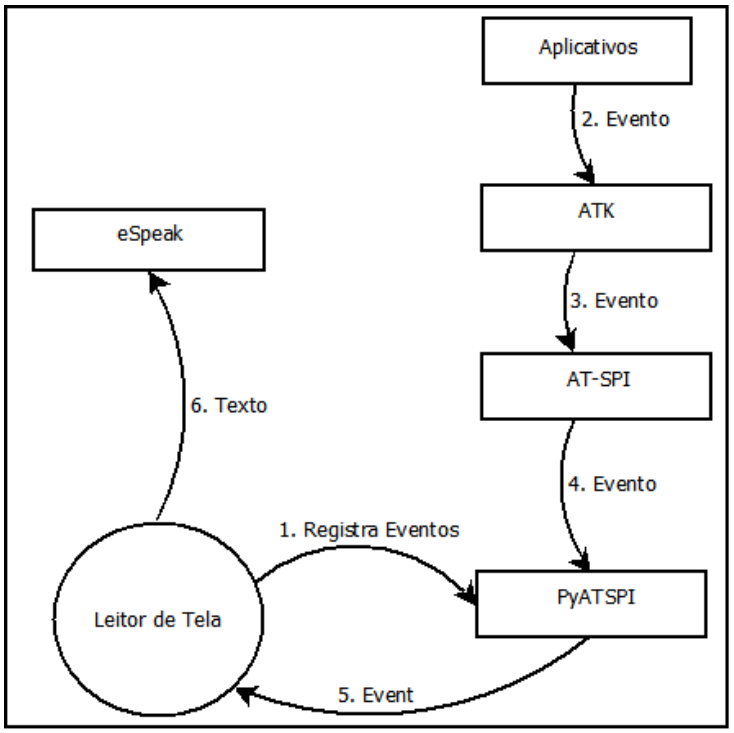

Figura 4.2: Diagrama de Contexto do Leitor de Tela

Como a Figura 4.2 ilustra, o papel do leitor de tela é registrar os eventos (seta 1) que deseja monitorar através da biblioteca pyatspi, e então receber esses eventos (seta 5) para tratá-los a fim de isolar o texto a ser falado. Depois que o leitor registra os eventos, através do método *RegisterEventListener* da pyatspi, a própria pyatspi irá chamar as funções do leitor passando o evento ocorrido como parâmetro (*Event*). As funções do leitor são responsáveis por buscar os textos da tela do computador através das informações contidas nesses eventos. Com o texto "em mãos" o leitor chama o sintetizador de voz eSpeak (seta 6), que tem a responsabilidade de ler o texto processado.

Outro aspecto demonstrado na Figura 4.2 é o caminho que um evento percorre até chegar ao leitor de tela. Primeiramente, um evento é gerado em algum aplicativo (seta 2), com o usuário clicando em um botão, por exemplo. Se o aplicativo implementa a ATK ou algum outro recurso de acessibilidade compatível com a AT-SPI, este evento será mandado para a AT-SPI (seta 3). A

biblioteca pyatspi implementa a interface AT-SPI, ou seja, os eventos mandados para a interface chegarão na pyatspi (seta 4), então esta redirecionará os eventos para aqueles aplicativos que se registraram para receber estes eventos (seta 5), que como já foi dito, é o caso do leitor de tela.

4.2.7 Identificação dos eventos

Entre as primeiras tarefas realizadas, está a análise dos eventos que a biblioteca pyatspi[14] é capaz de monitorar e os dados e informações que possuem[15]. Depois de analisar todos os eventos e dimensionar quais aparentavam ser interessantes para a leitura, passou-se a monitorar o sistema, com ajuda do aplicativo Accerciser. Através de uma ação executada no sistema podemos visualizar os eventos gerados e as suas respectivas informações.

Nesta primeira etapa foram levantados alguns eventos que aparentavam ser úteis para serem utilizados pelo leitor de tela. Estes eventos basicamente são relacionados as ações que exibem textos na tela com a movimentação do usuário pelo sistema, como os textos encontrados em menus, ícones, janelas e editores de texto. Abaixo, citamos os eventos selecionados para a leitura e uma sucinta descrição sobre cada um:

Focus: O evento focus é responsável por monitorar todos os componentes gráficos da tela que podem receber o foco. Em uma utilização normal do sistema, este evento é o que mais será gerado, pois todo menu, botão, ícone, aba, etc. que receber foco gerará tal evento, e com o processamento realizado sobre este evento é possível obter o texto contido nestes componentes gráficos.

Object:text-changed:insert: Toda digitação ou inserção de texto nos aplicativos gerará este evento. Nota-se que este evento é importante, pois é responsável pela geração de eventos quando o usuário está editando um texto. Além de monitorar o teclado, este evento também registra as inserções de texto realizadas por aplicativos (ou pelo usuário no caso de um comando colar). Para se obter o texto do evento, não é necessário tanto processamento, já que basta acessar um atributo do objeto Event [16].

Object:text-changed:delete: Semelhante ao evento anterior, também está ligado à edição

[14] A documentação da biblioteca pode ser encontrada no site: http://people.gnome.org/ parente/pyatspi/doc/.

[15] Alguns exemplos desses eventos são: mouse:abs, terminal:application-changed, window:close, object:model-changed, document:reload, e podem ser encontrados neste endereço: http://accessibility.linuxfoundation.org/a11yspecs/atspi/adoc/atspi-events.html

[16] http://people.gnome.org/ parente/pyatspi/doc/pyatspi.event.Event-class.html.

de textos no sistema, mas só é gerado ao se realizar uma exclusão de texto em vez de uma inserção, sendo também primordial na tarefa da leitura de tela no momento de uma edição de texto. O processo de obtenção do texto referente ao evento é idêntico ao do evento anterior.

Object:text-caret-moved: Mais um evento relacionado à edição de textos. O evento ocorre quando o cursor do texto é movimentado, logo se difere dos outros por que não ocorre somente em uma modificação do texto na tela, mas acontece quando o usuário movimenta o cursor de digitação sobre o texto (usando as setas, por exemplo). O processamento deste evento não é tão simples quanto os outros eventos relacionados a textos. Neste caso, a informação que obtemos com o evento é a posição do cursor no texto, quando este sofre uma movimentação. Devido a isto, precisamos obter o texto referente ao evento acionando a função *getText* do objeto *Event* e a partir da posição do cursor no texto encontrar o trecho correto manipulando uma cadeia de caracteres.

Window:activate e Window:create: Estes dois eventos estão ligados a manipulação de janelas, sendo semelhantes entre si, porém se diferenciam no momento em que são gerados. O Window:activate é gerado no momento em que uma janela é ativada, ou seja, no momento em que ela é colocada em destaque estando escondida ou minimizada anteriormente. Já o evento Window:create acontece apenas uma vez na "vida" de uma janela, ele ocorre no momento em que uma janela é criada na tela. Para poder chegar ao texto a ser lido pelo leitor deve-se acessar um atributo do objeto Event, que contém o nome da janela.

O sistema é monitorado pelo leitor de tela através da pyatspi com base nos eventos citados acima. Através desses eventos conseguimos monitorar as ações de abertura de janelas, ativação de janelas, seleções, inserções, deleções e edições de texto, seleções de menus, itens de menus, botões e ícones. Desta forma, conseguimos um relacionamento satisfatório com o usuário, pois muitas ações que ele executa no sistema serão lidas.

4.2.8 Implementação das funções relativas aos eventos selecionados

Depois dos eventos devidamente escolhidos, implementamos os recursos da biblioteca pyatspi, a fim de construir um código que resulta-se em algo semelhante com o que o *software* Accerciser exibe. Esta tarefa era necessária para chegarmos aos dados textuais relevantes dos componentes gráficos da tela, que já eram exibidos no Accerciser, embora outros textos

que precisaríamos ler ainda não eram exibidos no mesmo e aparentavam precisar de um processamento mais complexo. Para implementar a função que nos retorna os textos do evento foi utilizado o método *registerEventListener* da classe *Registry*. O código mais simples possível que mostra tal execução é mostrado a seguir.

```
import pyatspi
reg = pyatspi.Registry
def foco(event):
    print event

reg.registerEventListener(foco, 'focus')
reg.start()
```

O código mostrado acima inicia com a importação da biblioteca pyatspi e logo abaixo é criado um objeto *Registry*. O método *start* da última linha é responsável por criar um *loop* que se encarrega de monitorar os eventos, somente depois de executá-lo é que os eventos serão registrados e processados pela função *foco*. O método *registerEventListener* registra para monitoração o evento *focus*, e quando ocorrer o evento, a função *foco* será chamada pela pyatspi que passará como parâmetro o objeto *Event* que contem as informações a respeito do evento de foco que ocorreu no sistema. Neste exemplo, dentro da função *foco* apenas imprimimos o conteúdo de *Event*, que terá como saída um texto semelhante ao seguinte:

```
focus:(0, 0, None)
source: [menu | Sistema]
application: [application | gnome-panel]
```

Um detalhe importante aqui, é que o texto a ser lido neste tipo de evento já aparece nesta impressão. Neste evento o que interessa ao usuário é o conteúdo de *source*, que corresponde ao item que ganhou o foco naquele momento. Dentro de *source* podemos encontrar dois itens, o primeiro item (*menu*) corresponde ao tipo de componente gráfico que o usuário selecionou, outras opções que apareceriam no lugar de *menu* seriam, por exemplo, item de menu, ícone, botão, etc. O segundo item (*Sistema*) corresponde ao nome do componente que o usuário selecionou, por padrão o nome que aparece neste local também é o nome que está na tela, ou seja, é exatamente este texto que deve ser lido ao usuário. Quanto aos outros eventos, o processo para uma impressão do objeto *Event* é idêntico, bastando substituir o evento 'focus' por outro, porém o processo para encontrar o texto a ser lido é mais complexo.

Nestes pequenos trechos de código mostrados acima, podemos ter a ideia de como todo o sistema foi implementado e de seu funcionamento: o método *registerEventListener* primeiramente é chamado várias vezes, passando se uma função e um evento diferente como parâmetro para cada uma dessas chamadas. Então, para cada chamada do método *registerEventListener* é implementada uma função que tem o objetivo de processar o evento, a fim de encontrar o texto que deve ser falado. Além dessas funções, o *script* de leitura ainda conta com outras funções implementadas para controlar o fluxo de execução do leitor.

Depois que temos o texto a ser lido já isolado, o passo seguinte é chamar o sintetizador de voz para executar a leitura. A tarefa é simples e não necessita de outro tipo de processo mais complexo. Durante a execução do algoritmo o processamento deve ser simplificado, até por que, este não pode interferir na utilização do sistema, ou seja, o algoritmo deve ser suficientemente rápido para não interromper a execução e o uso normal do sistema. Não existe nenhum tipo de estrutura na memória como uma pilha ou fila para gravar estes eventos, acontece que, a utilização em um ambiente gráfico é muito diferente entre os usuários, assim não existe razão para armazenar os eventos, e como o usuário só necessita ouvir o texto relacionado ao evento no momento em que ele acontecer, no caso do evento ser repetido pelo usuário, a ação não é buscar na memória, mas sim processar o novo evento que a AT-SPI gerou.

Com a construção do código descrita acima chegou-se ao resultado final que é um *script* na linguagem de programação Python. Este pode ser executado via terminal do Linux, ou pode ser incorporado nos códigos de outros aplicativos (um exemplo de incorporação do leitor a outro sistema pode ser visto na Seção 4.2.9). A execução do leitor de tela através deste *script* ocorre sem travamentos ou problemas de qualquer natureza e em relação as funcionalidades e qualidade do leitor, a comparação realizada com o leitor de tela do Orca na Seção 4.4.1 deste mesmo capítulo pode mostrar como este leitor chegou a um nível de qualidade muito satisfatório.

4.2.9 Incorporação do Leitor de Tela a outro Aplicativo

A possibilidade de incorporação do código em Python do leitor de tela a outros aplicativos é a principal vantagem e inovação deste trabalho. A partir deste trabalho outros desenvolvedores dos mais diversos tipos de aplicações podem contar com um módulo externo de leitura de tela. A ideia é que os aplicativos que desejam utilizar o módulo de leitura o façam de duas maneiras

diferentes. A primeira possibilidade é incorporar o código do leitor dentro da aplicação, esta possibilidade é demonstrada nesta seção, com a incorporação do código do leitor no código do ampliador xLupa. A segunda possibilidade é a execução paralela, onde a execução é feita pelo próprio aplicativo ou pelo usuário no interpretador Python, esta possibilidade incapacita o controle da leitura por parte do aplicativo.

A incorporação, como mostrada nesta seção é mais adequada, pois assim o controle da leitura pode ser realizado internamente ao aplicativo do desenvolvedor. Como ocorre no xLupa, o aplicativo que incorpora o código do leitor pode gerenciar questões como a velocidade de leitura e quando esta leitura é executada.

Para fazer tarefa semelhante com o que o leitor desenvolvido propõe, o desenvolvedor de outros aplicativos tinha algumas outras possibilidades como, por exemplo, obter o código fonte de um leitor de tela com código aberto, como o Orca, e estudá-lo para então adaptar o código do leitor ao código de seu aplicativo. Esta tarefa não é uma das mais fáceis de ser realizada, pois o processo contem alguns obstáculos, como o próprio estudo de código desconhecido e adaptação deste código se isso for possível. O código do leitor desenvolvido é um *script* em arquivo único de fácil compreensão, diferente de outros sistemas de leitura de telas, onde o código é dividido entre vários arquivos de incontáveis linhas de código.

Logo abaixo podemos ver a seção onde exemplificamos uma das maneiras possíveis para incorporar o código em Python do leitor de tela a outro *software*. Nesta seção o leitor é incorporado a um código escrito na linguagem de programação C.

Incorporação do Leitor de Tela ao xLupa

O xLupa é um ampliador de tela desenvolvido na linguagem de programação C, e em sua versão 3.0 já conta com um leitor de tela, que foi a base para o desenvolvimento deste projeto. No entanto, como o leitor desenvolvido é muito diferente daquele do xLupa 3.0 uma nova incorporação do código em Python do leitor precisa ser feita no código C do ampliador.

Para incorporar o código do leitor ao código do xLupa temos a dificuldade de ter que trabalhar com duas linguagens de programação. Porém, a linguagem Python é uma das linguagens mais viáveis para este tipo de incorporação.

A incorporação do código Python dentro do código em C é possível devido a biblioteca

python.h[17]. Esta biblioteca contem funções e procedimentos para realizar todos os processamentos exigidos na incorporação[18].

A solução escolhida para incorporar o leitor de tela ao ampliador foi criar dentro do código C um interpretador Python[19], deste modo, o *script* do leitor de tela pode funcionar como quando é executado em um terminal, que é a execução normal de uma linguagem interpretada, caso do Python.

Para utilizar o interpretador Python internamente ao código C necessitamos primeiramente incluir a biblioteca no código através da linha:

```
<include> "python.h"
```

Com a inclusão da biblioteca podemos usar todos os seus recursos. Um desses recursos permite a criação de um interpretador. Para iniciar o interpretador precisamos primeiramente executar a função:

```
Py_Initialize();
```

Quando o interpretador deixar de ser usado deve-se encerrá-lo para que deixe de ocupar memória e recursos da máquina através da função:

```
Py_Finalize();
```

Depois de executar a função para iniciar o interpretador podemos executar comandos no interpretador pela função:

```
PyRun_SimpleString();
```

Esta função é executada passando o comando em Python como parâmetro para a função, para exemplificar, o código abaixo mostra como seria a execução de uma impressão na tela através do comando em Python:

```
PyRun_SimpleString("print i");
```

A palavra "*print*" é o comando em Python para impressão na tela e o "*i*" é uma variável qualquer. O funcionamento de um interpretador segue uma execução linha a linha, e este interpretador interno ao C funciona da mesma maneira, ou seja, as funções e os comandos Python

[17] http://docs.python.org/extending/extending.html

[18] Para fazer a incorporação as principais fontes de pesquisa foram as documentações encontradas no próprio site da biblioteca e em [Matthew 2002].

[19] http://docs.python.org/extending/embedding.html

devem estar dispostos no código C da mesma maneira que estão dispostos no código Python original.

Com os passos descritos acima realizados no código do xLupa, o leitor de tela pode ser utilizado normalmente dentro do ampliador. A execução do leitor de tela dentro do xLupa funciona da mesma maneira que a execução do leitor via *script* pelo terminal.

Outra possibilidade para execução do *script* do leitor de tela é a execução direta, sem a necessidade de incorporação do código, bastando apenas executar o *script* através da função:

```
system("python scriptLeitor.py");
```

Na função "*system*", o comando "*python*" inicia o interpretador Python e "*scriptLeitor.py*" é o arquivo de *script* do leitor de tela.

4.3 Problemas Ocorridos no Desenvolvimento

No desenvolvimento de um leitor de telas há muitos problemas conhecidos que devem ser superados, como a escolha do sintetizador de voz, a estratégia de implementação, a escolha dos eventos e ações a serem lidas, as interfaces de usuário, o suporte a documentos de formatos diferentes e o suporte a diversos aplicativos [Raj 1998]. Além desses problemas ainda existem aqueles problemas específicos que aconteceram em tempo de codificação, estes problemas são descritos logo abaixo.

Um dos primeiros eventos implementados foi o evento de inserção de textos, e também foi o primeiro evento a gerar um problema. Alguns aplicativos geram inserção de textos, e isso nem sempre é visível ou importante para ser lido ao usuário, por exemplo, o site do Gmail[20] possui com um contador de espaço disponível, mas esse contador muda a todo o momento e acaba gerando um evento de inserção de texto a cada mudança, mesmo se a página não estiver selecionada. Infelizmente, não podemos desprezar o evento de inserção de texto pela sua importância na execução do leitor. A solução neste caso foi mapear o evento que causa o problema e na hora de processar os eventos verificamos que se trata do evento do Gmail, então cancelamos a sua execução sem produzir a fala. Este mapeamento é a simples verificação do nome do aplicativo que está gerando o evento, bem como o nome do próprio evento.

[20] www.gmail.com

Outro problema enfrentado foi a questão da fala ter que ser cortada quando outro evento ocorre, pois é muito desagradável para o usuário ter que ouvir todo o texto de um item de menu para só então se movimentar para outro item, no caso do usuário que navega em um menu, por exemplo. Isso ocorre por que o usuário ganhando experiência acaba conhecendo o sistema e sabe o texto daquele item sem precisar ouvir todo o texto do componente. Essa funcionalidade que o leitor deve atender depende do sintetizador de voz, pois depois de isolar o texto para a leitura ele é mandado ao sintetizador e este sintetiza o texto transformando-o em voz. Neste processo não temos uma maneira muito elegante para fazer com que o sintetizador interrompa a fala no meio da frase ou palavra. A solução encontrada foi interromper o processo responsável pela síntese de voz, assim a fala é interrompida no momento em que está sendo lida pelo sintetizador de voz.

Quando é digitada uma letra em um editor de texto dois eventos acontecem, a letra aparece na tela e o cursor do texto é movimentado, ou seja, acontece um evento de Object:text-changed:insert e um de Object:text-caret-moved, e como monitoramos estes dois eventos as duas funções que os tratam são executadas. O problema nesse caso aparece quando inserimos um texto (comando colar, por exemplo) e os dois eventos são chamados, sendo que o último a ser chamado é o evento de movimentação de cursor, ou seja, este evento cortaria o anterior e a leitura a ser executada seria referente à letra ao lado do cursor, que é o que o evento de movimentação de cursor deve ler. Logo, todo o texto colado que deveria ser a leitura correta seria descartado pelo leitor.

A solução para contornar os dois eventos que são gerados é sempre executar a leitura da inserção de texto. E a leitura de uma movimentação de cursor deve ser executada apenas quando essa movimentação não foi causada por uma inserção, isso só acontece quando o usuário movimenta o cursor usando o mouse ou as setas do teclado sobre o texto. Para garantir esta solução criou-se uma variável que faz esse controle, tomando um certo valor a cada inserção e voltando ao valor antigo quando ocorrer uma movimentação, claro que, quando o valor da variável for modificada na função de inserção (que é chamada antes da movimentação) a função de movimentação é finalizada sem realizar o processamento para executar leitura. Quando a variável não sofrer alteração na função de inserção o evento de movimentação de cursor poderá ser lido.

Para tentar solucionar algumas questões relacionadas a síntese de voz, como a funciona-

lidade da fala cortada, houve a tentativa de implementação da biblioteca pyttsx[21], do mesmo desenvolvedor da pyatspi e criada com o intuito de ajudar a comunicação entre sintetizadores e leitores de tela. A implementação do código com a biblioteca e a utilização de seus recursos é prática, porém, a biblioteca não executa a leitura da maneira desejada. A principal diferença na forma da leitura quando é utilizado a biblioteca pyttsx é que não se consegue interromper a fala que está sendo lida naquele momento. Conseguimos, no entanto eliminar o efeito de fila e falar apenas o último evento gerado[22] (o que também ocorre no leitor sem o uso da pyttsx), mas mesmo com esse beneficio o uso da biblioteca não faz sentido, pois a exigência principal é justamente cortar o texto que está sendo lido no momento.

Outras bibliotecas foram pesquisadas para uso no leitor, porém não puderam ser usadas, como por exemplo, a PySpeech[23] que não foi utilizado pois só pode ser executada em Windows.

Um dos problemas encontrados que não pode ser considerado um erro é que alguns botões clicados não são lidos. Isso acontece em alguns casos, como por exemplo, quando clica-se em fechar e o programa exibe um janela com as opções “salvar e sair”, “sair sem salvar” e “cancelar”. Então quando o usuário clica em qualquer botão para sair a leitura deste botão teria que ocorrer, porém muitas vezes ao clicar em sair o programa é fechado e outra janela é ativada rapidamente, ou seja, a leitura do botão não ocorrerá, pois a sua leitura será cortada para dar lugar a leitura da janela que foi ativada. Como dito, não pode ser considerado um erro, por que esse tipo de execução é o esperado em um sistema que trabalha com janelas. Infelizmente, neste caso não existe maneira para verificar esse tipo de execução a fim de ler os dois eventos (clique no botão e ativação da janela).

Os problemas descritos até aqui são relativos a parte de desenvolvimento e codificação do leitor de tela, porém estes não são os únicos problemas enfrentados. Os problemas abaixo descrevem os obstáculos enfrentados durante a etapa de testes do leitor, sendo que alguns desses problemas foram verificados apenas em algumas máquinas, ou seja, os problemas abaixo são de origem do sistema operacional ou da máquina do usuário.

Em alguns computadores verificou-se que existe um controle do volume dos sons do com-

[21] http://pypi.python.org/pypi/pyttsx

[22] Tal comportamento é construído com o uso da função *stop()*, sempre antes de executar as funções *say()* e *runAndWait()*, assim todos os textos que foram colocados na fila para serem lidos são retirados da fila, e o texto atual pode ser lido.

[23] http://code.google.com/p/pyspeech/

putador mais complexo que em outros computadores (diferenciando o volume dos alto-falantes e da saída do fone de ouvido, por exemplo), neste caso o ideal é aumentar o volume de todas as saídas de sons possíveis, executando o aplicativo para controle dos sons do computador.

Outro problema relacionado ao som do computador é a concorrência de áudio. Por exemplo, se um vídeo estiver sendo reproduzido e a leitura de tela for ativada, um erro de acesso ao dispositivo de áudio pode ser disparado (se a execução for em um terminal o erro será exibido) e a leitura não acontece. A solução é verificar que nenhum programa esteja utilizando o áudio do computador.

No Linux Ubuntu os recursos de acessibilidade podem estar desativados. Nesse caso, o sistema simplesmente não funcionará, pois depende da ativação desses recursos para poder se comunicar com a AT-SPI que só começa a trabalhar com essa ativação. Para ativar esse recurso basta no menu Sistema, acessar Preferências e clicar em Tecnologias Assistivas, marcando a opção Habilitar tecnologias assistivas, para finalizar clique em fechar e encerre a sessão, então entre no sistema novamente para que a modificação tenha efeito.

4.4 Análise dos Resultados

Descrevemos nesta seção um comparativo entre o leitor desenvolvido e o leitor de tela do Orca, a fim de analisar as funcionalidades do leitor verificando a sua qualidade e desempenho frente a outro aplicativo que é atuante no mercado. Essa comparação está focada nos componentes gráficos que os leitores são capazes de ler e para mensurar este dado foi construída uma tabela que demonstra quais são os componentes que cada um dos leitores é capaz de ler.

Este capítulo ainda conta com a discussão das possibilidades futuras no desenvolvimento deste leitor e com uma sucinta conclusão discutindo aspectos sobre o leitor e seu desenvolvimento.

4.4.1 Comparação com o Leitor de Tela do Orca

Como já foi dito, acreditamos que o leitor desenvolvido não seja concorrente direto de nenhum outro leitor de tela, porém neste tópico tentaremos mostrar as semelhanças e diferenças entre esses dois leitores. A comparação com o Orca foi realizada devido aos recursos e tecnologias que são utilizados por ambos os leitores, assim podemos verificar qual utiliza melhor estas

tecnologias.

As primeiras diferenças visíveis entre os leitores de tela são as interfaces gráficas e os recursos de cada um. Enquanto o Orca conta com várias funcionalidades e com uma interface gráfica para configuração desses recursos o leitor aqui desenvolvido não conta com uma interface gráfica e nem com outras funcionalidades se não a própria leitura de tela. Isto se deve ao fato do leitor ser um *script* para ser executado via terminal ou por outro aplicativo, mas se o desenvolvedor se interessar em ter uma interface gráfica para configuração das opções de leitura este o pode fazer, algo semelhante com a interface gráfica do xLupa para configurar o leitor de tela. Nesta seção será levado em conta apenas os recursos relacionados a leitura de tela do Orca.

O desempenho dos leitores de tela é uma das questões mais importantes em um leitor, pois o usuário não pode lidar com os inconvenientes de atrasos ou travamentos na execução. Nesse quesito os dois leitores trabalham eficientemente, durante os testes não houve nenhum registro de travamento ou atraso durante a execução dos mesmos.

Os dois leitores de tela no momento da execução iniciam a leitura instantaneamente, e com o usuário navegando pelo sistema as leituras dos dois leitores difere um pouco. A Tabela 4.1 exemplifica o funcionamento dos dois leitores listando os principais componentes gráficos de um sistema com interface gráfica e identificando quais desses componentes são lidos ou não pelos dois leitores. A tabela está dividida em três partes a primeira representa os componentes gráficos do sistema operacional e dos aplicativos que nele executam (a Figura 4.3 identifica estes componentes em uma tela padrão do Ubuntu), a segunda e a terceira parte demonstram o funcionamento dos leitores em duas aplicações específicas, o Gedit (um editor de textos padrão do Ubuntu) e o GNOME Terminal. Nestes dois últimos aplicativos as leituras estão vinculadas a eventos advindos da manipulação de textos na tela do computador.

Na Tabela 4.1 os itens assinalados como “sim” executam a leitura da forma esperada, lendo o texto que aparece no componente e os itens assinalados como “não” executam de forma a ler um texto equivocado para o componente ou nem chegam a executar a leitura. A linha do componente Teclas está relacionada as teclas que não registram uma edição de texto como as teclas Alt, Ctrl, Esc, Home, etc. A linha Notificações está ligada as mensagens exibidas pelo sistema operacional Ubuntu como, por exemplo, a notificação que é exibida quando a conexão de rede é ativada. A combinação de teclas Alt+Tab provoca a abertura de uma janela com ícones

Tabela 4.1: Comparação entre os Leitores de Tela

Eventos de navegação em janelas		
Componente gráfico	**Orca**	**Leitor implementado**
Janela	sim	sim
Menu	1	sim
Item de menu	2	sim
Ícone	sim	sim
Alt+Tab	sim	sim
Teclas	sim	não
Notificações	sim	sim
Botões	sim	sim
Eventos de edição de texto		
Editor de texto Gedit		
Evento	**Orca**	**Leitor implementado**
Inserção de texto	3	3
Remoção de texto	4	7
Texto colado	sim	sim
Movimentação do cursor	5	6
Seleção de texto	sim	sim
GNOME Terminal		
Evento	**Orca**	**Leitor implementado**
Inserção de texto	sim	sim
Remoção de texto	4	7
Texto colado	não	sim
Movimentação do cursor	sim	sim
Seleção de texto	sim	sim

das janelas que estão abertas naquele momento no sistema, a cada pressionamento da tecla Tab um novo ícone desta janela é selecionado. A linha Alt+Tab da tabela representa a fala gerada por essa navegação entre os ícones através da tecla Tab.

Os componentes gráficos do sistema que são listados na Tabela 4.1 podem ser identificados na Figura 4.3. Além de identificar alguns exemplos de componentes a imagem também identifica os textos que devem ser lidos ao usuário quando o evento advindo desses componentes for processado, esses textos estão sublinhados. Para exemplificar, vamos identificar um componente e seu texto, o Alt+Tab, como dito este comando abre uma janela contendo ícones das janelas abertas no sistema, na imagem a janela de Alt+Tab contem cinco ícones, e o nome da janela selecionada é "[Idioma & Texto]", note que na figura este texto está sublinhado, indicando que este é exatamente o texto que deve ser lido ao usuário no momento em que acontecer

o evento de Alt+Tab.

Para a ideia ficar mais clara, vamos exemplificar o funcionamento da fala de outro componente. Podemos notar na Figura 4.3 que o cursor do *mouse* está posicionado sobre um componente item de menu, notamos também que este item está selecionado (apresenta fundo mais escuro que seus vizinhos). Quando um item de menu é selecionado, como este está, é gerado um evento de foco, logo, como este é um dos eventos que o leitor é capaz de ler o texto relacionado a este componente gráfico deve ser lido. Na Figura 4.3 identificamos com um sublinhado os textos que devem ser lidos, neste caso, o texto relacionado ao item de menu que será lido é "Escritório".

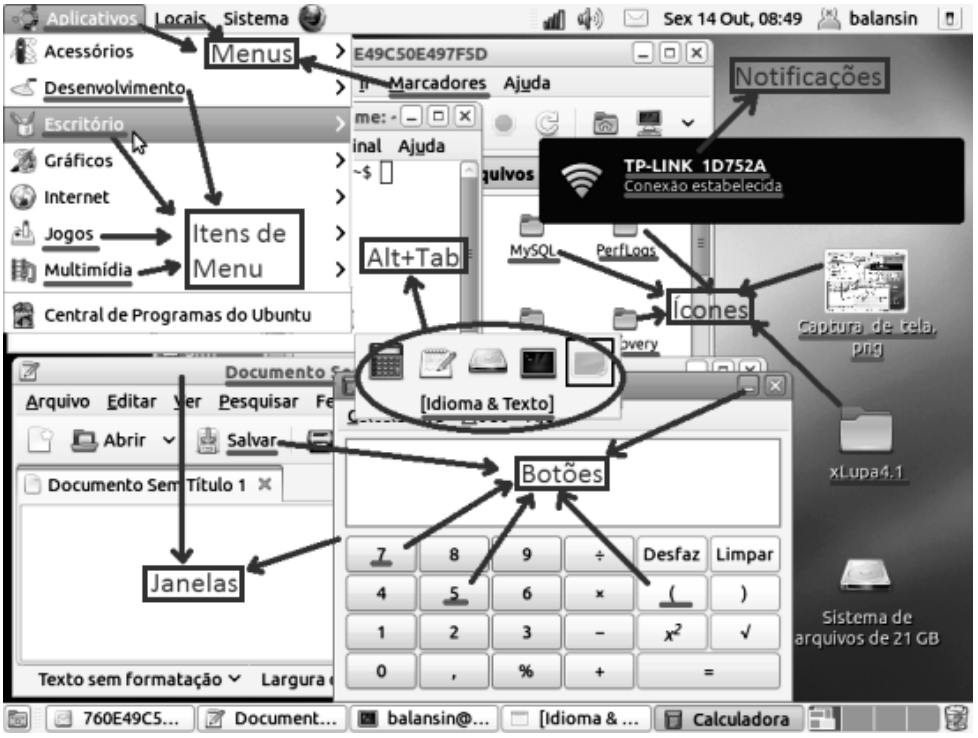

Figura 4.3: Componentes gráficos

Abaixo, segue uma pequena descrição sobre cada um dos itens numerados na tabela para demonstrar o fato ocorrido quando tentou-se executar a ação que deveria resultar na leitura do texto relacionado ao componente.

1. Na maioria das vezes a leitura é realizada corretamente, porém em alguns casos o texto lido não é compatível com o texto da legenda do menu.

2. A leitura acontece corretamente, com o leitor lendo o texto associado ao item, mas quando

a leitura de um item é iniciada e o usuário desloca o cursor para o próximo item de menu a leitura em curso não é interrompida provocando um retardo na leitura do item que o usuário está posicionado. Isto é, todos os itens pelos quais o cursor passou serão lidos até que o texto do item desejado seja lido. Isto certamente causa um incômodo ao usuário, tanto que no leitor desenvolvido este fato não acontece, pois a cada novo evento, que acontece quando o usuário desloca o cursor para o próximo item, a fala do item anterior é interrompida para dar vez a nova fala do item que foi selecionado por último.

3. Algumas vezes a letra a ser lida não é falada. Este erro possivelmente acontece na comunicação ou na execução do eSpeak, pois ocorre nos dois leitores de tela e na execução do eSpeak via terminal, ou seja, sem o uso dos leitores de tela. Outra possibilidade é que o som não seja executado por causa da baixa qualidade da placa de som.

4. Quando a tecla *backspace* é pressionada, o leitor lê *backspace* mais o texto deletado. Quando a tecla *delete* é pressionada o texto lido é "delete" e o caractere que estava à direita do caractere que foi apagado.

5. Com o deslocamento do cursor do texto para outra linha, esta linha é lida e com a movimentação do cursor horizontalmente ao longo do texto a letra à direita ao cursor é lida.

6. Com a movimentação do cursor de texto o caractere à direita do cursor é lido. Se o cursor for posicionado no início de uma linha, a linha inteira é lida.

7. Nas digitações das teclas *delete* e *backspace* o leitor lê "delete" e a letra ou texto que foi apagado. Esta proposta difere da execução do Orca pelo fato de entendermos que no momento de uma deleção de texto a melhor proposta é justamente a execução da leitura do texto que foi apagado. Entendemos também que a execução do Orca é equivocada, pois uma mesma ação (deleção de texto) é executada de duas maneiras diferentes, em um caso lendo o texto apagado e em outro lendo o caractere a direita do texto que foi apagado.

Pode-se observar com esta rápida comparação que os dois leitores têm qualidades a se ressaltar e defeitos a se considerar, porém os dois podem ser colocados no mesmo nível. O leitor do Orca peca principalmente por em alguns casos não cortar os eventos ao ocorrer outro evento, obrigando o usuário a esperar a fala relacionada a um evento terminar para poder ouvir a fala

do evento que deseja. Já o principal incômodo do leitor desenvolvido é a não leitura das teclas (Ctrl, Alt, Caps Lock, etc.).

Na maioria dos itens relacionados o funcionamento da leitura de ambos os leitores é idêntica, sendo assim, uma possível mudança de leitor de tela por algum usuário teria uma adaptação mais natural. Outros aspectos que podem ajudar uma possível adaptação para quem trocar de leitor de tela é o uso do mesmo sintetizador de voz pelos dois leitores de tela, ou seja, a voz que é pronunciada não sofrerá modificações, além disso, a leitura da maioria dos componentes acontece exatamente igual em ambos os leitores. Isto acontece por que não existe muito o que diferenciar, quando um evento ocorre, o processo natural é ler somente o texto relacionado ao evento, logo todos os leitores executam a leitura de forma semelhante.

O leitor de tela do Orca foi criado por uma equipe com suporte de um grande sistema operacional e vem sendo desenvolvido a alguns anos, enquanto que este projeto foi desenvolvido durante o último ano com base no leitor de tela do xLupa 3.0, por isso, parece que o Orca seja a melhor solução. Mas apesar dos pontos positivos e negativos do leitor quando comparado ao leitor do Orca acreditamos que os leitores são similares, assim, este leitor que evoluiu da primeira versão acoplada ao xLupa pode ser utilizado sem perda de qualidade na leitura de tela.

Como o leitor desenvolvido está no mesmo nível que um dos leitores de tela mais utilizados no Linux, ele se destaca para ser utilizado como *script*, que é o principal objetivo deste projeto. O leitor do Orca não oferece esta funcionalidade, seu código até é aberto a comunidade, porém fica trabalhoso para o desenvolvedor de outro aplicativo entender o código e adaptá-lo. Enquanto que, no *script* desenvolvido a facilidade de incorporação a outros códigos ou uso sem incorporação é muito mais simplificado e é neste quesito a principal vantagem da adoção deste leitor.

Capítulo 5

Conclusão e Trabalhos Futuros

O leitor de tela desenvolvido atende todas as expectativas e requisitos exigidos na fase inicial do projeto. Em termos da execução da leitura de tela, a maioria dos componentes gráficos da tela são lidos da forma esperada e a construção do código como um *script* Python atendeu as expectativas.

Com a comparação realizada com o leitor de tela do Orca teve-se a ideia da qualidade que este leitor de tela alcançou, e podemos ainda destacar que este ainda pode melhorar, com a implementação de alguns requisitos descritos logo mais abaixo.

A inovação deste trabalho ocorre justamente por conta do desenvolvimento do leitor como um *script* Python. A principal justificativa para este tipo de desenvolvimento se deve ao fato de auxiliar outros desenvolvedores que criam outros tipos de sistemas, porém querem o fazê-lo mais acessível aos seus usuários, incorporando na execução de seu sistema um módulo de leitura de tela.

O desenvolvimento correu normalmente, apesar de pequenos contratempos já descritos anteriormente. As ferramentas, linguagens e aplicações utilizadas durante toda a fase de desenvolvimento atenderam as expectativas e auxiliaram o processo de implementação do leitor de tela. A fase de desenvolvimento foi encerrada com a incorporação do código do leitor de tela, que foi testada e realizada no xLupa, obtendo o funcionamento esperado. A próxima versão do aplicativo já deverá ser lançada contendo a nova versão do leitor de tela incorporada[1]. Essa incorporação do código do leitor em outro sistema mostrou que um dos principais objetivos deste trabalho foi alcançado, o *script* Python pode rodar normalmente via terminal ou pode também ser adicionado junto a outro sistema, como no caso do xLupa.

[1] O xLupa pode ser adquirido através do site http://projetos.unioeste.br/campi/xlupa/

O leitor de tela implementado já tem uma qualidade muito boa, porém alguns quesitos podem melhorar a execução e consequentemente a iteração com o usuário.

Uma questão deste leitor que deve ser investigada em um trabalho futuro é a captura das teclas de comando do computador (Alt, Ctrl, Tab, Caps Lock, Pause, Print Screen, Page Up, etc.). Essas teclas são muito utilizadas pelos usuários em alguns programas, porém na maioria dos casos não geram edição de texto, por isso, tivemos dificuldade em implementar este requisito. Devido a importância que estas teclas trazem para a navegação do sistema acreditamos que está funcionalidade precisa ser a primeira a ser realizada no futuro. Em nossas pesquisas não identificamos funcionalidades da AT-SPI que poderiam auxiliar na captura dos eventos que são produzidos ao pressionar destas teclas, por isso, os futuros desenvolvedores podem procurar outros meios para tratar esta questão.

Outra característica que pode aumentar a qualidade do leitor é a leitura de mais eventos. Lendo mais eventos do sistema o leitor pode aumentar a iteração com o usuário e isso é primordial para o sucesso da ferramenta. O leitor de tela pode ser atualizado para conseguir ler mais eventos do sistema ou pode identificar quais os aplicativos mais utilizados pelos usuários para realizar um estudo focando a comunicação entre esse aplicativo e o leitor[2] para verificar se esses aplicativos funcionam corretamente junto ao leitor, caso contrário, uma programação mais focada na comunicação com este aplicativo pode ser realizada para melhorar a leitura, isto não foi implementado ainda neste projeto por que a ideia aqui era criar um módulo genérico que comunica-se com todos os aplicativos do sistema da mesma forma.

Na Seção 4.3 do Capítulo 4 descrevemos a tentativa de utilizar algumas bibliotecas para executar a fala. Neste quesito da implementação pode haver uma maior investigação para descobrir outras possibilidades de execução da síntese de voz. Esta questão não é tão primordial, já que o leitor está executando a leitura de forma correta sem o uso de nenhuma biblioteca, porém seria interessante a utilização de alguma biblioteca para deixar o processo mais simples e mais confiável. Por exemplo, uma adoção de uma biblioteca para tratar o processo de leitura pode, talvez, acabar com um incômodo visto na Seção 4.4.1 deste mesmo Capítulo. O incômodo é a não execução da leitura, em alguns momentos, pelo eSpeak, o que certamente atrapalha o usuário do sistema. Como este problema foi verificado no leitor implementado, no Orca e na execução

[2]É preciso lembrar que a comunicação não ocorre diretamente entre esses dois aplicativos, mas passa por várias outras camadas.

do eSpeak, trabalhamos de forma que o problema era por parte de *hardware* ou um defeito do próprio eSpeak, por isso não trabalhamos muito nesta questão, mas ela pode ser superada com essa adoção de uma biblioteca.

Referências Bibliográficas

[Accessibility 2011]ACCESSIBILITY, G. *What solutions are available in GNOME for people with disabilities*. 2011. Consultado na INTERNET: http://projects.gnome.org/accessibility/solutions.html, 2011.

[Allen, Hunnicutt e Klatt 1987]ALLEN, J.; HUNNICUTT, M.; KLATT, D. From text to speech: The mitalk system. *216 pp*, 1987.

[Aple 2011]APLE. *Accessibility (ApplicationServices/HIServices) Reference*. 2011. Consultado na INTERNET: http://developer.apple.com/library/mac/documentation/Accessibility/Reference/Accessibi 2011.

[Bessa 2011]BESSA, A. *What is Accerciser?* 2011. Consultado na INTERNET: http://live.gnome.org/Accerciser, 2011.

[Blenkhorn e Evans 2000]BLENKHORN, P.; EVANS, G. Architecture and requirements for a windows screen reader. In: IET. *Speech and Language Processing for Disabled and Elderly People (Ref. No. 2000/025), IEE Seminar on*. [S.l.], 2000. p. 1–1.

[Braga 2007]BRAGA, D. Máquinas falantes: Novos paradigmas da língua e da linguística. 2007.

[Canonical 2011]CANONICAL. *ubuntu-br*. 2011. Consultado na INTERNET: http://www.ubuntu-br.org/ubuntu, 2011.

[Castro et al. 2004]CASTRO, A. et al. Sintetizador texto-voz com autômatos adaptativos. *Trabalho de Conclusão de Curso, USP, São Paulo*, 2004.

[Coelho et al. 2004]COELHO, L. et al. Na ponta da língua: Uma nova forma de acesso à informação. In: *Actas da Conferência da Associação Portuguesa de Sistemas de Informação*. Lisboa, Portugal: [s.n.], 2004.

[Cook e Hussey 2001]COOK, A.; HUSSEY, S. Assistive technologies: Principles and practice. Mosby, 2001.

[CPqD 2011]CPQD. *CPqD Texto Fala*. 2011. Consultado na INTERNET: http://www.cpqd.com.br/solucoes-e-produtos/214-cpqd-texto-fala.html, 2011.

[Deller, Proakis e Hansen 2000]DELLER, J.; PROAKIS, J.; HANSEN, J. Discrete-time processing of speech signals. In: INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. [S.l.], 2000.

[Diggs 2011]DIGGS, J. *pyatspi*. 2011. Consultado na INTERNET: http://live.gnome.org/GAP/PythonATSPI, 2011.

[Donovan 1996]DONOVAN, R. Trainable speech synthesis. Citeseer, 1996.

[Englund 2004]ENGLUND, C. *Speech recognition in the JAS 39 Gripen aircraft adaptation to speech at different G-loads, Masters thesis*. Dissertação (Dissertação de Mestrado) — Department of Speech, Music and Hearing, Royal Institute of Technology, Stockholm, 2004.

[eSpeak 2011]ESPEAK. *eSpeak text to speech*. 2011. Consultado na INTERNET: http://espeak.sourceforge.net/, 2011.

[Foundation 2008]FOUNDATION, F. S. *at-spi - Free Software Directory*. 2008. Consultado na INTERNET: http://directory.fsf.org/project/at-spi/, 2011.

[Foundation 2009]FOUNDATION, L. *IAccessible2*. 2009. Consultado na INTERNET: http://www.linuxfoundation.org/collaborate/workgroups/accessibility/iaccessible2, 2011.

[Foundation 2011]FOUNDATION, P. S. *Python Programming Language Official Website*. 2011. Consultado na INTERNET: http://www.python.org, 2011.

[Isaacson 2011]ISAACSON, E. *Accerciser Manual v0.2.0*. 2011. Consultado na INTERNET: http://library.gnome.org/devel/accerciser/1.12/accerciser.html, 2011.

[Körting, Costa e Silva 2005]KöRTING, T.; COSTA, R.; SILVA, F. da. Um sistema texto-fala livre. In: *WSL'05: Workshop sobre Software Livre*. [S.l.: s.n.], 2005.

[Lee 2008]LEE, S. *Python Powered Accessibility*. 2008. Consultado na INTERNET: http://live.gnome.org/Accessibility/PythonPoweredAccessibility, 2011.

[Lemetty 1999]LEMETTY, S. *Review of Speech Synthesis Technology*. Dissertação (Dissertação de Mestrado) — Helsinki University of Technology, 1999.

[Matthew 2002]MATTHEW, R. S. N. *Professional Linux Programando*. 1. ed. [S.l.]: MAKRON BOOKS, 2002.

[Microsoft 2011]MICROSOFT. *UI Automation Overview*. 2011. Consultado na INTERNET: http://msdn.microsoft.com/en-us/library/ms747327.aspx, 2011.

[Microsoft 2011]MICROSOFT, M. *Microsoft Active Accessibility*. 2011. Consultado na INTERNET: http://msdn.microsoft.com/en-us/library/ms697707(v=vs.85).aspx, 2011.

[Oracle 2011]ORACLE. *Java SE Desktop Accessibility - Java Access Bridge For Windows OS*. 2011. Consultado na INTERNET: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136191.html, 2011.

[Parente 2009]PARENTE, E. *Um Módulo de voz em Português do Brasil para o Sistema de Síntese de Voz Festival*. Dissertação (Monografia) — Universidade de Brasília, Brasília, 2009.

[Project 2011]PROJECT, G. *GNOME*. 2011. Consultado na INTERNET: http://www.gnome.org/, 2011.

[Rabiner e Schafer 1978]RABINER, L.; SCHAFER, R. *Digital processing of speech signals*. [S.l.]: Prentice-hall Englewood Cliffs, NJ, 1978.

[Raj 1998]RAJ, A. A. A. *Multi-Lingual Screen Reader and Processing of Font-Data in Indian Languages*. Dissertação (Dissertação de Mestrado) — COPIN – Language Technologies Research Center International Institute of Information Technology, HYDERABAD - INDIA, Fevereiro 1998.

[Ramachandran e Mammone 1995]RAMACHANDRAN, R.; MAMMONE, R. *Modern methods of speech processing*. [S.l.]: Springer, 1995.

[Raman 1996]RAMAN, T. V. Emacspeak direct speech access. In: *ASSETS '96: Second Annual ACM Conference on Assistive Technologies*. Nova York: [s.n.], 1996. p. 32–36.

[Schmi 2011]SCHMI, G. *GNOME Accessibility Architecture (ATK and AT-SPI)*. 2011. Consultado na INTERNET: http://accessibility.kde.org/developer/atk.php, 2011.

[Schroeder 2001]SCHROEDER, J. The fundamentals of text-to-speech synthesis. In: *VoiceXML Review*. [S.l.: s.n.], 2001.

[Sonza, Mello e Moro 2009]SONZA, A. P.; MELLO, F. Z. de; MORO, R. *e-MAG*: Leitores de tela - descrição e comparativo. [S.l.], Dezembro 2009.

[Thatcher 1994]THATCHER, J. Screen reader/2: Access to os/2 and the graphical user interface. In: *Proc. of The First Annual ACM Conference on Assistive Technologies*. Los Angeles - California: [s.n.], 1994. p. 39–47.

[Trivi 2011]TRIVI, J.-M. *An introduction to Text-To-Speech in Android*. 2011. Consultado na INTERNET: http://android-developers.blogspot.com/2009/09/introduction-to-text-to-speech-in.html, 2011.

[Valiati 2000]VALIATI, J. F. *Reconhecimento de Voz para Comandos de Direcionamento por meio de Redes Neurais*. Dissertação (Dissertação de Mestrado) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, Porto Alegre, Novembro 2000.