CENTRO UNIVERSITÁRIO DA FEI

WESLEY DAL'COL VON DOELINGER

CONTROLE DE *DESKTOP* POR VOZ

São Bernardo do Campo
2012

WESLEY DAL'COL VON DOELINGER

CONTROLE DE *DESKTOP* POR VOZ

Trabalho de Conclusão de Curso I apresentado ao curso de Ciência da Computação do Centro Universitário da FEI, como requisito parcial para avaliação, orientado pelo professor Doutor Rodrigo Filev Maia.

São Bernardo do Campo
2012

WESLEY DAL'COL VON DOELINGER

CONTROLE DE *DESKTOP* POR VOZ

Trabalho de Conclusão de Curso I - Centro Universitário da FEI

Comissão julgadora:

_____

Professor Doutor Rodrigo Filev Maia
Orientador

_____

Professor Doutor Plínio Tomaz Aquino Junior
Examinador

_____

Professor Doutor Paulo Sérgio Silva Rodrigues
Examinador

São Bernardo do Campo

22 de junho de 2012

RESUMO

Ao longo do desenvolvimento dos sistemas computacionais, teclado, mouse e monitor consolidaram-se como os dispositivos de entrada e saída mais usados, influenciando como as interfaces das aplicações são projetadas. Entretanto outros dispositivos como câmera, microfone e caixas de som também podem ser utilizados no desenvolvimento de interfaces complementares. Assim, este trabalho propõe a criação de um sistema capaz de controlar a interface gráfica de um computador através de comandos de voz, permitindo ao usuário acessar as funcionalidades mais comuns de um *desktop* e suas aplicações.

Palavras-chave: Reconhecimento de fala. Comandos de voz. Controle de aplicações.

ABSTRACT

Throughout the development of computer systems, keyboard, mouse and monitor were consolidated as commonly used input and output devices, influencing how the applications interfaces are designed. However, other devices such as camera, microphone and speakers can also be used to develop additional interfaces. Thereby, this paper proposes the creation of a system capable of control the graphical user interface of a computer via voice commands, allowing the user to access the most common features of a desktop and applications.

Keywords: Voice recognition. Voice commands. Application control.

SUMÁRIO

1 INTRODUÇÃO

Ao longo da história da computação diversos tipos de computadores, sistemas e periféricos foram construídos, sendo que enquanto alguns caíram em desuso, outros se consolidaram e persistem até hoje.

Sendo os mais utilizados, esses dispositivos influenciam como as aplicações são projetadas. Um sistema de *eye tracking*, por exemplo, é projetado para ser utilizado em um computador com câmera.

Algumas vezes, dois ou mais desses periféricos ou tecnologias são combinados para criar uma aplicação mais interessante. Um editor de texto, por exemplo, é desenvolvido tendo-se em mente que ele será utilizado em um ambiente com interface gráfica, teclado e mouse, combinando os recursos que esses dispositivos oferecem.

Da mesma forma, o foco deste trabalho é combinar algumas tecnologias existentes para dar ao usuário a possibilidade de interagir com o computador de outra maneira, através de comandos de voz.

1.1 Objetivo

O objetivo principal deste trabalho é criar um sistema que receba uma entrada de voz através de um microfone, decodifique o sinal de áudio, reconheça possíveis comandos e então os encaminhe para o sistema operacional.

Tem-se ainda como objetivo secundário a proposição de uma arquitetura que torne a aplicação independente de *engine* de reconhecimento de fala e que permita a adição e remoção de conjuntos de comandos conforme os programas em execução no desktop.

1.2 Metodologia

Para atingir o seu objetivo, este trabalho solucionará o problema em duas etapas: o reconhecimento da fala e a execução dos comandos.

A primeira etapa será realizada por um componente que vai encapsular um reconhecedor de fala. Dado um sinal de áudio como entrada, é produzido como saída um texto que melhor represente o que foi dito.

Em seguida, um segundo componente receberá como entrada o texto produzido anteriormente sobre o qual será feita uma análise para encontrar comandos conhecidos e então executá-los.

1.3 Desafios

O primeiro desafio apresenta-se no momento da captura da fala, no qual as condições ambientais e os equipamentos utilizados podem influenciar significativamente na qualidade do sinal de áudio utilizado como entrada do sistema (SELTZER, 2003).

O segundo concentra-se na maneira de como a fala é gerada, por exemplo, se for realizada de forma planejada - com o uso de comandos previamente determinados - os sistemas de reconhecimento automático tendem a obter uma maior eficiência. Já no caso de um discurso espontâneo - como em um diálogo - existe uma degradação significativa na qualidade do reconhecimento (NEDEL, 2004).

O terceiro está relacionado com a estrutura básica das palavras, os fonemas. Dependendo do sotaque e do usuário, os fonemas podem ter a sua duração modificada em relação ao que o sistema está preparado para reconhecer (*ibidem*).

Outro grande problema é definir o escopo de comandos que o sistema aceitará. Não há como saber antecipadamente, quais programas estão instalados na máquina do usuário, suas versões e capacidades. Isso afeta a forma como a arquitetura do sistema será desenhada.

Além disso, para serem compatíveis, as aplicações existentes atualmente precisam ter sido desenvolvidas com a capacidade de aceitar comandos externos de alguma forma. Isso impõe um limitador na abrangência do sistema proposto.

1.4 Potencial de uso

Conforme citado anteriormente, este trabalho propõe uma forma de entrada de dados complementar não tendo, portanto, a pretensão de ser um esforço especificamente nas áreas

de acessibilidade ou usabilidade. Entretanto é notável que os principais beneficiados por esta tecnologia serão as pessoas com alguns tipos de inabilidades motoras e/ou visuais.

De acordo com os dados do último censo do Instituto Brasileiro de Geografia e Estatística (IBGE), cerca de 14,5% da população brasileira se declara portadora de alguma deficiência física ou mental, o que em números absolutos representa mais de 24,5 milhões de pessoas. Proporcionalmente, a maior incidência está nas regiões Norte e Nordeste. Além disso, em 75,1% dos casos as deficiências são físicas, motoras ou de visão. (IBGE, 2005)

Para essa parcela da população, tarefas corriqueiras como enviar um email, ou digitar uma carta tornam-se desafios muito mais complexos do que deveriam, pois, atualmente as interfaces mais comuns entre um usuário e um computador são baseadas na interação entre teclado, mouse e monitor.

Ainda que o uso destes periféricos seja razoavelmente simples para a maioria das pessoas, para alguém com tetraplegia, por exemplo, operar um teclado ou *mouse* diretamente é algo no mínimo improvável. Da mesma forma, uma pessoa com dificuldades visuais tem problemas para interagir com pequenos *widgets* e ícones, ou ainda com telas que usam esquemas de cores com pouco contraste.

A expectativa é que embora não seja um projeto especificamente voltado para pessoas com essas dificuldades, este trabalho possa ser bem aproveitado por elas ajudando a melhorar a sua qualidade de vida.

2 FUNDAMENTAÇÃO

2.1 Introdução

Neste capítulo serão apresentados brevemente os conceitos básicos que fundamentam este trabalho, bem como algumas aplicações existentes que foram analisadas durante as pesquisas preliminares nas plataformas Windows, Mac OSX e Linux.

Tanto o Windows quanto o Mac OSX já oferecem nativamente seus próprios sistemas de reconhecimento de fala, com níveis excelentes de precisão. No Linux, entretanto, não há um pacote oficial para essa tarefa.

Foram analisadas algumas iniciativas para a plataforma Linux, cada uma apresentando variados graus de sucesso. Dentre as observadas, a que chamou mais atenção foi o Gnome Voice Control (GVC). Ele é um sistema que permite controlar um *desktop* Gnome por voz, através de um pequeno conjunto de comandos pré-estabelecidos (GNOME LIVE, 2011).

A abordagem do GVC é parecida com a que será utilizada aqui e será explicada em detalhes no capítulo de modelagem. Resumidamente, ele usa uma *engine* de reconhecimento de fala existente, no caso o CMU Sphinx, para obter os comandos a partir de um sinal de voz. Em seguida os comandos são executados através de bibliotecas específicas.

Este trabalho não propõe novas técnicas de reconhecimento de fala ou melhorias nas existentes. Ele utilizará *engines* de reconhecimento existentes. Portanto, os conceitos fundamentais sobre reconhecimento de fala serão abordados apenas superficialmente, para efeito de contextualização.

Adicionalmente, também será apresentado o básico sobre envio de comandos para outras aplicações, dentro do contexto de um *desktop* Linux. A escolha desse ambiente será melhor explicada no capítulo sobre a modelagem.

2.2 Sistemas de Reconhecimentos de Fala

Um sistema de reconhecimento de fala é basicamente uma aplicação que toma como entrada um sinal de áudio (voz) e após um processamento retorna como saída um conjunto de palavras, independente do locutor. Um reconhecedor de voz, por outro lado preocupa-se principalmente com a identificação do locutor (TOGNERI; PULLELLA, 2011).

O objetivo de um sistema de reconhecimento de fala pode ser definido então como a obtenção de uma sequência de palavras W, a partir de um sinal de áudio X. Dentre outras abordagens, isto pode ser visto como um problema de decisão, no qual é preciso encontrar a melhor sequência W que represente a informação embutida em X (CHOU; JUANG, 2003).

2.2.1 Princípios Básicos

O processamento nos sistemas de reconhecimento de fala pode ser subdividido em duas etapas, a extração de características e a classificação baseada nas características obtidas (NEDEL, 2004).

No reconhecimento automático de fala, o sinal observado é uma medida da modificação da pressão do ar gravado por um transdutor. A fala é capturada como um sinal unidimensional variando no tempo. Em seguida o extrator de características converte o sinal de fala em uma sequência parametrizada de vetores de características que serão utilizados pela fase de classificação (*Ibidem*).

2.2.2 Extração de Características

O objetivo da fase de extração de características é obter um conjunto de valores que representem o sinal de entrada (SELTZER, 2003), ou seja, dada uma grandeza física qualquer, a extração de características é fase responsável por obter uma representação computacional dessa grandeza.

No contexto de reconhecimento de fala o mais usual é extrair um conjunto de Coeficientes Mel-Cepstrais. Para isso o sinal de áudio é dividido em frames tipicamente com duração entre 20ms e 30ms, sendo aplicada sobre eles uma transformada de Fourier discreta, num processo chamado Short-Time Fourier Transform (STFT) (*Ibidem*).

Em seguida é calculado o quadrado dos coeficientes SFTF para descartar as informações de fase - os valores negativos são mapeados para outros valores positivos - que não são importantes para o reconhecimento (*Ibidem*).

Na etapa seguinte, sobre os espectros de frequência, são aplicados os filtros Mel e o resultado é posto em uma escala apropriada produzindo um vetor de coeficientes Mel-Cepstrais (*Ibidem*).

Como resultado desse processo, são obtidos os Coeficientes Mel-Cepstrais que representam o sinal de áudio que foi dado como entrada.

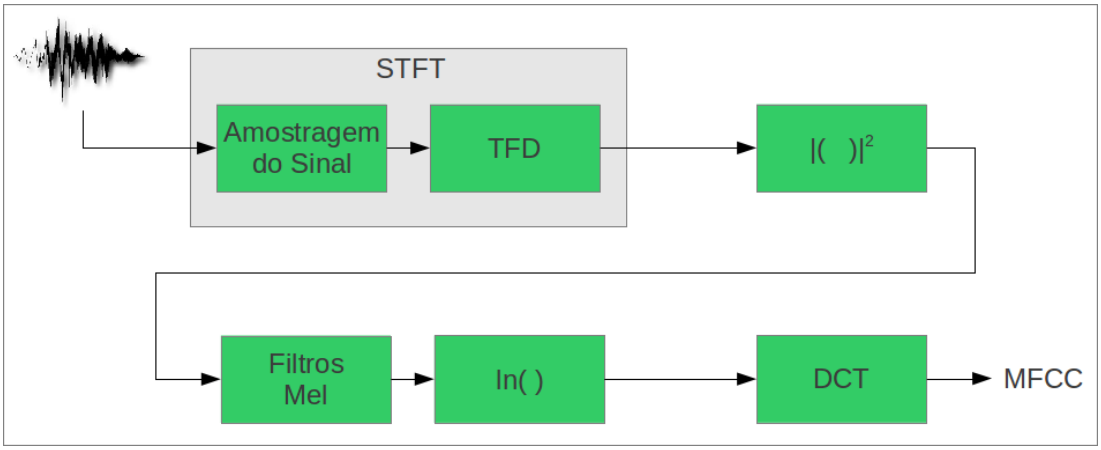

Figura 1 – Processo de extração de característica
Fonte: Autor “adaptado de” Nedel, 2004, p. 5

2.2.3 Classificadores

Os classificadores são elementos computacionais utilizados para determinar a qual categoria uma determinada entrada pertence.

No caso do reconhecimento de fala, eles são utilizados para identificar os elementos de linguagem que melhor representem a informação que estava embutida no sinal de áudio, ou seja, eles usam os vetores de característica para tentar descobrir o que foi dito pelo locutor (LI, 2005).

Em outras palavras, as categorias são as palavras ou expressões de um idioma. O classificador precisa descobrir em qual expressão cada entrada melhor se encaixa.

Para solucionar esse problema podem ser utilizadas entre outras abordagens, redes neurais ou como é mais frequente na literatura de reconhecimento de fala, Modelos Ocultos de Markov (HMM, do inglês *Hidden Markov Models*) (CHOU; JUANG, 2003).

A maioria dos sistemas tidos como "estado da arte" no reconhecimento de fala, entretanto, optam por uma visão estatística do problema, tentado encontrar as palavras que tenham

a maior probabilidade de representar o sinal de áudio (*Ibidem*), o que resulta na escolha de classificadores baseados em HMM.

2.2.3.1 Modelos Ocultos de Markov

Antes de falar sobre Modelos Ocultos de Markov, é preciso definir o que é uma Cadeia de Markov.

Uma Cadeia de Markov pode ser definida como uma máquina de estados com um conjunto de estados S = {s1; s2; ... ; sn} e um conjunto de transições t = {t1, t2, ... , tm}. A cada transição do estado s[i] para o estado s[j] está associada uma probabilidade p[i][j] que depende apenas do estado atual s[i]. (GRINSTEAD, SNELL, 1997)

A partir de um estado inicial previamente definido, são aplicadas sucessivas transições até chegar a um estado final também conhecido. (*Ibidem*)

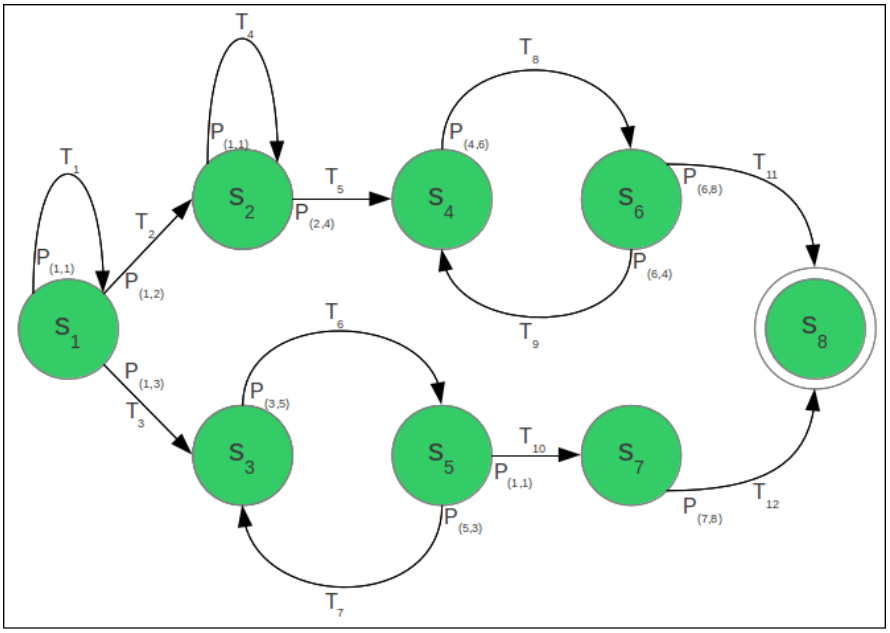

Figura 2 – Cadeia de Markov
Fonte: Autor

Um modelo oculto de Markov é uma cadeia de Markov na qual os estados intermediários não são observáveis. Apenas os estados finais podem ser vistos pelo usuário (STAMP, 2012).

Eles são utilizados, por exemplo, para descobrir qual a próxima palavra mais provável de ocorrer em uma determinada frase, onde os passos intermediários não são importantes, mas sim a palavra escolhida no final do processo.

Os HMMs são utilizados pelos modelos acústicos e modelos de linguagem, que serão brevemente explicados a seguir.

2.2.3.2 Modelos Acústicos

Um modelo acústico é um modelo oculto de Markov que associa uma probabilidade de ocorrência de uma determinada palavra pertencente a um vocabulário conhecido, dado um conjunto de características de entrada (KUHN; MORI, 1990).

Em outras palavras, dado um dicionário mapeando um vocabulário previamente conhecido e sua representação fonética, o modelo acústico ajuda a encontrar qual é a palavra desse dicionário que melhor representa a sequência de fonemas de entrada.

2.2.3.3 Modelo de Linguagem

Um modelo de linguagem é um modelo oculto de Markov que associa uma probabilidade de ocorrência a uma sequência ordenada de palavras $W = \{w1,...,wn\}$ em um determinado idioma. Ele é utilizado para reduzir o espaço de busca de termos (*Ibidem*).

Por exemplo, dado um idioma contendo 20000 palavras e uma sentença qualquer composta por três palavras, existem $20000^3 \approx 8x10^{12}$ sentenças possíveis, ou seja, para encontrar a informação "gato mia alto" seria necessário comparar a entrada com as 8 bilhões de sentenças possíveis.

Com um modelo de linguagem, tem-se que probabilidade de ocorrência de determinadas sentenças é diferente da probabilidade de ocorrência de outras, assim o algoritmo de busca não precisa testar as opções menos prováveis.

2.2.3.4 Modelos de duração

Um modelo de duração de palavras é um outro modelo estatístico utilizados em alguns trabalhos de reconhecimento de fala, que penaliza hipóteses levantadas pelo sistema de reconhecimento que estejam fora da duração média de dada palavra (YNOGUTI, 1999).

Para que essa duração média seja razoavelmente precisa, é necessário estimá-la a partir de uma grande base de dados, o que a torna inviável em situações onde o vocabulário é muito grande (*Ibidem*).

2.2.4 Considerações

Os modelos estatísticos baseados em HMM são utilizados para determinar a probabilidade de ocorrência das sequências de palavras em determinado idioma. Eles são combinados de forma a tentar obter a melhor resposta possível dada uma sequência de entrada e uma base de dados razoavelmente grande representando o idioma.

Várias são as dificuldades, desde a presença de ruídos, às variações de locução que podem aparecer devido a sotaques, diferentes velocidades de pronúncia, ou ainda por problemas temporários com a voz do locutor. Devem ser ainda consideradas as alterações causadas pelas diferentes características físicas do *hardware* utilizado para captação do sinal.

2.3 Enviando Comandos para Aplicações

O segundo componente do sistema proposto por este trabalho é o responsável por receber o texto decodificado pelo sistema de reconhecimento de fala, verificar se ele representa algum comando, para em caso afirmativo, enviá-lo em seguida às aplicações.

Isto implica que as aplicações precisam dispor de algum mecanismo através do qual possam receber comandos.

Infelizmente, a maioria das aplicações comuns em um *desktop* atual não são sensíveis a uma interface de comandos, o que é razoável, pois, para isso cada uma teria que implementar seu próprio interpretador de comandos. Além disso, aplicações *desktop* normalmente são

planejadas para serem executadas em um ambiente previamente conhecido, com suporte a monitor, teclado e mouse, por exemplo, onde uma interface de comandos não é necessária.

Para subjulgar esse obstáculo, uma abordagem possível é aproveitar a arquitetura do X Window System que é capaz de enviar comandos para as janelas das aplicações, conforme será detalhado adiante. Essa é a abordagem utilizada pelo Gnome Voice Control.

2.3.1 X Window System

O X Window System, também conhecido como X11 ou simplesmente X, é um conjunto de softwares implementados sobre uma arquitetura cliente-servidor responsáveis por criar um ambiente gráfico em sistemas baseados em UNIX ou compatíveis (SCHEIFLER; GETTYS, 1986).

Um sevidor X é um programa que interage diretamente com os dispositivos de entrada e saída, fornecendo uma interface para que outros programas, os clientes, acessem dispositivos como teclado, mouse e display de forma transparente (*Ibidem*).

Os clientes e o servidor trocam mensagens pela rede de forma que uma aplicação cliente executando local ou remotamente, possa requisitar ao servidor que ele lhe entregue as entradas e desenhe na tela as saídas (PARAVAN, 1994).

Pode parecer estranho que um servidor esteja na máquina do usuário, enquanto que os clientes executam remotamente, mas o foco está nas aplicações e não no usuário. Uma aplicação cliente se conecta a um servidor X e requisita lhe determinadas ações. Ele recebe essas requisições e as executa ou não.

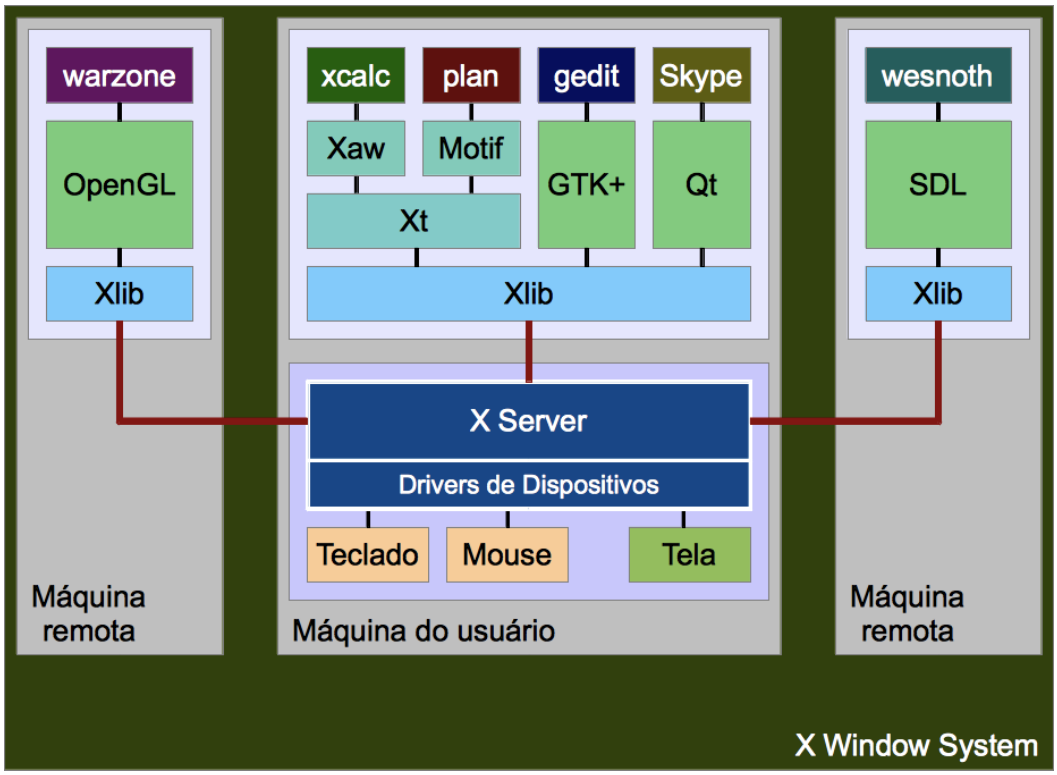

Figura 3 – X Window System
Fonte: Autor “adaptado de” Nye, 1995, p.9

Essa arquitetura também permite que uma aplicação cliente envie mensagens a outros clientes, ainda que indiretamente, sendo a característica a ser explorada neste trabalho para envio de comandos a outras aplicações.

2.3.2 Camadas e Componentes

A comunicação entre clientes e servidor é feita através da troca de mensagens do protocolo do X Window System, um formato que é pouco amigável para humanos (SCHEIFLER, 2004). Assim algumas camadas de abstração foram adicionadas para facilitar o trabalho dos desenvolvedores de aplicações.

Isso significa que embora seja possível montar manualmente as mensagens do protocolo e enviar diretamente para as aplicações, normalmente utiliza-se bibliotecas que expõem uma *API* mais amigável.

Um dos desafios deste trabalho também é escolher o nível adequado de abstração que permita um desenvolvimento simples ao mesmo tempo em que forneça as funcionalidades necessárias.

2.3.3 Clientes do Protocolo X

A primeira camada de abstração está nos clientes do protocolo, que encapsulam o sistema de mensageria e fornecem funções para interação básica com o servidor X.

Eles não dão suporte a *widgets* como botões, menus, etc. Os *widgets* são implementados por outras bibliotecas escritas a partir dos clientes.

As implementações mais conhecidas, embora existam ainda outras abordagens, são a CLX (SCHEIFLER, 1989), escrita em Common Lisp e a Xlib (SCHEIFLER; GETTYS, 2002), escrita em C.

Ao utilizar uma dessas bibliotecas, o programador passa a não precisar conhecer todos os detalhes do protocolo em si, mas ainda dispõe de um arcabouço razoavelmente limitado para escrever aplicações não triviais.

2.3.4 *Toolkits*

Os *toolkits* são implementações de abstrações de um nível um pouco mais alto que as bibliotecas clientes de protocolo, sendo construídos a partir delas. Eles fornecem suporte a *widgets* e ao ferramental necessário para desenvolver aplicações gráficas.

O primeiro deles, chamado X Toolkit Intrinsics (MCCORMACK, ASENTE, SWICK, 1994), ou simplesmente Xt, fornece funções para criação e manipulação de *widgets*, mas não implementa nenhum widget em si. Os widgets são implementados por outras bibliotecas, como o Xaw e o Motif.

Atualmente o Xt caiu em desuso, sendo substituído por outros toolkits mais modernos e completos como GTK+ (GNOME, 2011) e o Qt (NOKIA CORPORATION, 2011).

Tanto o Qt quanto o GTK+ fornecem todo o ferramental necessários para criação de *widgets*, janelas, tratamento de eventos, utilizando abstrações de alto nível, que são mais próximas do vocabulário dos desenvolvedores e designers de aplicação, do que de engenheiros de protocolo, permitindo assim a criação de aplicações cada vez mais sofisticadas e simples de manter.

2.3.5 Bibliotecas para controle de janelas

Construídas sobre os toolkits, algumas bibliotecas permitem certo nível de acesso e controle sobre janelas de outras aplicações. Sendo normalmente utilizadas em tarefas de gerenciamento, elas fornecem algumas das funcionalidades necessárias para enviar comandos para as aplicações alvo.

Entretanto, como suas finalidades não são especificamente para construir uma interface de comandos para aplicações que não são desenhadas para ter uma, será preciso combinar mais de uma delas para atingir o objetivo desejado.

A primeira biblioteca a ser analisada é a libwnck - Window Navigator Construction Kit (GNOME DEV CENTER, 2011). Ela é uma biblioteca que pode ser utilizada na confecção de sistemas que necessitam exercer algum controle e gerenciamento sobre outras janelas, como selecioná-las pelo nome, maximizar, minimizar, entre outras.

A segunda a se considerar é a lib AT-SPI (Assistive Technology Service Provider Interface) (GNOME DEV CENTER, 2011). Ela foi criada dentro do projeto de acessibilidade do Gnome, mas sua especificação foi adotada em outros projetos, como por exemplo, Java/Swing, Mozilla suite, OpenOffice.org e Qt.

Ela permite emular o envio de eventos de teclado e mouse para janelas que suportem sua especificação.

Dessa forma, combinando a libwnck e a libatspi é possível através de uma aplicação, simular comandos de teclado e mouse e enviá-los a uma outra aplicação.

2.4 Considerações

O reconhecimento de fala é uma das etapas fundamentais para o sucesso deste projeto. Entretanto ele não foca no desenvolvimento ou aprimoramento das técnicas atualmente em uso. Ele confia e depende de sistemas que já foram implementados com esse fim.

Na área de envio de comandos para as aplicações, uma das dificuldades está no fato de que elas não foram concebidas expondo uma interface de comandos, o que no contexto de um *desktop* é normal.

Para contornar este problema, este trabalho vai utilizar uma abordagem que consiste em emular eventos de teclado e mouse através de bibliotecas existentes, para em seguida enviar esses eventos para as janelas das aplicações.

3 MODELAGEM

Este capítulo apresenta a arquitetura do sistema proposto e discute algumas das decisões de projeto adotadas no processo de desenvolvimento. Ele é subdividido em cinco seções conforme mostrado a seguir.

A seção 3.1 apresenta a análise de requisitos do sistema, e a partir deles são construídos na seção 3.2 os casos de uso. A arquitetura proposta é apresentada na seção 3.3, sendo seguida pela seção 3.4 que descreve os fluxos de execução e de dados da aplicação. Por último, na seção 3.5 são discutidas brevemente algumas escolhas feitas durante esta fase do projeto.

3.1 Requisitos do Sistema

Os requisitos de um sistema são a descrição do que o sistema deve fazer, podendo ser classificados como funcionais ou não-funcionais (SOMMERVILLE, 2010. p. 83,84).

Enquanto os requisitos funcionais definem cada uma das funcionalidades que o sistema deve ter, os requisitos não-funcionais são restrições que se aplicam ao sistema como um todo, como performance e segurança por exemplo.(*Ibidem*, p 84, 85)

A elicitação de requisitos serve para definir claramente o que o sistema precisa fazer, mas mesmo essa distinção entre requisitos funcionais e não-funcionais nem sempre é tão clara como sugerem as suas definições (*Ibidem*, p. 85).

Dessa forma, partindo-se dos capítulos anteriores, é possível definir uma "especificação informal" do sistema como "desenvolver uma aplicação que permita a um usuário controlar um *desktop* Linux através de comandos de voz", e a partir dela derivar a especificação formal dos requisitos do projeto.

3.1.1 Requisitos Funcionais

Este projeto possui dois requisitos funcionais fundamentais que não apenas definem o seu funcionamento básico, mas também balizam a própria arquitetura dos seus componentes.

3.1.1.1 Requisito 1 - Reconhecer comandos de voz

O primeiro requisito funcional fundamental deste sistema é reconhecer comandos de voz. Ele deve ser capaz de receber um sinal de áudio (voz), decodificá-lo e interpretá-lo como um comando previamente determinado, ou determinar que o sinal de áudio não representa nenhum comando conhecido.

3.1.1.2 Requisito 2 - Comandar uma aplicação em um *desktop* GNU/Linux

O segundo requisito funcional fundamental deste sistema é poder comandar uma aplicação em um *desktop* Linux. Dado um comando previamente definido, obtido a partir do requisito 1, ele precisa ser capaz de executar esse comando sobre alguma das aplicações executando no *desktop*.

3.1.2 Requisitos Não-funcionais

Por não consistir em um sistema crítico, mas sim uma forma auxiliar de comandar aplicações no *desktop*, este projeto apresenta poucos requisitos não-funcionais.

3.1.2.1 Performance

O Sistema precisa reconhecer os comandos e executá-los rapidamente comportando-se como os outros dispositivos de entrada usuais. Não precisa responder em tempo real, mas não pode demorar demais a ponto de inviabilizar o seu uso como dispositivo de entrada.

Neste momento ainda não são definidos parâmetros formais para uma medida de performance, mas o tempo de resposta do sistema pode ser utilizado para isso.

3.1.2.2 Acurácia

O sistema precisa atingir um bom nível de acerto no reconhecimento de comandos para que o usuário não tenha que constantemente repeti-los. Novamente ainda não são definidos parâmetros formais, mas ela pode ser calculada levando-se em consideração a taxa de erro, por exemplo.

3.1.3 Ambiente Operacional

O sistema concebido por este projeto deverá ser executado em um *desktop* padrão executando Ubuntu 12.04, Precise Pangolin.

Para a captura de áudio será utilizado somente os microfones normalmente presentes em *desktop*s comuns, ou seja, não será utilizado nenhum microfone especial.

Como especificação mínima de hardware estabelece-se os requisitos recomendados para a execução do Ubuntu 12.04, conforme seu manual de instalação (UBUNTU DOCUMENTATION TEAM, 2012).

3.1.4 Escopo do Projeto

Como apresentado anteriormente o objetivo deste projeto é o desenvolvimento de um sistema que permita o controle de um *desktop* por comando de voz.

Para executar esta tarefa será utilizada uma *engine* de reconhecimento de voz existente, ou seja, este projeto não pretende implementar ou melhorar nenhum algoritmo de reconhecimento de voz, mas sim utilizar uma ferramenta que já faça isso.

Ainda de acordo com o capítulo 2, como a execução de comandos depende do sistema de janelas, este projeto limita-se a executar os comandos disponíveis para as janelas das aplicações, ou seja, ele não vai poder executar tarefas que não estão disponíveis atualmente.

3.2 Casos de Uso

3.2.1 Diagrama de Caso de Uso

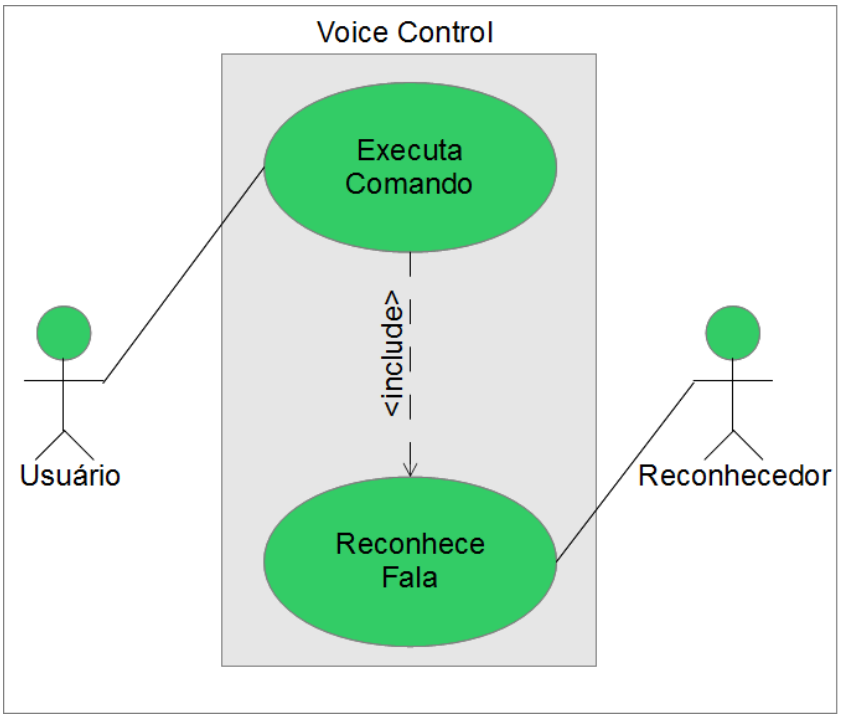

Figura 4 – Diagrama de Caso de Uso
Fonte: Autor

3.2.2 Detalhamento do Caso de Uso

3.2.2.1 Caso de Uso 1 - Executa Comando

Tabela 1 – Caso de uso 1 – Executa comando

Sumário	O Usuário executa o comando por voz.
Ator Primário	Usuário
Precondições	O sistema de captação de áudio deve estar funcionando perfeitamente.
Fluxo Principal	1. Captura a fala do usuário. 2. Envia o áudio capturado para o reconhecedor. 3. Recebe a sentença reconhecida. 4. Busca comando na sentença reconhecida. 5. Executa o comando reconhecido.
Fluxo de Exceção	1. Caso não exista comando na sentença reconhecida nada será executado.
Pós-condições	Nenhuma
Inclusões	Reconhece Fala

Fonte: autor

3.2.2.2 Caso de Uso 2 - Reconhece Fala

Tabela 2 – Caso de uso 2 – Reconhece fala

Sumário	Decodifica o sinal de áudio.
Ator Primário	Reconhecedor
Precondições	Deve existir áudio para ser reconhecido.
Fluxo Principal	1. Recebe o áudio capturado. 2. Aplica algoritmos de reconhecimento de fala. 3. Cria um texto com a sentença reconhecida. 4. Retorna a sentença reconhecida.
Fluxo de Exceção	1. Caso nenhuma sentença seja reconhecida será retornado uma sentença vazia.
Pós-condições	Uma sequência de palavras reconhecidas.

Fonte: autor

3.3 Arquitetura

A arquitetura básica do sistema pode ser descrita pelo diagrama de arquitetura. Nele é possível ver três blocos principais, o reconhecedor de fala, o executor de comandos e os *plugins*.

O reconhecedor de fala é a parte responsável por transformar o sinal de áudio falado pelo usuário em texto.

O texto gerado pelo reconhecedor de fala então alimenta o executor de comandos que vai verificar se esse texto é algum dos comandos conhecidos para em seguida executá-lo.

Os *plugins* são utilizados de duas formas distintas. No reconhecedor de fala ele é utilizado para manter o sistema independente da *engine* de reconhecimento. No executor de comandos eles são utilizados para permitir que as aplicações exportem os comandos que reconhecem, de forma a poder carregá-los dinamicamente.

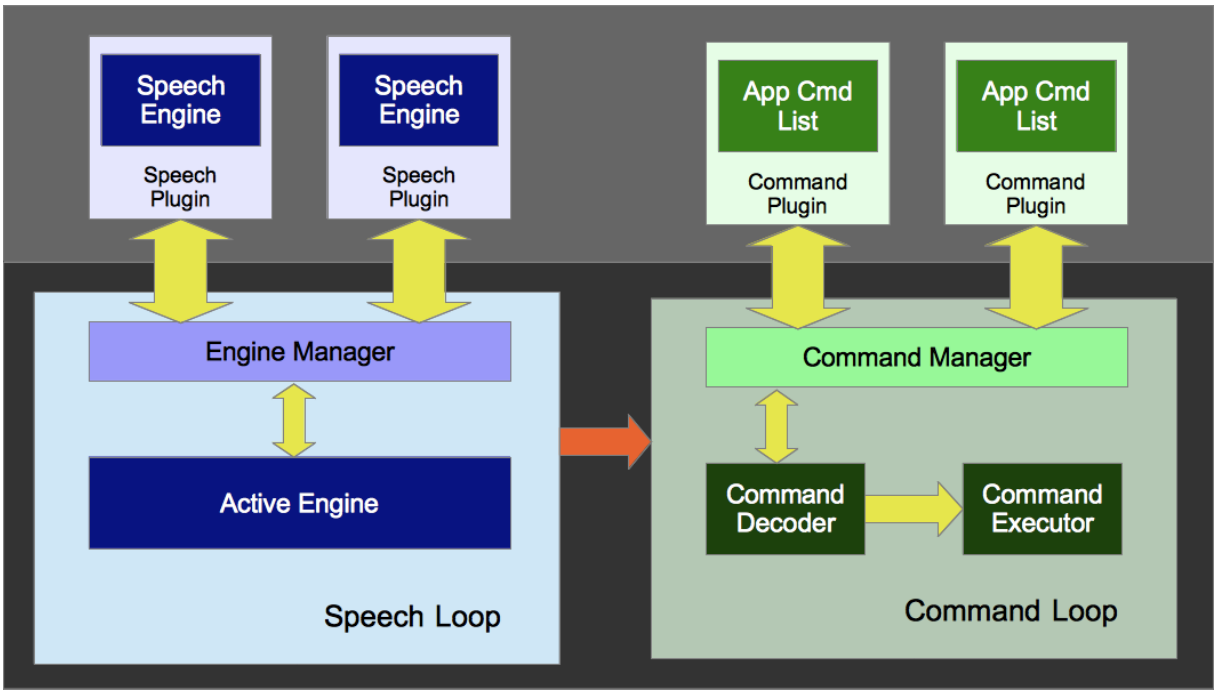

Figura 5 – Diagrama de Arquitetura
Fonte: Autor

A abordagem de *plugins* permite não só que o sistema torne-se independente da *engine* de reconhecimento, como também o exime da responsabilidade de ter que conhecer todos os comandos de antemão.

Cada aplicação que quiser se tornar compatível, precisa apenas implementar um *plugin* com os seus comandos. O executor vai carregá-lo assim que a aplicação estiver disponível, tão cedo quanto possível. A partir daí ele passa a conhecer os comandos que a aplicação exportou.

3.4 Fluxos

Esta seção discute brevemente o ciclo de execução da aplicação e os diagramas de fluxo de dados.

3.4.1 Ciclo de execução

O ciclo de execução da aplicação é ilustrado no fluxograma abaixo.

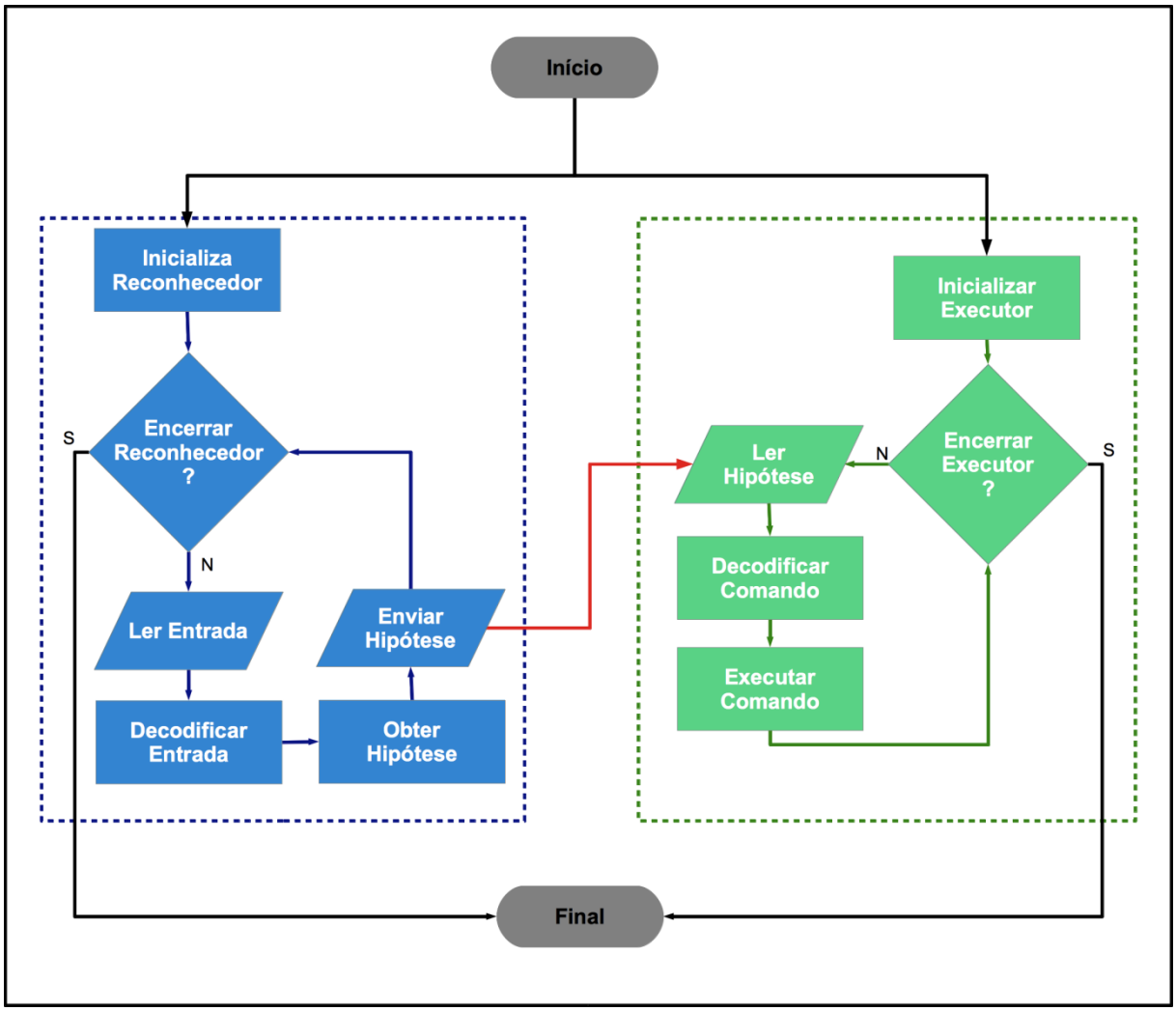

Figura 6 – Fluxo de execução da aplicação
Fonte: Autor

Após a inicialização da aplicação, que compreende entre outras coisas o carregamento dos gerenciadores e *plugins* adequados, são disparados os dois componentes principais, o reconhecedor, em azul e o executor em verde. Em seguida cada um inicia seu próprio loop independentemente.

Enquanto o reconhecedor não receber a ordem para encerrar, ele repetidamente lê a entrada de áudio do microfone, decodifica, determina o texto mais provável que corresponde à entrada, o que é chamado de hipótese e envia ao executor.

O executor por sua vez, lê a hipótese produzida pelo reconhecedor, busca na sua lista de comandos por um que corresponda ao texto da hipótese, tenta executá-lo e repete este ciclo até que lhe seja ordenado encerrar.

Por uma questão de simplificação, toda vez que o executor falhar em encontrar um comando correspondente à entrada, ele entende *No Operation* (NOP).

O encerramento de qualquer um dos dois loops implica necessariamente no encerramento do sistema.

Note que o reconhecedor não entrega um comando, mas sim um texto que pode ou não ser um comando. Cabe ao executor determinar se o texto recebido é ou não um comando válido antes de aceitá-lo.

Embora a análise de requisitos sugira que o reconhecedor entregue comandos, eles não dizem respeito à *engine* de reconhecimento de voz, mas sim às aplicações que o executor reconhece. Dessa forma mantemos o baixo acoplamento entre os elementos do sistema.

3.4.2 Diagramas de Fluxo de Dados

Os diagramas de fluxo de dados, ou DFDs, descrevem o ponto de vista do fluxo de dados dentro da aplicação, ou seja, o processo como eles são transformados desde a entrada até a sua saída. (PRESSMAN, 2009. p.187)

Neste projeto o fluxo compreende o caminho percorrido pelos dados enquanto são transformados de sinais de áudio em um microfone para eventos de entrada em uma aplicação.

3.4.2.1 DFD Nível 0

O diagrama de fluxo de dados nível 0 oferece uma visão geral dos dados sendo transformados pela aplicação. Um sinal de áudio é convertido pelo sistema em um evento que é enviado ao *X Window System*.

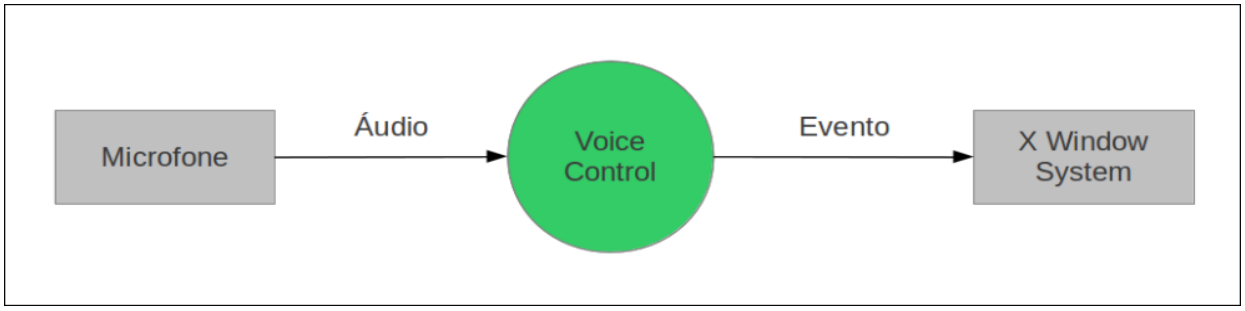

Figura 7 – Diagrama de Fluxo de Dados Nível 0
Fonte: Autor

3.4.2.2 DFD Nível 1

O diagrama de fluxo de dados nível 1 apresenta uma visão levemente mais detalhada do processo de transformação dos dados conforme eles percorrem o sistema.

O sinal de áudio é transformado em texto pelo reconhecedor de fala. Esse texto é encaminhado ao executor de comandos que o transforma em um evento para em seguida enviá-lo ao *X Window System*.

É possível notar que o fluxo dos dados é linear, ou seja, o sistema comporta-se como uma linha de produção, aplicando uma nova transformação a cada nova etapa sem ciclos.

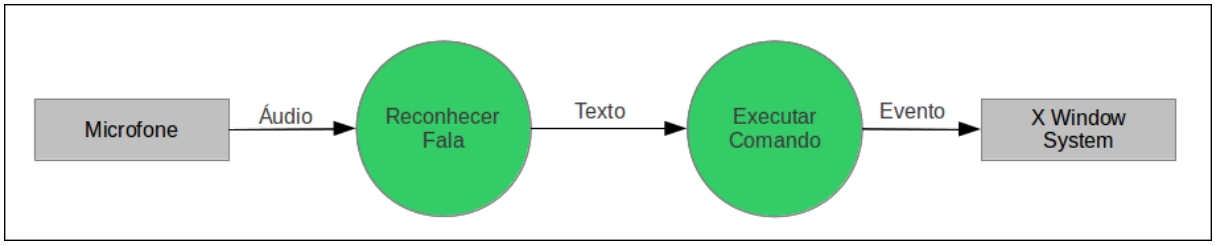

Figura 8 – Diagrama de Fluxo de Dados Nível 1
Fonte: Autor

3.5 Considerações do Capítulo

Nesta seção serão levantadas algumas considerações sobre escolhas que foram feitas durante a fase inicial do projeto como resultado das pesquisas preliminares do grupo.

3.5.1 Escolha do CMU Sphinx

Este trabalho foi bastante influenciado pelo projeto Gnome Voice Control que originalmente foi construído utilizando a *engine* de reconhecimento de fala CMU Sphinx da Carnegie Mellon.

O grupo iniciou por ele suas pesquisas sobre reconhecimento de fala, encontrou outros projetos como o Julius, o PerlBox, o SpeechControl, entre outros, e acabou por constatar que a Carnegie Mellon é um grande centro de referência no assunto, sendo o Sphinx inclusive utilizado em vários outros projetos relacionados.

Baseado nisso, a escolha não somente do Sphinx, mas também dos temas de algumas seções abordadas no capítulo de fundamentação foi uma consequência natural.

3.5.2 Escolha do ambiente Ubuntu

Como visto anteriormente tanto o Windows quanto o Mac OSX já possuem sistemas de reconhecimento de fala integrados, o que os tornam alvos desinteressantes para a tarefa, enquanto que no Linux não há um software oficial para isso.

Como os principais beneficiados por este projetos são pessoas com certos tipos de deficiências, a escolha do Ubuntu como distribuição também é uma escolha natural dada a sua filosofia de que "todo usuário de computador deve ser capaz de utilizar todo o software, independente de deficiências"[1]. (UBUNTU, 2012)

3.5.3 Modelagem Estruturada versus Modelagem Orientada a Objetos

A escolha da modelagem estruturada foi motivada por três razões principais, levando-se em conta aspectos técnicos e históricos, conforme detalhado a seguir.

Em primeiro lugar a *API* do Sphinx que será utilizada é disponibilizada em linguagem C (HUGGINS-DAINES, 2010), através de um conjunto de funções e não através de um conjunto de classes. Para trabalhar com orientação a objetos seria necessário criar uma camada a mais de software com um Adaptor (GAMMA *et al.*, 1994), cuja única finalidade seria fornecer uma API orientada a objetos para fazer o mesmo que a API original já faz.

O segundo motivo é que a última atualização no GVC foi um *patch* para que ele pudesse trabalhar com o Julius. Essa possibilidade de troca da *engine* motivou a escolha de uma abordagem de alta coesão e baixo acoplamento, encapsulando a *engine* de reconhecimento como um componente plugável. Isso evitará *patches* futuros, já que para utilizar uma nova *engine* bastará implementar um novo *plugin* para encapsulá-la.

No Linux, uma das formas mais comuns de se construir *plugins* carregáveis dinamicamente é através do uso da biblioteca dlopen, que conforme será detalhado durante a fase de

[1] Tradução do autor para "*Should be able to use all software regardless of disability.*"

desenvolvimento do projeto, é mais simples de ser implementado diretamente em linguagem C (ISOTTON, 1996).

Por último, o paradigma adotado tem que ser escolhido levando-se em conta os benefícios que ele trará ao projeto, e com base nos argumentos anteriores e no contexto deste trabalho, a orientação a objetos não traz vantagens significativas.

Nada impede, no entanto, que uma abordagem mista seja adotada para tirar proveito das classes utilitárias da STL, por exemplo.

3.5.4 Processos *versus Threads versus* Sequência

A arquitetura deste projeto prevê dois elementos distintos, cada um com um loop independente, conforme é explicitado tanto no diagrama de arquitetura quanto no fluxograma mostrados anteriormente. Esses elementos podem ser implementados como dois processos distintos ou como duas *threads* dentro do mesmo processo.

Uma abordagem alternativa deriva dos diagramas de fluxos de dados, onde o sistema apresenta-se como uma *pipeline*, tendo um fluxo de dados linear. Eles mostram que o reconhecedor e o executor são abstrações que não necessariamente precisam executar em paralelo.

Essa última observação não invalida a arquitetura apresentada, pois ela foi construída sobre abstrações que continuam valendo independente da escolha de implementação. O sistema continuará tendo um reconhecedor e um executor.

As escolhas de implementação são deixadas para a próxima fase do projeto, onde precisarão ser melhor analisadas questões sobre performance, simplicidade, manutenibilidade, entre outras.

4 CONSIDERAÇÕES

O objetivo deste capítulo é apresentar os resultados esperados, algumas das conclusões obtidas até o momento, bem como elencar algumas das discussões que são deixadas em aberto para serem melhor analisadas até a fase seguinte do projeto.

4.1. Resultados Esperados

Como resultados esperados ao final do projeto, em uma visão macro, espera-se ter disponível um sistema de reconhecimento de comandos por voz, que possa ser utilizado para manipular as principais aplicações de um *desktop* comum, ao menos através de suas funcionalidades mais conhecidas.

Para atingir este objetivo, é proposta uma aplicação a ser implementada conforme descrita no capítulo 3, visando atingir dois sub-objetivos bem definidos: reconhecer um comando de voz e enviar um comando para uma aplicação.

Além disso, espera-se que ela possua a capacidade de ser extendida de forma a conseguir reconhecer novos comandos conforme eles são implementados em suas respectivas aplicações, ou ainda, quando uma nova aplicação é adicionada ao *desktop*.

4.2. Obstáculos Previstos

Durante a fase inicial de pesquisas, foram identificados algumas áreas sensíveis que afetam diretamente o projeto, seja influenciando, inspirando ou ainda limitando o que será possível implementar.

4.2.1. Anomalias relativas ao sinal de áudio

Ao lidar com sistemas de reconhecimento de fala, a literatura aponta alguns problemas recorrentes, como por exemplo os relacionados às características do equipamento, à presença

de ruídos, às diferentes características acústicas dos ambientes onde eles são executados, ou ainda aqueles relacionadas à voz do locutor, ou locutores.

O idioma que será utilizado durante o reconhecimento também é um fator limitante importante, pois para cada idioma há modelos de linguagem e processos de treinamento distintos. Para adicionar um novo idioma, primeiramente será preciso realizar todo um conjunto de tarefas que por sua vez dependem do reconhecedor utilizado.

4.2.2 Envio de comandos para outras aplicações

Outra dificuldade encontrada foi a não existência, ao menos não documentada, de uma interface padrão para envio de comandos para as aplicações. Cada uma delas é implementada com funcionalidades e objetivos específicos, portanto, nem todos os comandos que podem ser enviados a uma aplicação, poderiam ser enviados a outra.

As aplicações de um *desktop* normalmente são planejadas para serem utilizadas em um ambiente bem definido, com interface gráfica, teclado e mouse. Por isso, dentro desse contexto, uma camada de comandos não é necessária.

4.3 Propostas

Este trabalho propõe soluções para alguns dos problemas encontrados, ao passo que reconhece que alguns deles podem ser melhor cobertos por outros projetos.

4.3.1 Reconhecimento de fala

Quanto aos problemas relacionados ao reconhecimento de fala, este projeto não tem a pretensão de resolvê-los, pois estão além do seu escopo. Ele parte do princípio que os problemas encontrados durante o reconhecimento de fala, são ou podem ser melhor resolvidos por trabalhos específicos nessa área.

Como a solução proposta também prevê que as *engines* de reconhecimento possam ser adicionadas ou removidas de forma transparente, este projeto limita-se a utilizar o CMU Sphinx, ao passo que outros trabalhos poderiam implementar *plugins* para outras *engines*.

Além disso, como o objetivo é comandar um *desktop* comum, também não serão utilizados microfones ou ambientes especiais, o que pode afetar os resultados do reconhecimento. De qualquer forma, este é um problema previsto e que faz parte da proposta fundamental do projeto, ser utilizado em um *desktop* em condições consideradas comuns.

4.3.2 Comandar aplicações

O envio de comandos para outras aplicações é um dos problemas resolvidos pelo sistema proposto. Para isso serão utilizadas as bibliotecas AT-SPI e WNCK, que também já foram utilizadas pelo Gnome Voice Control, projeto que serve referência para este sistema.

Elas permitem a implementação de um sistema de envio de comandos para as janelas das aplicações alvo, de forma que isso só funcionará em aplicações que possuem interface gráfica.

Se ao contrário do modo como são implementadas hoje, as aplicações possuíssem uma camada que expusesse uma interface com os seus comandos, tanto teclado e mouse, quanto um reconhecedor de voz poderiam acessá-la de forma transparente.

A proposta formal de uma nova arquitetura para aplicações desktop, que acomode essa camada, inclusive, pode ser o ponto de partida para trabalhos futuros.

REFERÊNCIAS

CHOU, W., JUANG B. ***Pattern Recognition in Speech and Language Processing***. New York: CRC Press, 2003

HUGGINS-DAINES, D. *PocketSphinx API Documentation*. Disponível em:

<http://cmusphinx.sourceforge.net/api/pocketsphinx/>. Acesso em: 6 maio 2012.

GAMMA, E. et al. ***Design Patterns****: Elements of Reusable Object-Oriented Software*. New Jersey: Addison-Wesley, 1994.

GETTYS, J. et. al. **Xlib - C *Language* X *Interface***: X *Window System Standard*. Massachusetts: X Consortium, 2002.

GNOME DEV CENTER. **GTK+ 3 *Reference Manual***. Disponível em: <http://developer.gnome.org/gtk3/stable>. Acesso em: 7 maio 2012.

_____. **Libwnck *Reference Manual***. Disponível em: <http://developer.gnome.org/libwnck/stable/>. Acesso em: 6 maio 2012.

_____. **AT-SPI C *Bindings Reference Manual***. Disponível em: <http://developer.gnome.org/at-spi-cspi/1.32/>. Acesso em: 6 maio 2012.

GNOME LIVE! **Gnome Voice Control**. Disponível em: <http://live.gnome.org/GnomeVoiceControl>. Acesso em: 6 maio 2012.

GRINSTEAD C. M., SNELL, J. L., ***Introduction to Probability***. 2nd ed. Rhode Island: American Mathematical Society, 1997.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. **IBGE e CORDE abrem encontro internacional de estatísticas sobre pessoas com deficiência.** Disponível em: <http://www. ibge.gov.br/home/presidencia/noticias/noticia_visualiza.php?id_noticia=438&id_pagina=1>. Acesso em: 9 abr. 2012.

ISOTTON, A. **C++ dlopen *mini howto***. Disponível em: <http://www.isotton.com/devel/docs/C++-dlopen-mini-HOWTO/C++-dlopen-mini-HOWTO.html>. Acesso em: 9 abr. 2012.

KUHN, R., MORI, R. A *Cache-Based Natural Language Model for Speech Recognition.* ***IEEE Transactions on Pattern Analysis and Machine Intelligence***. USA, v. 12, n. 6, jun. 1990.

LI, X., ***Combination and Generation of Parallel Feature Streams for Improved Speech Recognition***. 2005. Ph.D. Thesis - Carnegie Mellon University, Pittsburgh, 2005.

MCCORMACK, J., ASENTE, P., SWICK R. R. **X Toolkit Instrinsics**: C Language Interface. [S.l.]: X Consortium, 1994.

NEDEL, J. P. ***Duration Normalization for Robust Recognition of Spontaneos Speech via Missing Feature Methods***. 2004. Ph.D. Thesis - Carnegie Mellon University, Pittsburgh, 2004.

NOKIA CORPORATION. ***Online Reference Documentation***. 2011. Disponível em: <http://doc.qt.nokia.com/>. Acesso em: 7 maio 2012.

NYE, A. **X *Protocol Reference Manual***: for X 11 *version* 4, *release* 6. [S.l.]: O'reilly, 1995.

PARAVAN, A.**TCP/IP for MVS, VM, OS/2, and DOS X *Window System Guide*:** GG24-3911-01**.** 2nd ed. North Carolina: [s.n.], 1994.

PRESSMAN, Roger. ***Software Engineering: A Practitioner's Approach*.** 7th edition. New York: Mc Graw Hill, 2009.

RABINER, L., JUANG, B.H. ***Fundamentals of speech recognition***. New Jersey: Prentice Hall, 1993.

SCHEIFLER, R. **CLX**: *Common LISP X Interface*. Massachusetts: [s.n.], 1989.

_____. ***X Window System Protocol****: X Consortium Standard*. Massachusetts: [s.n.], 2004.

SCHEIFLER, R.W., GETTYS, J. *The X Window System, ACM Transactions on Graphics.* ***Massachusetts Institute of Tecnology***, USA, v. 5, n. 2, p.79-109, abr. 1986.

SLETZER, L. M. ***Microphone Array Processing for Robust Speech Recognition***. 2003. Ph.D. *Thesis* - Carnegie Mellon University, Pittsburgh, 2003.

SOMMERVILLE, Ian. ***Software Engineering***. 9th ed. Massachusetts: Addison-Wesley, 2010.

STAMP, M. A *Revealing Introduction to Hidden Markov Models.* **San Jose State University**, USA, p. 1-20, fev. 2012. Em: < http://www.cs.sjsu.edu/faculty/stamp/RUA/ HMM.pdf>. Acesso em: 3 maio 2012.

TOGNERI, R., PULLELLA, D. *An Overview of Speaker Identification: Accuracy and Robustness Issues.* ***IEEE Circuits and System Magazine***, USA, p. 23-61, abr./mai/jun 2011.

UBUNTU. ***Our philosophy***. Disponível em: <http://www.ubuntu.com/project/about-ubuntu/our-philosophy>. Acesso em: 9 abr. 2012.

_____. ***Meeting Minimum Hardware Requirements***. 2012. Disponível em: <http://help.ubuntu.com/12.04/installation-guide/powerpc/minimum-hardware-reqts.html>. Acesso em: 9 abr. 2012.

YNOGUTI, C. A. **Reconhecimento de Fala Contínua Usando Modelos Ocultos de Markov.** 1999. Tese (Doutorado) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 1999.