UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LUIZ FERNANDO TOSCAN

PROTÓTIPO DE UM ASSISTENTE PESSOAL DIGITAL INTELIGENTE

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2019

LUIZ FERNANDO TOSCAN

PROTÓTIPO DE UM ASSISTENTE PESSOAL DIGITAL INTELIGENTE

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Engenharia de Computação, do Departamento Acadêmico de Informática - DAINF da Universidade Tecnológica Federal do Paraná, *Campus* Pato Branco, como requisito parcial para obtenção do título de "Engenheiro de Computação".

Orientadora: Profa. Dra. Kathya Silvia Collazos Linares

PATO BRANCO
2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

TERMO DE APROVAÇÃO

Às 14 horas do dia 15 de julho de 2019, na sala V106, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, reuniu-se a banca examinadora composta pelos professores Kathya Silvia Collazos Linares (orientadora), Beatriz Terezinha Borsoi e Luciene de Oliveira Marin para avaliar o trabalho de conclusão de curso com o título **Protótipo de um assistente pessoal digital inteligente**, do aluno **Luiz Fernando Toscan**, matrícula 01171895, do curso de Engenharia de Computação. Após a apresentação o aluno foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Profa. Kathya Silvia Collazos Linares
Orientador (UTFPR)

_____
Profa. Beatriz Terezinha Borsoi
(UTFPR)

_____
Profa. Luciene de Oliveira Marin
(UTFPR)

_____
Profa. Beatriz Terezinha Borsoi
Coordenador de TCC

_____
Prof. Pablo Gauterio Cavalcanti
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

Nas últimas décadas a tecnologia, em constante desenvolvimento, tem feito esforços para oferecer diversos dispositivos que contribuem a melhorar a qualidade de vida das pessoas, dentre estes há os assistentes pessoais digitais que foram desenvolvidos para auxiliar a organizar e realizar diversas tarefas do dia-a-dia. Nos últimos anos houve uma popularização no desenvolvimento de assistentes pessoais digitais controlados por voz, principalmente em *smartphones* mais modernos, resultado do avanço tecnológico e pesquisas na integração e combinação de técnicas como reconhecimento de voz, processamento de linguagem natural e utilização de inteligência artificial. Exemplos de assistentes digitais comandados por voz e que já fazem parte do cotidiano das pessoas, são: a Siri® da empresa Apple, Cortana® da empresa Microsoft e o Google Assistant®, da empresa Google. O protótipo desenvolvido utiliza sistemas prontos para a interface voz/texto e linguagem AIML para o processamento da linguagem natural. E, ainda, fornece assistência pessoal básica comandada por voz ao usuário, em âmbito residencial.

Palavras-chave: AIML. Assistente Virtual. Processamento de Linguagem Natural. *Chatterbots*.

ABSTRACT

In recent decades the technology, in constant development, has made efforts to put out several devices that contribute to improve the quality of life for the people, among them are the personal virtual assistants that were developed to help organize and perform a large number of tasks of the everyday rush. In recent years there has been a growth in the development of personal virtual assistants controlled by voice, mainly in modern smartphones, as a result of technological advances and research in the integration and combination of techniques such as voice recognition, natural language processing and the use of artificial intelligence. Examples of voice-activated virtual assistants that are already part of everyday life are: Siri® from Apple, Cortana® from Microsoft, and Google Assistant® from Google. The developed prototype uses systems ready for the voice / text interface and the AIML language for the processing of natural language. This prototype should give basic personal assistance commanded by voice to the user in a residential scope.

Keywords: AIML. Virtual assistant. Natural language processing. Chatterbots.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

A.L.I.C.E.	*Artificial Linguistic Internet Computer Entity* / Entidade Computadorizada de Linguagem Artificial para Internet
AIML	*Artificial Intelligence Mark-up Language*
API	*Application Programming Interface* / Interface de Programação de Aplicativos
ASR	*Automatic Speech Recognition*
BSD	*Berkeley Software Distribution*
CLR	*Common Language Runtime*
HMM	*Hidden Markov Model* / Modelo Oculto de Markov
IA	Inteligência Artificial
PLN	Processamento de Linguagem Natural
RAV	Reconhecimento Automático de Voz
RDF	*Resource Description Framework*
SO	Sistema Operacional
TTS	*Text-to-Speech*
XML	*eXtensible Markup Language*
WAV	*Waveform Audio File*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo serão apresentadas as motivações do trabalho assim como os objetivos e a justificativa.

1.1 CONSIDERAÇÕES INICIAIS

Nas últimas décadas o tempo tem se tornado cada vez mais uma variável importante na vida das pessoas. A tecnologia, em constante desenvolvimento, possui ferramentas e/ou programas computacionais que podem auxiliar na economia e na gestão do tempo, facilitando a vida de quem a utiliza. Os assistentes pessoais digitais são desenvolvidos para auxiliar a organizar e realizar diversas tarefas do dia-a-dia, nos últimos anos houve uma popularização no desenvolvimento de assistentes pessoais digitais controlados por voz, principalmente em *smartphones* mais modernos, resultado do avanço tecnológico e pesquisas na integração e combinação de técnicas como reconhecimento de voz, processamento de linguagem natural e utilização de inteligência artificial (MILHORAT, 2014).

A ideia básica de um assistente pessoal, de forma geral, é a de uma pessoa (ou agente) que seja capaz de fornecer ajuda distinta para determinados momentos e contextos de atuação, com a importante característica de adaptabilidade às diversas exigências, preferências pessoais e rotinas de seu dirigente (MILHORAT, 2014). Pode-se então inferir que quando usuários procuram por eses assistentes, eles têm expectativas de que apresentem simplicidade, flexibilidade e facilidade de interação, isto é, que se comportem de forma próxima aos seres humanos, sendo a forma de interação mais fácil, ou por assim dizer, a mais natural, por voz. Essa forma de interação não requer esforços cognitivos, nem recursos de atenção e/ou memória por parte do usuário para com o assistente.

Exemplos de assistentes digitais comandados por voz que já fazem parte do cotidiano das pessoas são: a Siri® da empresa Apple, Cortana® da empresa Microsoft, e o Google Assistant®, da empresa Google. Eles utilizam sistemas voltados para uso geral em atividades simples.

Para Preece et al. (2013, p.192), tecnicamente, “as interfaces de fala atingiram a maturidade, sendo muito mais sofisticadas e precisas do que a primeira geração de sistemas de fala no início de 1990, que ganhou a reputação de muitas vezes não ouvir o que uma pessoa tinha dito”.

Nesse contexto, o presente trabalho direciona-se para o desenvolvimento de um assistente pessoal digital comandado por voz, que permita gerar relatórios, por exemplo, de listas de estoque para: supermercado, farmácia, papelaria; que emita lembretes para atividades agendadas ou não como: reuniões, aniversários, clima, entre outros. O protótipo é de uso pessoal visando à customização do assistente virtual.

1.2 OBJETIVOS

Os objetivos dividem-se em geral e específicos, os últimos mostram os passos a serem seguidos durante a elaboração do protótipo.

1.2.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um protótipo para dar assistência pessoal básica em âmbito residencial com interface comandada por voz utilizando a linguagem *Artificial Intelligence Mark-up Language* (AIML).

1.2.2 Objetivos Específicos

Determinar as funcionalidades do protótipo;
Construir a base de conhecimentos para a área de aplicação;
Desenvolver o protótipo utilizando a linguagem AIML.

1.3 JUSTIFICATIVA

O conforto e a qualidade de vida é uma busca incessante do ser humano, assim como, a eficiência de sistemas com o objetivo de atender essas necessidades. Durante o dia diversas atividades devem ser realizadas e elas por sua vez estão associadas com dados de origens distintas: organizar horários para atividades laborais e domésticas, administrar, para os membros da família, as atividades e as necessidades relacionadas à alimentação, às saúde e à educação; atender grupos aos quais se pertence como: associações de bairro, associações comunitárias, etc. Lembrar essas atividades é o objetivo das agendas eletrônicas, entretanto elas devem ser programadas. Eventos inesperados ou de última hora, muitas vezes não são

programados e acabam sendo relegados à lembrança automática da pessoa, que se força a manter-se alerta, distraindo sua atenção às atividades realizadas, gerando preocupações adicionais.

Agendas eletrônicas, mesmo as comandadas por voz, não fazem parte da domótica. A ideia básica da domótica é interagir com os habitantes da residência e aprender dinamicamente com seus comportamentos (SGARBI, 2007). Essa interação é o que a agenda eletrônica precisaria possuir para tornar-se um assistente pessoal digital comandado por voz. Esse assistente forneceria suporte nas atividades das pessoas e poderia aprimorar-se apreendendo de forma gradual as necessidades e as atividades do seu usuário.

O assistente pessoal digital comandado por voz pode fazer parte das aplicações inteligentes em ambientes domésticos. Segundo Katsamanis (2014, p.5547), essas aplicações são recentes e são projetadas para oferecer novas formas de garantir segurança, consciência, conforto e controle do ambiente na vida domiciliar.

De outro lado, a interface de voz e a tecnologia de reconhecimento de fala fornecem liberdade para as pessoas trabalharem em ambientes ativos, permitindo usá-los sem precisar ficar segurando ou programando ativamente os dispositivos (LEE; GRICE, p.308). Assim, o desenvolvimento de um assistente pessoal digital inteligente pode contribuir para o conforto e a qualidade de vida das pessoas ao se manterem atentas às atividades a serem por elas desenvolvidas.

1.4 ORGANIZAÇÃO DO TRABALHO

O primeiro capítulo contém a introdução, a justificativa, o objetivo geral e os objetivos específicos do trabalho. O segundo capítulo aborda o referencial teórico na qual se baseia a construção do protótipo. No terceiro capítulo descrevem-se os materiais e os procedimentos realizados para desenvolver o protótipo e o quarto capítulo apresenta as conclusões e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo aborda os tópicos dos quais se baseia o trabalho. Inicia pelo Processamento de Linguagem Natural, que representa o âmago deste trabalho, seguido pelos tópicos Reconhecimento de Voz e *Chatterbot*, dois elementos importantes que serão combinados. O capítulo finaliza com a descrição dos conceitos principais da linguagem AIML, principal componente na construção do assistente pessoal digital inteligente desenvolvido como resultado da realização deste trabalho.

2.1 PROCESSAMENTO DE LINGUAGEM NATURAL

O Processamento de Linguagem Natural (PLN) é um ramo específico da Inteligência Artificial (IA) e apresenta uma tarefa complexa que envolve conhecimento em diferentes disciplinas: linguísticas, semiótica, ciência da computação, psicologia cognitiva dentre outras (NEVES, 2005, p.26).

A linguagem é uma capacidade humana que permite ao ser humano comunicar-se de forma confiável por meio de mensagens, na forma falada, escrita ou gesticulada (sinais), além de por meio de representações gráficas. Quando Alan Turing propôs o teste para determinar se uma máquina exibe comportamento inteligente[1], utilizou a comunicação em linguagem natural e forma textual. A proposta foi ponto de partida para o interesse de tornar a máquina um interlocutor ativo como um ser humano o seria (NORVIG; RUSSEL, 2013).

O PLN é uma área de pesquisa e aplicação da inteligência artificial que explora como os computadores podem ser utilizados para a compreensão e manipulação da linguagem natural em texto ou fala para uma determinada utilidade. As pesquisas nesta área objetivam modelar como os seres humanos entendem e usam a linguagem natural para que possam ser desenvolvidas ferramentas e técnicas apropriadas que permitam tornar os sistemas computacionais capazes de compreender e manipular a linguagem natural para poderem realizar as tarefas especificadas.

Os problemas a serem considerados na compreensão da linguagem natural são: o processo do pensamento; a representação e o significado da entrada linguística e o conhecimento do mundo. Isto é, pode-se iniciar no nível da palavra para determinar sua natureza e estrutura morfológica; logo, trabalha-se no nível da frase para determinar a ordem

das palavras, a gramática e o significado da frase inteira e finalmente verifica-se o contexto e o ambiente global ou domínio. Para a compreensão da linguagem natural é necessário distinguir entre os seguintes níveis interdependentes que as pessoas usam para extrair significados da linguagem escrita ou falada:

- O nível fonético ou fonológico que trata da pronúncia;
- O nível morfológico que trata das pequenas partes das palavras que levam significado, prefixos e sufixos;
- O nível léxico que trata do significado léxico das palavras;
- O nível sintático que trata da gramática e da estrutura das frases;
- O nível semântico que trata do significado das palavras e das frases;
- O nível de discurso que trata com a estrutura de diferentes tipos de texto que utilizam estruturas de documentos e;
- O nível pragmático que trata do conhecimento que vem do mundo externo, isto é, fora do conteúdo do documento.

Um sistema de PLN pode envolver todos ou alguns desses níveis de análise tornando-o um sistema altamente complexo (CHOWDHURY, 2003).

Para Bird et al. (2009, p.22) aplicações que utilizam processamento de linguagem natural possuem as qualidades de habilidade para trabalhar com linguagem e o potencial para poupar esforço humano na automatização, além de possuírem a característica fundamental da programação de possibilitar que máquinas tomem decisões baseadas no comportamento do ser humano e executem certas instruções quando atender a determinadas condições ou repetidamente em *looping* até atingir uma condição esperada. Esses autores também ressaltam que "um importante objetivo da pesquisa PLN tem sido o de fazer avançar a difícil tarefa de construir tecnologias que 'entendem a linguagem', utilizando técnicas superficiais, mas poderosas em vez de capacidades de conhecimento e raciocínio irrestrito".

Existe uma longa tradição em linguística e na filosofia da linguagem que visualiza a linguagem natural essencialmente como uma linguagem declarativa de representação do conhecimento e que tenta estabelecer sua semântica formal, porém, a visão moderna sobre o assunto é a de que ela serve como um meio de comunicação em vez de pura representação. Assim, o significado de uma sentença qualquer pode depender também do contexto em que

[1] Teste de Turing: Experimento para determinar se uma máquina possuía inteligência: um juiz humano conversa em linguagem natural com um ser humano e uma máquina, os participantes estão separados uns dos outros. Se o juiz humano não distinguir o ser humano da máquina, considerava-se que a máquina passou no teste.

ela foi formulada. Finalmente, as linguagens naturais sofrem de ambiguidade, o que causaria dificuldades para o desenvolvimento do pensamento (NORVIG; RUSSEL, 2013).

Bird et al. (2009, p. 33) também ressaltam que “um importante objetivo da pesquisa PLN tem sido o de fazer avançar a difícil tarefa de construir tecnologias que ‘entendem a linguagem’, utilizando técnicas genéricas, mas poderosas, em vez de capacidades de conhecimento e raciocínio irrestrito”.

2.2 RECONHECIMENTO DE VOZ

A fala para comunicação entre os seres humanos é algo tão natural que sua utilização na interação do ser humano com computadores pode facilitar e agilizar na realização de atividades corriqueiras e na aceitação de pessoas que não possuem certas habilidades para interagir por meio de teclados ou telas sensíveis ao toque. Porém, obviamente, o processo de reconhecimento de voz por dispositivos computacionais está sujeito a diversos problemas. Paula (2013) escreve sobre a complexidade envolvida no fenômeno da fala no ser humano:

> O trato vocal e os articuladores são órgãos biológicos com propriedades não lineares, cuja operação não está apenas sob o controle consciente, mas também é afetado por fatores que vão desde a educação ao estado emocional do indivíduo. Como resultado, as vocalizações podem variar muito em termos de seu sotaque, pronúncia, articulação, aspereza, nasalidade, altura, volume e velocidade. Além disso, durante a transmissão, os padrões de fala irregulares podem ainda ser distorcidos pelo ruído de fundo e por ecos, bem como características elétricas (se telefones ou outros equipamentos eletrônicos são usados) (PAULA, 2013, p. 17).

Um sistema de Reconhecimento Automático de Voz (RAV), também conhecido por *Automatic Speech Recognition* (ASR), converte o sinal de voz em texto. Em sistemas RAV o sinal de voz é capturado por um microfone e é, então, analisado. A análise ocorre com a separação em quadros do sinal, que é convertido para uma sequência de vetores de observação finitos que são comparados com um grupo de modelos estocásticos acústicos e linguístico, produzindo uma lista encadeada de hipóteses para o conteúdo do texto reconhecido, associados a um valor de confiança que indica o grau de proximidade com as palavras possíveis. Apenas as transcrições que atingem o limiar definido são então aceitas pelo sistema (MILHORAT, p. 460, 2014).

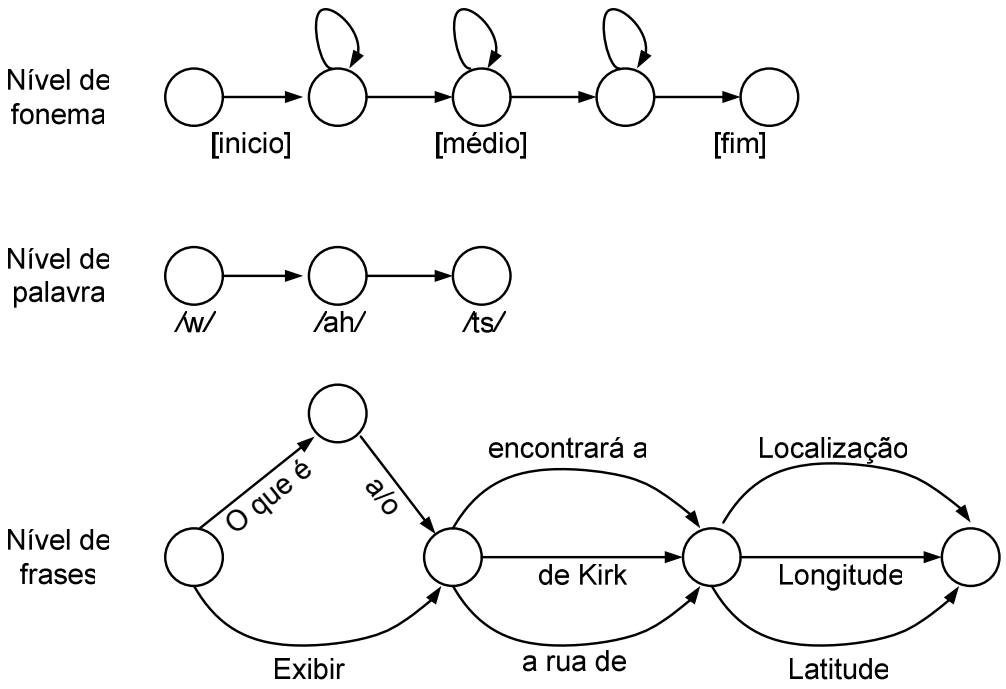
Oliveira et al. (2011) define que um sistema típico de ASR adota uma abordagem estatística baseada em Modelos Ocultos de Markov (*Hidden Markov Model*s - HMM), que

são largamente usados no reconhecimento automático da voz e do locutor porque manipulam muito bem os aspectos estatísticos e sequências do sinal de voz.

Um Modelo Oculto de Markov consiste em uma máquina de estados finita que modifica seu estado a cada unidade de tempo. Sua estrutura consiste de um modelo matemático formado por uma cadeia de estados conectados entre si, cada transição entre os estados possui uma probabilidade de ocorrência, sendo que cada estado está vinculado a um processo estocástico, que pode ser discreto ou contínuo, conhecido como processo de observação de saída (SILVA, 2010, p. 14).

Segundo Paula (2013, p. 25), um HMM possui essa denominação pelo fato de "poder ser pensado como uma caixa preta, em que a sequência de símbolos de saída gerada ao longo do tempo é observável, mas a sequência de estados visitados ao longo do tempo está escondida da vista". E que na aplicação de HMM para reconhecimento de fala, os estados são interpretados como modelos acústicos, indicando que os sons são suscetíveis de serem ouvidos durante os seus segmentos correspondentes da fala. A Figura 1 ilustra como as transições e estados em um HMM podem ser estruturados de forma hierárquica, com o fim de representarem fonemas, palavras e frases.

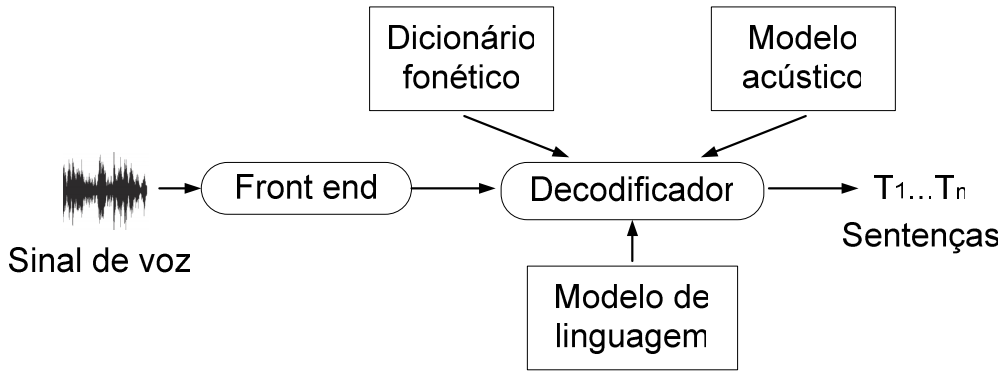
Figura 1 - Um HMM estruturado hierarquicamente



Fonte: Paula (2013, p. 26).

Além disso, um ASR é composto por cinco blocos principais: *front-end*, dicionário fonético, modelo acústico, modelo de linguagem e decodificador ou reconhecedor, conforme ilustrado na Figura 2.

Figura 2 - Os principais blocos que contêm um sistema ASR



Fonte: Oliveira et al. (2011).

2.2.1 *Front-End*

O início do processamento da fala é a conversão analógico-digital que decodifica a onda analógica em dados digitais. Durante a conversão, o sistema faz o uso de filtros, para filtrar o som digitalizado com a intenção de remover ruídos indesejáveis na parametrização do sinal da fala (processo de filtragem). Esta primeira fase do processo de reconhecimento de fala é conhecido como *front-end.* Devido à natureza aleatória do sinal de voz, utilizá-lo na sua forma original não é uma prática recomendável (LOUZADA, 2010).

2.2.2 Dicionário Fonético

A conversão de uma sequência de caracteres em sequência de fones é um importante pré-requisito para serviços que envolvem reconhecimento e/ou síntese de voz. Contudo, a tarefa não é trivial e diversas técnicas de conversão vêm sendo adotadas ao longo da última década. Existe um número bem menor de estudos na área dedicados à língua português-brasileiro quando comparado à língua inglesa, por exemplo. O dicionário consiste, basicamente, de um conjunto de palavras com suas respectivas transcrições fonéticas (SILVA, 2010, p. 10).

2.2.3 Modelo Acústico

O modelo acústico gera um modelo matemático, a partir das características (*features*) extraídas da fala, que representa um sinal original. Isso ocorre de tal forma que os segmentos da fala que são desconhecidos possam ser mapeados de modo correto e que seja determinada a palavra correspondente. No caso de sistemas de reconhecimento de fala contínua o modelo matemático é determinado por fonemas e não por palavras isoladas (LOUZADA, 2010).

2.2.4 Modelo de Linguagem

Utilizar somente informações acústicas não é suficiente para o bom desempenho de um sistema RAV, já que assim, uma palavra poderia ser seguida por qualquer outra, o que dificulta muito o trabalho do decodificador. Existem duas formas de definir o que pode ser reconhecido: gramáticas livres de contexto e modelos de linguagem (SILVA, 2010, p. 18).

2.2.5 Decodificador

Um dos módulos finais de um sistema ASR é o decodificador, que tem a função de gerar a transcrição de amostras de fala desconhecidas para sua forma textual. O processo de transcrição desempenhado pelo decodificador ocorre com a utilização dos módulos apresentados nas Seções anteriores. O processo de reconhecimento é controlado por uma rede de palavras construída a partir do modelo de linguagem. A rede consiste em um conjunto de palavras (nós) conectadas por arcos, onde cada arco possui uma probabilidade de ocorrência (transição). Com base no dicionário fonético e um conjunto de modelos HMMs estimados, consegue-se determinar, dada uma fala desconhecida, a probabilidade de qualquer um dos caminhos que a rede pode percorrer. O decodificador tem como objetivo encontrar os caminhos que são mais prováveis (LOUZADA, 2010, p. 23). Ainda sobre sistemas ASR, eles podem ser caracterizados e avaliados por vários parâmetros, como os apresentados no Quadro 1.

Quadro 1 - Parâmetros típicos que caracterizam um sistema ASR

Parâmetros	Descrição
Vocabulário	A dificuldade de um reconhecedor de fala é diretamente proporcional ao aumento do vocabulário utilizado ou de palavras semelhantes. É comum

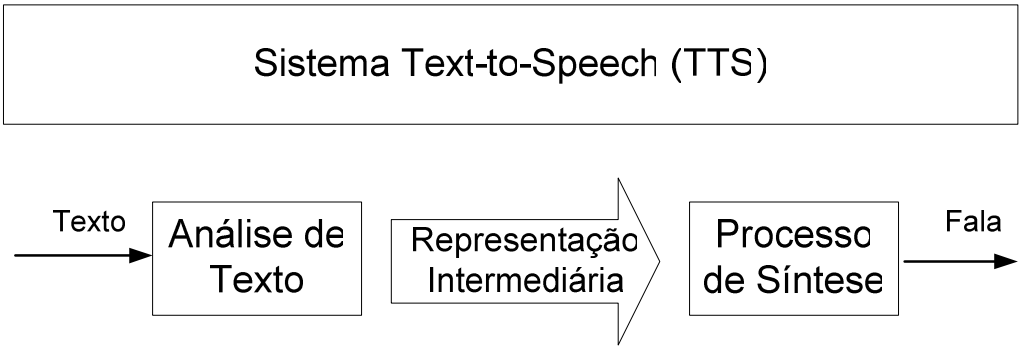
	classificar como pequenos os sistemas que reconhecem menos de 20 a 50 palavras e sistema que reconhecem mais de 20.000 palavras, como grandes (BRAGA, 2006, p. 18).
Dependência do Locutor	**Dependente do Locutor**: quando o sistema reconhece apenas a fala dos locutores para os quais ele foi treinado.
	Independente do Locutor: quando o sistema reconhece a fala de qualquer locutor, sem a necessidade uma fase de treinamento com o locutor em questão. Isso é possível pois o sistema foi previamente treinado com vários locutores.
Modo de Fala	**Palavras Isoladas**: são os sistemas mais simples de serem desenvolvidos, sendo necessário haver pequenas pausas entre as palavras de uma locução.
	Fala Contínua: são os mais complexos e difíceis de serem implementados, pois devem ser capazes de lidar com todas as características e vícios da forma natural de falar, dentre os quais podem ser citados: durações de palavras desconhecidas, efeitos de coarticulação e pronúncia descuidada.
Perplexidade	Um conceito bastante aplicado para calcular a dificuldade de reconhecer uma palavra é a perplexidade. Pode-se defini-la como a média do número de palavras que pode seguir uma palavra depois que o modelo de linguagem foi aplicado.
Relação sinal-ruído	Problemas que podem impactar no desempenho do sistema, como: ruídos, ambiente, distorção de acústica, diferentes microfones e outros.

Fonte: Adaptado de Louzada (2007, p. 23) e Braga (2006).

Text-to-Speech (TTS) é um sistema texto-voz para conversão de texto em voz. É uma técnica comumente usada para saída de voz, como um *feedback* de fala para os usuários, uma conversação. TTS é implementado inteiramente em software e capacidade padrão de áudio é requerida. Este sistema possui várias aplicações, incluindo leitura de mensagens eletrônicas e geração de *prompts* de voz para aplicações de voz (LEE; GRICE, 2006, p. 311).

Neto, Batista e Klautau (2012, p. 306) apresentam na Figura 3 um diagrama funcional simples de um sistema TTS típico composto por duas partes: um *Front End* e um *Back End*.

Figura 3 - Diagrama funcional de um sistema TTS



Fonte: Neto, Batista e Klautau (2012, p. 306).

Lee e Grice (2006, p. 311) ainda apresentam os diferentes tipos de métodos utilizados que são fundamentais em interfaces que utilizam reconhecimento e geração de voz, como, por exemplo, modos dependentes ou independentes do locutor, modos isolados ou contínuos.

2.3 INTERFACE VOZ-TEXTO

Existem diversos reconhecedores de fala disponíveis no mercado em praticamente todos os idiomas. Entretanto, eles são, em geral, demasiadamente caros e de código fechado, o que dificulta a adaptação do programa para aplicação específica.

Por este motivo, para o presente trabalho foi selecionado o decodificador Julius, que, além de ser gratuito, possui código aberto.

2.3.1 Julius

O Julius é um decodificador de alto desempenho para reconhecimento de voz. Inicialmente projetado para reconhecimento de voz em Japonês, tem sido bastante utilizado por pesquisadores e desenvolvedores de software em todos os idiomas (SILVA, 2010, p. 28).

O Julius é um programa decodificador com código aberto usado para reconhecimento de fala contínua. Desenvolvido para os sistemas operacionais Windows e Linux, possui um grande vocabulário, alto desempenho e versatilidade. Com ele, pode-se executar o reconhecimento de voz tanto em tempo real por meio de dispositivos de entrada, como o microfone, quanto de arquivos de áudio previamente armazenados nos formatos *Waveform Audio File* (WAV), ou arquivo de áudio de forma de onda, ou RAW (do inglês, cru; significando arquivos que possuem as informações exatamente como foram captadas pelo microfone) (PERICO; SHINOHARA; SARMENTO, 2014).

Perico, Shinohara e Sarmento (2014, p. 53) explicam que o funcionamento do Julius é bastante simples: “ao ser executado, ele busca a entrada de áudio indicada e o arquivo de configuração com extensão *.jconf*, sendo apontados quais devem ser os modelos acústicos e de linguagem a serem utilizados. Além disso, o Dicionário Fonético também é carregado pelo programa”.

2.3.2 Coruja

O Coruja é uma *Application Programming Interface* (API) ou Interface de Programação de Aplicativos desenvolvida na linguagem de programação C++ que possibilita o controle em tempo real do decodificador de reconhecimento de voz Julius e de um Modelo Acústico. Resumidamente, a API permite que aplicações utilizem os serviços do software sem precisar se envolver com detalhes de implementação (linguagem de baixo nível de programação) (PERICO; SHINOHARA; SARMENTO, 2014).

Ese sistema ASR foi desenvolvido para o reconhecimento da língua português-brasileiro e é distribuído sob a licença *Berkeley Software Distribution* (BSD). O Coruja oferece modelos acústicos e de linguagem, além de uma API própria construída para facilitar a tarefa de controlar o decodificador Julius. Essa API visa flexibilidade quanto à linguagem de programação, sendo compatível com a especificação *Common Language Runtime* (CLR) (OLIVEIRA et al., 2011).

2.3.3 *Cloud Speech-to-Text*

Diferentemente das alternativas apresentadas nas seções anteriores, desde março de 2016, a empresa Google disponibiliza para desenvolvedores uma API para reconhecimento *online* de voz. Com o *Cloud Speech-to-Text* o áudio gravado é enviado ao servidor da empresa e a conversão do áudio em texto é realizada pela aplicação de modelos de redes neurais avançados e tecnologia de aprendizado de máquina (GOOGLE CLOUD, 2019).

2.4 *CHATTERBOT*: ROBÔ DE CONVERSAÇÃO

Alan Turing, pioneiro na inteligência artificial e na ciência da computação, na década de 50, em seu famoso ensaio “Computing Machinery and Intelligence”, propôs uma abordagem diferente para o conceito de: “máquinas pensantes”. Para o autor, o questionamento não deveria ser se máquinas podem pensar, mas sim, se essas máquinas podem passar por um teste de inteligência comportamental, dando origem ao então famoso teste de Turing. O teste de Turing consiste em fazer um programa desenvolver uma conversação com um interrogador por 5 minutos. O interrogador deve adivinhar se teve a

conversação com um programa ou uma pessoa. Se não fosse possível distinguir o ser humano da máquina, então o sistema poderia ser considerado inteligente (NORVIG; RUSSEL, 2013).

Foi a partir desse desafio que foi iniciado o desenvolvimento dos sistemas simuladores de diálogos que, atualmente, possuem aplicabilidade em diversas áreas. Apontado como um dos sistemas pioneiros da área, o ELIZA simula o diálogo do usuário como se ele fosse o paciente de um psicólogo virtual. Neste programa, que data de 1966, as técnicas de PLN são bastante simplistas, mas servem para ilustrar uma das primeiras aplicações que, ainda hoje, é bastante popular nas aplicações do tipo *chatterbot* (VIEIRA; LOPES, 2010, p. 185)/

Norvig e Russel (2004, p. 916) também destacam que alguns *chatterbots* conseguiram passar no teste e enganar os interrogadores por 5 minutos, como por exemplo, o programa ELIZA e o A.L.I.C.E.. Porém, nenhum deles chegou perto do critério de 30% do tempo contra julgadores treinados e o campo da IA como um todo dedicou pouca atenção aos testes de Turing.

O questionamento proposto por Alan Turing está inserido num contexto de fundamentos filosóficos que divide a IA em duas hipóteses, sendo a asserção de que as máquinas talvez pudessem agir de maneira inteligente (ou, quem sabe, agir como se fossem inteligentes) é chamada hipótese de IA fraca pelos filósofos, e a asserção de que as máquinas que o fazem estão realmente pensando (em vez de simularem o pensamento) é chamada hipótese de IA forte (NORVIG; RUSSEL, 2013, p. 915). O PLN e, consequentemente, *chatterbots*, estão inseridos no campo de pesquisa da IA fraca, pois buscam reproduzir os processos de desenvolvimento que resultaram no funcionamento normal da língua.

Os *chatterbots* são definidos por Galvão (2003, p. 18) como sistemas que utilizam linguagem natural para dialogar com o usuário e, atualmente, consideram-se alternativas capazes de desempenhar o papel de facilitadores em diversas aplicações como, por exemplo, suporte ao consumidor, educação à distância e propósito geral. Além disso, são sistemas que têm a capacidade de explorar o comportamento social do usuário perante o computador, mesmo quando não são programados com essa intenção.

Rabello e Marchi e Fossatti (2011) reforçam ainda, que “esses sistemas foram elaborados com o intuito de tornar a interação entre o homem e os computadores mais familiar, dando a impressão de que o sistema possui personalidade própria”.

Neves (2005) propõe uma classificação em três gerações de acordo com a forma como esses sistemas são implementados e a evolução da utilização de tecnologias. A

caracterização das três gerações é apresentada no Quadro 2, bem como o *chatterbot* pioneiro de cada geração.

Quadro 2 - As três gerações dos *chatterbots*

Gerações	Característica	*Chatterbot*
Primeira Geração	Buscam palavras-chaves e utilizam o casamento de padrões e regras gramaticais para dar continuidade ao dialogo. O casamento de padrões é utilizado para verificar se a frase digitada pelo usuário possui a estrutura desejada, para então encontrar uma estrutura relevante, pontos de alinhamento e substituir a parte do casamento por outra estrutura.	ELIZA
Segunda Geração	Utilizam de princípios de inteligência artificial, como redes neurais, para simular a conversa. Apesar da sofisticação destes *chatterbots*, eles não tiveram melhor atuação que os da primeira geração.	JULIA
Terceira Geração	Possuem uma apresentação mais atrativa, com uma interface gráfica que estimula o diálogo. Utilizam da linguagem de marcação AIML.	A.L.I.C.E.

Fonte: Adaptado de Comarella e Café (2008).

2.4.1 ELIZA

O ELIZA é um programa de computador capaz de conversar em linguagem natural com seres humanos que mantém o *status* de ser um dos sistemas de IA mais populares no mundo, com versões em diferentes idiomas e implementações para quase todas as plataformas existentes (*mainframes*, computadores pessoais, consoles de jogos etc.) (NEVES, 2005, p. 28).

Além disso, segundo descrito por Leitão (2004, p. 3), é um programa para psicanálise baseado no princípio psicanalítico Rogeriano, o qual consiste em repetir as frases do paciente, conseguindo sua introspecção sem envolvimento de opiniões do psicanalista. O autor ainda ressalta que a personalidade desenvolvida por ELIZA é uma das mais bem definidas entre os robôs de conversação, sendo bastante compreensiva, porém sem a capacidade de lembrar o que foi dito durante a conversa.

2.4.2 A.L.I.C.E.

Artificial Linguistic Internet Computer Entity, ou Entidade Computadorizada de Linguagem Artificial para Internet *(*A.L.I.C.E.) é um *chatterbot* que marca o início da terceira

geração, conhecido por *Alicebot* também se baseia em mecanismos de detecção de padrões em conjunto com uma base de conhecimento na linguagem AIML para elaborar as suas respostas. Apesar de o A.L.I.C.E. ser bastante semelhante ao ELIZA, o uso de arquivos estruturados de acordo com a norma proposta por Dr. Richard S. Wallace permitiu o surgimento de várias variantes desse agente de conversação e o desenvolvimento de bases de dados de conhecimento em várias línguas e sobre diversos temas específicos. A.L.I.C.E possui sua base disponibilizada (BRANDÃO, 2012, p. 23).

2.4.3 Ultra Hal

Os *chatterbots* Ultra Hal podem, entre outras coisas, auxiliar no aprendizado de determinados assuntos, que são recuperados de bancos de dados disponíveis na Internet. Pode, também, servir de interface de comandos em linguagem natural para um sistema operacional (CASTANHO, 2002, p. 25).

O Ultra Hal pode ser visto como uma espécie de assistente, possuindo uma memória interna que guarda números de telefone, endereços, e-mails, apontamentos, aniversários ou qualquer outra informação. Ele pode automaticamente discar números de telefone, abrir e-mails e lembrar aos usuários de datas de aniversários (MOURA, 2003, p. 22).

2.5 ARTIFICIAL INTELLIGENCE MARK-UP LANGUAGE

Desenvolvida durante os anos de 1995 e 2000 por Richard S. Wallace e a comunidade A.L.I.C.E. AIML é uma linguagem de marcação utilizada para o desenvolvimento de robôs de conversação. Atualmente, é uma linguagem baseada na em *eXtensible Markup Language* (XML) pois foi projetada visando a padronização da gramática (WALLACE, 2014).

Por ser baseada em XML, a linguagem AIML é descrita por marcações denominadas de *tags* que compõem pares de entrada-saída. Quanto mais bem planejada a base AIML, maior será o desempenho de coerência na conversação (MACHADO, 2005).

Baseando-se na documentação sobre a linguagem AIML, Leonhardt et al. (2003, p. 57) a definem como sendo uma linguagem de fácil aprendizagem e utilização, que “apresenta um conjunto de *tags* e comandos simples para implementação da base de conhecimento de um c*hatterbot* e serve para analisar as mensagens enviadas pelo usuário e decidir a forma como essas mensagens devem ser respondidas”.

As principais *tags* que compõem a estrutura da linguagem AIML, segundo Wallace (2014), são apresentadas no Quadro 3.

Quadro 3 - Principais *tags* da Linguagem AIML

Tag	**Descrição**
<*aiml*>	Inicia e termina um bloco programado em AIML;
<*category*>	Identifica uma "unidade de conhecimento" na base de conhecimento; Uma categoria se divide em *pattern* e *template*.
<*pattern*>	Identifica um padrão de mensagem simples frequentemente utilizado por usuários; É o padrão que uma entrada do sistema irá buscar.
<*template*>	Contém a resposta para uma mensagem do usuário;

Fonte: Wallace (2014).

O AIML descreve uma classe de objetos conhecidos como objetos AIML e parcialmente o comportamento dos *chatterbots* que processam a linguagem utilizando um interpretador. Os objetos AIML são compostos de por duas unidades básicas: as *tags* tópico (<*topic*>) e categoria (<*category*>) (WALLACE, 2014).

Cada categoria possui um padrão de entrada (<*pattern*>), que será comparado com a sentença digitada pelo usuário pelo interpretador AIML e um padrão de resposta (<*template*>*)*, que será usado para montar uma sentença a ser retomada ao usuário (GALVÃO, 2003). O Quadro 4 apresenta a estrutura fundamental de uma categoria básica escrita em AIML.

Quadro 4 - Base de um categoria em AIML

<*category*>
<*pattern*> PADRÃO DE ENTRADA </*pattern*>
<*template*> Modelo(s) para construção de respostas </*template*>
</*category*>

Fonte: Adaptado de Moura (2003, p.26)

Um exemplo de uso dessa marcação e o diálogo que é gerado são apresentados no Quadro 5.

Quadro 5 - Exemplo de uso da *tag* Categoria

Exemplo	**Diálogo**
<*category*> <*pattern*> OI * </*pattern*> <*template*> Oi, tudo bem? </*template*>	**Usuário**: Oi *chatbot*. **BOT**: Oi, tudo bem?

</*category*>	

Fonte: Adaptado de Leonhardt et al. (2003).

No exemplo apresentado no Quadro 5, o padrão de entrada "Oi", seguido de qualquer informação, terá como resposta ou possível resposta à sentença "Oi. Tudo bem?". O caractere * representa um conjunto de caracteres que pode ou não existir na sentença de entrada, esse conjunto também é chamado de coringa e seu valor pode ser utilizado na *tag* <*template*>, por meio do uso da *tag* <*star*>.

Leonhardt (2005) explica, ainda, que com a utilização de AIML, é possível definir mais de uma resposta para um único padrão e especificar critérios de escolha de cada uma das respostas. Além disso, as mais de 20 outras *tags* que compõem a linguagem AIML são responsáveis por fornecer a desenvoltura necessária para o *chatterbot* propor uma solução à mensagem enviada.

Alguns dos elementos mais utilizados da linguagem AIML estão definidos no Quadro 6.

Quadro 6 - Exemplo das *tags* da Linguagem AIML mais utilizadas

Tag	**Descrição/Exemplo**	**Uso**
srai	Utilizadas na redução simbólica de entradas, chama outras categorias recursivamente; Como exemplo de uso, se reconhecido o padrão "HI THERE" como entrada, o *template* buscará pela categoria "OI" e retornará a sua respectiva resposta, economizando código, pois a resposta pode ser a mesma.	<*category*> <*pattern*>HI THERE</*pattern*> <*template*><***srai***>OI</***srai***></*template*> </*category*>
random	A *tag random* é utilizado em conjunto com uma lista de valores para retornar randomicamente um valor, dando a ideia de maior flexibilidade nas respostas. No exemplo de uso ao lado, se nenhum padrão para a entrada for reconhecido, ela é tratada como um coringa e o robô escolherá uma das três respostas possíveis para retornar ao usuário.	<*category*> <*pattern*>*</*pattern*> <*template*> <***random***> <*li*>O que foi isso?</*li*> <*li*>Eu não entendi.</*li*> <*li*>Você pode repetir mas de um jeito diferente?</*li*> </***random***> </*template*> </*category*>
that	É um elemento filho e opcional da categoria, é utilizado para estabelecer contexto para o padrão. O robô irá lembrar-se da última sentença que foi dita por meio do valor da etiqueta <*that*>. **Interlocutor**: Eu gosto de chá. **Robô**: Você gosta de chá quente ou gelado? **Interlocutor**: Quente.	<*category*> <*pattern*>EU GOSTO DE CHÁ</*pattern*> <*template*>Você gosta de chá quente ou gelado?</*template*> </*category*> <*category*> <*pattern*>QUENTE</*pattern*>

	Robô: Eu também.	<***that***>VOCÊ GOSTA DE CHÁ QUENTE OU GELADO</***that***> <*template*>Eu também.</*template*> </*category*> <*category*> <*pattern*>GELADO</*pattern*> <***that***>VOCÊ GOSTA DE CHÁ QUENTE OU GELADO</***that***> <*template*>Sério? Gelado parece tão sem graça.</*template*> </*category*>
topic	O elemento *topic* é um predicado que permite contextualizar categorias por mais de uma interação, diferentemente da tag *that*. Basicamente, sempre que um tópico for setado, o robô começa a buscar por padrões dentro daquele tópico definido. A seguir, é possível verificar um exemplo de diálogo contextualizando a conversa utilizando o elemento *topic:* **Interlocutor**: Vamos falar de café. **Robô**: Eu gosto de café. **Interlocutor**: Eu bebo o meu puro. **Robô**: Eu prefiro o meu com leite. **Interlocutor**: Vamos falar de chá. **Robô**: Eu gosto de chá. **Interlocutor**: Eu bebo o meu puro. **Robô**: Eu prefiro o meu com açúcar.	<*category*> <*pattern*>VAMOS FALAR DE * </*pattern*> <*template*> OK, eu gosto de <*set name*="topic"> <*star* /></*set*> </*template*> </*category*> <***topic*** *name*="café"> <*category*> <*pattern*>EU BEBO O MEU PURO </*pattern*> <*template*>Eu prefiro o meu com leite</*template*> </*category*> </***topic***> <***topic*** *name*="chá"> <*category*> <*pattern*> EU BEBO O MEU PURO </*pattern*> <*template*>Eu prefiro o meu com açúcar</*template*> </*category*> </***topic***>
think	Utilizada para setar predicados e em situações em que o robô deve processar a entrada do usuário, sem retornar o resultado desse processamento. Um exemplo de diálogo é proposto a seguir: **Interlocutor**: Meu nome é Luiz **Robô**: Eu vou lembrar do seu nome **Interlocutor**: Você sabe meu nome? **Robô**: Seu nome é Luiz	<*category*> <*pattern*>MEU NOME É *</*pattern*> <*template*> <***think***><*set name*="name"> <*star* /> </*set*></***think***> Eu vou lembrar do seu nome. </template> </category>
set	O elemento *set* exerce diferentes funções dependendo de qual dos blocos da categoria, *pattern* ou *template*, o código esta: No *pattern*: significa que irá buscar em uma coleção de *strings* o valor correspondente. Por exemplo, uma coleção que contenha todos os	No pattern: <*category*> <*pattern*> <***set***>cores</***set***> É UMA COR? </*pattern*> <*template*>Sim, <*star* /> é uma cor. </*template*>

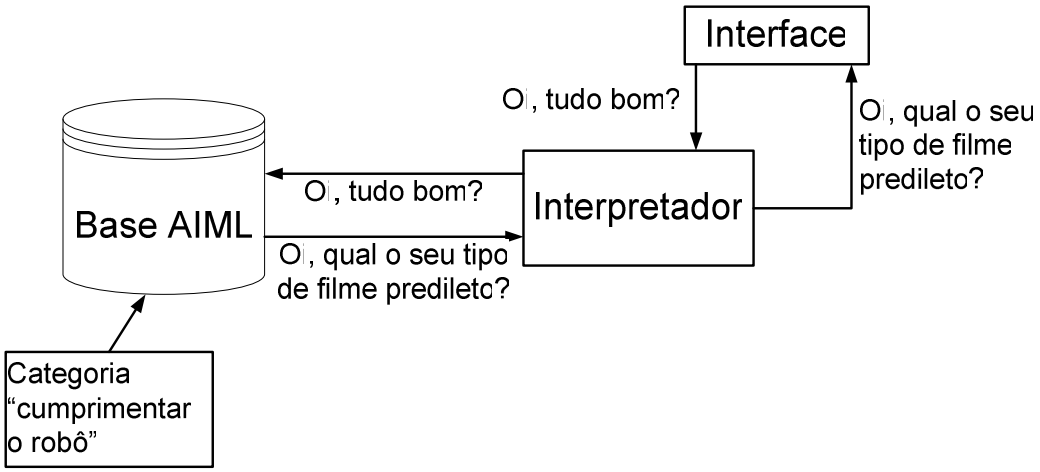
	nomes de cores armazenados, se o padrão de entrada fizer parte do *set*, a resposta retornada será afirmativa. Caso contrário, o padrão não será reconhecido, logo, a resposta será negativa. No *template*: seta predicados e variáveis para serem utilizados em outros momentos e retornados por meio da *tag get*. Um diálogo de exemplo pode ser observado a seguir: **Interlocutor**: Meu nome é Luiz **Robô**: Prazer em te conhecer, Luiz **Interlocutor**: Qual é o meu nome? **Robô**: Seu nome é Luiz	</*category*> <*category*> <*pattern*>* É UMA COR?</*pattern*> <*template*>Não, <star /> não é uma cor.</*template*> </*category*> No template: <*category*> <*pattern*>MEU NOME É *</*pattern*> <*template*> Prazer em te conhecer, ***<set name="username">***<*star* />.***</set>*** </*template*> </*category*> <*category*> <*pattern*>QUAL É O MEU NOME? </*pattern*> <*template*>Seu nome é <*get name*="username" />.</*template*> </*category*>
star	É utilizada para repercutir partes da entrada do usuário que são representadas em formato de coringa. Um diálogo possível está exemplificado a seguir: **Interlocutor**: Minha cor favorita é verde. **Robô**: A sua cor favorita é verde.	<*category*> <*pattern*>MINHA * FAVORITA É * </*pattern*> <*template*>A sua ***<star />*** favorita é ***<star index="2" />***</*template*> </*category*>

Fonte: Adaptado de Pandorabots (2019).

Galvão (2003) comenta que apesar de serem agentes, os *chatterbots* utilizam uma arquitetura na qual a sequência de raciocínio (perceber-raciocinar-agir) encontra-se integrada, sendo totalmente definida por meio de categorias. O interpretador AIML recebe a sentença do usuário (dados perceptivos) e identifica a categoria correspondente a ela (raciocínio). Uma vez selecionada a categoria, o interpretador retorna o seu *template* para o usuário (ação) (GALVÃO, 2003).

O interpretador AIML é um módulo fundamental para o funcionamento do *chatterbot*. A Figura 4 apresenta um exemplo de funcionamento de um interpretador AIML. O interpretador recebe como entrada as sentenças digitadas pelos usuários e então busca uma categoria na base AIML cujo padrão de entrada seja o mais adequado para a sentença atual. Em seguida, o interpretador monta a resposta que será devolvida ao usuário, verificando antes se existem *tags* a serem processadas no *template* de saída dessa categoria. Por fim, o interpretador envia à interface a sentença montada a partir do *template* de saída (MOURA, 2003).

Figura 4 - Exemplo de funcionamento de um interpretador AIML



Fonte: Moura (2003, p. 65).

Machado (2005, p. 23) ressalta que "quanto mais bem planejada a base AIML, maior será o desempenho de coerência na conversação". Além disso, também relata o modo como a padronização da linguagem contribui para o desenvolvimento dessa linha de pesquisa:

> Devido à característica de padronização da linguagem AIML, proporcionada pelo XML, existem hoje diversas tecnologias de código aberto responsáveis pelo processamento das bases AIML. Isto incentiva a comunidade a desenvolver mais e mais aplicações e ferramentas que se beneficiem da linguagem AIML, tornando o padrão cada vez mais sedimentado e confiável. (MACHADO, 2005, p. 24)

Oliveria (2010), no entanto, aponta que apesar de ser uma linguagem bastante utilizada para construção de *chatterbots*, o AIML ainda apresenta falhas na contextualização da conversação.

3 O PROTÓTIPO DE UM ASSISTENTE PESSOAL DIGITAL INTELIGENTE

Neste capítulo descrevem-se os materiais e passos executados para a construção do protótipo, bem como o resultado obtido dessa execução.

3.1 ESCOPO DO PROTÓTIPO

O protótipo do assistente pessoal digital inteligente desenvolvido deve receber como entrada do usuário uma frase dita por ele. A gravação é feita pelo pacote de áudio nativo do sistema operacional Raspbian e é iniciado ao ser pressionado o botão conectado ao pino 16 da placa. Foi definido um período fixo de oito segundos de gravação. Após gerar o arquivo do tipo .wav, ele é repassado para o *Cloud Speech Recognition* que retorna a sentença em formato de texto, servindo de entrada para o interpretador AIML que julgará a melhor saída, podendo ser em texto simples, ou utilizar as aplicações externas para produzir a resposta esperada.

As entradas esperadas serão todas em linguagem natural, claras, objetivas e que obedeçam ao tempo máximo de oito segundos no momento da gravação.

Diferentemente do inglês, na língua portuguesa, uma pergunta sem o uso do ponto de interrogação pode ser facilmente interpretada como uma afirmação. Considerando isso, foram definidas expressões de início de sentença para denotar perguntas que não são explicitas e o protótipo conseguir diferenciá-las corretamente, visto que a transcrição dos áudios não envolve o uso de pontuação. O Quadro 7 apresenta três tipos definidos.

Quadro 7 – Desambiguação para o início de perguntas

Início de pergunta	Comentário
Será que *	Utilizadas, principalmente, para previsão do tempo, consultas ao calendário e listas.
Você sabe se *	
Você acha que *	

Fonte: autoria própria.

3.2 BASE DE CONHECIMENTO

Além de exercer as funcionalidades propostas, o protótipo ainda é um robô de conversação e deve manter algumas características importantes e para que isso seja possível, é utilizada a base padrão oficial em inglês do *chatterbot* A.L.I.C.E, por ser uma base AIML importante e muito usada para aprendizado ou desenvolvimento de outros *chatterbots*. Para a utilização pelo protótipo, os textos são traduzidos.

Os arquivos serviram de base para a criação de novas categorias, principalmente as especificas que são responsáveis por preparar os dados que são utilizados nas aplicações que oferecem assistência ao usuário. Quatro novos arquivos foram adicionados à base padrão: “listas.aiml”, “previsãodotempo.aiml” e “agenda.aiml”. Todos os arquivos AIML são carregados para a memória pelo interpretador, eles são considerados o cérebro do robô.

3.3 HARDWARE

A utilização de minicomputadores para o desenvolvimento de projetos que envolvam hardware se mostram extremamente eficientes e cada vez mais atrativos, uma vez que as possibilidades atreladas a esses dispositivos só tem crescido, enquanto seu custo faz o caminho inverso e torna-se relativamente cada vez mais acessível.

No caso do protótipo proposto, foi utilizado o minicomputador *Raspberry Pi* 3 - *Model B* apresentado na Figura 5.

Figura 5 - Raspberry Pi 3 – Model B

Fonte: The MagPi Magazine (2016).

O Raspberry Pi (RPi) é um computador do tamanho de um cartão de crédito e a versão 3 do modelo B está presente no mercado desde 2015. O Quadro 8 apresenta as principais especificações do hardware, o destaque desse modelo está no fato de possuir Bluetooth e Wireless integrados à placa.

Quadro 8 - Especificações do Raspberry Pi 3 B

	Raspberry Pi 3 - Model B
Processador	Broadcom BCM2837
CPU	ARM Cortex-A53 com quatro núcleos e 1.2GHz
RAM	1GB LPDDR2 @ 900 Mhz
Armazenamento Flash	Entrada cartão SD
Rede	10/100M Ethernet e 2.4GHz 802.11n Wireless
Bluetooth	Bluetooth 4.1 Classic
Portas	Saída HDMI, conector analógico de áudio e vídeo 3.5mm, quatro entradas USB 2.0, Ethernet, *Camera Serial Interface* (CSI), *Display Serial Interface* (DSI)
Periféricos (GPIO)	12 –GPIO, USART, SPI, I2C (P1 and P5), CSI (*camera serial interface*) e DSI (*display serial interface*),
Alimentação	USB 5V 322ma @ idle. Rated at 700mA
Suporte à Linux	Distribuições ARM Linux que suportam ARMv6 Dois dos principais OS suportados: Debian e Arch Linux ARM

Fonte: Adaptado de The MagPi Magazine (2016).

Para que a interface seja comandada por voz, a captação de áudio é feita por meio de um microfone USB acoplado a uma das quatro portas disponíveis. Além disso, os pinos de uso geral também precisaram ser utilizados para a implementação de um botão simples que inicia a gravação do áudio. Apesar da API utilizada para converter áudio em texto permitir *stream* e reconhecimento contínuo, além das conversões terem um limite de apenas 60 minutos sem que taxas sejam aplicadas, para que a complexidade não fosse elevada, a conversação, que precisa ser em turnos, está limitada a áudios fixos de 8 segundos por botão pressionado.

3.4 FERRAMENTAS UTILIZADAS

Para desenvolvimento do protótipo, o Raspberry Pi foi preparado para executar o Sistema Operacional (SO) Raspbian, que é uma versão baseada em Debian e otimizada para o hardware embarcado da placa e disponibilizada de forma gratuita para desenvolvedores de

aplicações utilizando o Raspberry Pi. O SO é considerado o principal da família de microcomputadores Pi (RASPBIAN, 2019).

A linguagem de programação definida para implementação foi Python, uma vez que o Raspbian já conta com a linguagem previamente instalada e, como de projetos e tutoriais voltados ao aprendizado que utilizam a linguagem para programar aplicações na placa sugerem, é uma linguagem com muitos recursos e de fácil entendimento.

3.4.1 Program Y: O Interpretador AIML

A peça chave para a construção de *chatterbots* que implementam sua base de conhecimento com a linguagem AIML é a utilização de um interpretador. No desenvolvimento em Python, um interpretador que tem se destacado é o Program Y, que nos últimos anos foi preparado para interpretar todas as *tags* da versão 2.0 do AIML. Além de acrescentar novas possibilidades, como a utilização de *sets* e *maps* dinâmicos, suporte ao padrão de formatação de dados *Resource Description Framework* (RDF) muito utilizado em web semântica e de ser altamente extensível ao possibilitar facilmente a inclusão de novos módulos e APIs ao projeto (STERLING, 2019).

A adaptação de qualquer *chatterbot* com base AIML para serem compatíveis com o Program Y depende basicamente da configuração correta dos arquivos de configuração (config.yaml) e de chamada dos *scripts* (sestra.sh). No arquivo de chamada apenas define-se qual a pasta raiz do *bot*, qual o módulo cliente principal (nesse caso o próprio console) e qual o arquivo de configuração será utilizado. O arquivo do protótipo ficou definido conforme apresentado no Quadro 9.

Quadro 9 – Exemplo do script

```
#! /bin/sh
clear
export BOT_ROOT='/home/pi/mybots/tcc'
python3 -m programy.clients.events.sestra.client
–-config $BOT_ROOT/config/xnix/config.yaml
--cformat yaml
--logging $BOT_ROOT/config/xnix/logging.yaml
--bot_root $BOT_ROOT
```

Fonte: autoria própria.

O arquivo de configuração está dividido em três partes: *Client, Bot* e *Brain*. Cada seção define diferentes aspectos do *chatterbot*. Em *Client* estão as configurações com a localização de todos os arquivos que compõem o *chatterbot* AIML (categorias, *sets*, *maps*, predicados...) e de como o interpretador será executado. Para esse caso, a opção definida foi *console*, pois o cliente será executado diretamente pelo Shell do Raspbian.

Em *Bot* é configurado o andamento e o comportamento do *chatterbot*, independente da base de conhecimento. Para o protótipo foi mantido o padrão. Em *Brain* estão as propriedades que definem como o cérebro irá interpretar, processar as entradas e apresentar as saídas. É nesta seção em que as variáveis, coleções e mapas dinâmicos precisam ser setados. No caso de extensões, o caminho precisa ser setado no próprio arquivo AIML, como parâmetro da *tag* <*extension*>.

Um exemplo de como os padrões produzidos pelo interpretador e que serão passados para as outras aplicações pode ser observado no Quadro 10.

Quadro 10 - Exemplo do uso da etiqueta *extension*

```
<category>
   <pattern> INICIAR LISTA * </ pattern>
   <template> <srai> INICIAR_LISTA <extension path="programy.extensions.lists.Listas">
                <star />
                </extension>
                 </srai>
    </template>
</category>

<!---Retorno da extensão Listas ---!>
<category>
   <pattern> INICIAR_LISTA * </pattern>
   <template> A Lista <star /> foi criada. </template>
</category>
```

Fonte: autoria própria.

Em linhas gerais, é criada uma *string* que será tratada pelo módulo Listas (programy.extensions.lists.Listas), executando o que for preciso e retornando uma resposta para ser repassada ao usuário.

3.4.2 Outras Aplicações

Uma parte essencial do trabalho proposto está na realização de tarefas específicas que dependem de aplicações externas para servirem ao usuário. O Quadro 11 apresenta todas as APIs utilizadas para tornar em realidade as funcionalidades propostas.

Quadro 11 - Outras aplicações utilizadas

API/Ferramenta	Comentário	Referência
Wikipedia	Biblioteca Python que torna fácil a busca e obtenção de informações disponibilizadas na Wikipedia.	https://pypi.org/project/wikipedia/
Yahoo Weather	Pacote Python para conseguir informacões de clima e tempo para qualquer localização, bem como previsões do tempo e condições do vento, atmosfera e astronomia.	https://pypi.org/project/weather-api/
Cloud Speech Recognition	API que faz a conversão de áudio em texto.	https://cloud.google.com/speech-to-text/
Calendar	API para agendar os eventos que forem sendo solicitados pelo usuário através do assistente.	https://developers.google.com/calendar/
Yagmail	Cliente GMAIL para o envio de e-mails ao usuário como forma de lembretes ou troca de informações mais completas.	https://pypi.org/project/yagmail/
PyDrive	Pacote Python que facilita a utilização da API do Google Drive para armazenamento de arquivos em nuvem.	https://pypi.org/project/PyDrive3/

Fonte: autoria própria.

3.5 FUNCIONALIDADES DO PROTÓTIPO

Para a realização deste trabalho foram definidas algumas atividades cotidianas que precisam ser desempenhadas pelo protótipo, a fim de prover auxilio ao usuário, sendo elas as apresentadas no Quadro 12.

Quadro 12 - Funcionalidades do protótipo

Funcionalidade	Descrição
Agendar compromissos	Através da integração com um calendário, o usuário deve ser capaz de agendar compromissos para que seja lembrada posterior mente pela interface.
Escrever listas	Escrever e salvar listas ditadas pelo usuário, como listas de compras para o supermercado, papelaria, etc.
Escrever notas	Escrever e salvar pequenas notas que sejam ditadas pelo usuário.
Fornecer informações gerais	Fornecer assistência sobre informações gerais que podem ser obtidas de forma imediata com uma simples pesquisa em um site buscador, bem como informações sobre a previsão do tempo e etc.
Lembretes ou alarmes	Ser capaz de lembrar o usuário de tarefas previamente programadas pelo próprio usuário, além de pequenos lembretes gerais, como por exemplo, a data de aniversário de alguém importante para o usuário, reuniões.

Fonte: autoria própria.

3.5.1 Predicados

O protótipo possui uma série de predicados que correspondem ao conjunto de informações e conhecimento que o *bot* pode adquirir durante as conversas. São basicamente variáveis que ficam armazenadas em um arquivo texto e são carregadas sempre que o *bot* for iniciado, mantendo assim o conhecimento através do tempo. Também é uma forma de declaração de variáveis que são utilizadas pelas categorias AIML. Um exemplo de como o arquivo de predicados está organizado pode ser observado no Quadro 13.

Quadro 13 – Exemplo de organização dos predicados

```
name:Luiz Fernando
firstname:Luiz
middlename:Fernando
lastname:Toscan
fullname:Luiz Fernando Toscan
job:perguntar
birthday:perguntar
birthdate:perguntar
sign:perguntar
lembrete:perguntar
```

Fonte: autoria própria.

A estrutura padrão de armazenamento é “nome:valor”, sendo o nome do predicado separado de seu valor padrão correspondente pelo caractere “:”. Os casos de valores padrão “perguntar” podem indicar duas situações que foram definidas no projeto do protótipo: informações que ainda não foram adquiridas pelo *chatterbot* e que serão perguntadas quando surgir a oportunidade, sem a possibilidade de alteração automática pelo robô, ou valores que mudam durante uma conversação, mas que sempre precisam ser perguntados novamente pois funcionam puramente como variáveis de funções específicas. Para o segundo caso, o arquivo físico nem chega a ser alterado, tudo é realizado apenas em memória e tempo de execução. Por exemplo, o valor do predicado “lembrete” mudará apenas durante o processo de recuperação de um lembrete para o usuário ou na inserção de um novo.

O arquivo de predicados está definido na seção *Client* do arquivo de configuração do protótipo conforme o Quadro 14, sendo *properties_storage* os predicados específicos do *bot* e *defaults* os predicados referentes ao usuário e todo o sistema.

Quadro 14 - Definição dos arquivos de predicados

```
properties_storage:
        file: $BOT_ROOT/storage/properties/properties.txt
defaults_storage:
        file: $BOT_ROOT/storage/properties/defaults.txt
```

Fonte: autoria própria.

3.5.2 Rotina Padrão

O protótipo foi pensado para ser um dispositivo que fique em algum lugar da residência, sempre em funcionamento e esperando por uma entrada do usuário. Todos os dias pela manhã ele deve emitir relatórios sobre as informações que o *chatterbot* possui agendadas para o dia e/ou semana e enviar para o e-mail do usuário, assim como informações úteis sobre feriados e previsão do tempo. O Quadro 15 apresenta um exemplo de relatório criado pelo *bot*.

Quadro 15 - Exemplo de relatório

```
Olá Luiz, estas são as atividades que você possui programado:
Para hoje (10 de Junho de 2019):
   - Dentista às 14:30
   - Mercado
Para a semana (08 de Junho de 2019 à 14 de Junho de 2019):
   - Reunião às 15:30 na Quinta-feira dia 11 de Junho de 2019
Pela previsão parece que hoje teremos um dia nublado e com
probabilidade de chuva. É bom levar o guarda-chuva.
```

Fonte: autoria própria.

Além dos relatórios diários fixos, em toda a primeira conversa do dia que o *bot* iniciar com o usuário, todos os compromissos agendados para o dia devem ser informados, assim como a previsão do tempo. O protótipo sempre aguardará por uma saudação para iniciar a conversação e adaptará a resposta de forma aleatória para que as respostas não se tornem engessadas.

3.5.3 Listas e Notas

Fazer listas e armazenar pequenas notas é uma das principais funcionalidades do protótipo. Listas tendem a ser um bom fator para a organização e controle, as listas de tarefas, por exemplo, servem como exercício para uma melhor execução e aproveitamento do tempo devido ao planejamento envolvido. Listas de mercado e de compras, por sua vez, ajudam a controlar o estoque e até mesmo as finanças da pessoa que as utiliza.

Definiram-se quatro tipos específicos de listas que o protótipo deve tratar: Mercado, Farmácia, Padaria e Compras em geral. Na criação e manipulação dessas listas, estabelecem-se quatro informações mandatórias que o protótipo precisa coletar para realizar suas funções. As informações padrão de uma lista mostram-se no Quadro 16, alguns valores são opcionais, além disso, o *bot* pode deduzir algumas no momento da entrada, dependendo da categoria correspondente.

Quadro 16 - Informações padrão de uma lista

Informação	Predicado	Descrição
Título	lista-titulo	Um nome simples para a lista. No caso de listas específicas, o título será o seu tipo, por exemplo, "Lista do Mercado".
Item	lista-item	Predicado que contém o(s) item(ns) da lista.
Quando	lista-quando	O usuário será perguntado o dia em que a lista será utilizada. A entrada pode ser informada em forma de dias da semana ou data.
Hora	lista-hora	Indica a hora e/ou o momento do dia (Manhã, Tarde, Noite, Dia Todo).

Fonte: autoria própria.

Uma série de comandos foi criada que podem gerar essas listas e ao iniciar essa rotina, o *chatterbot* não responderá sobre outros assuntos até a lista ser fechada. O usuário deve informar uma data possível de quando a lista será utilizada, para que seja mantido um controle. O Quadro 17 apresenta as principais categorias para a criação de manipulação de listas.

Quadro 17 - Categorias padrão para criação e manipulação de listas

Categoria	Descrição
Faça uma lista *	Inicia o processo, qualquer assunto fora do tópico será ignorado até o

	usuário fechar a lista.
Feche a lista *	Encerra o processo de criação de uma lista.
Adicione * a lista *	Inclui o item na lista indicada pelo segundo coringa, verificando se o item já faz parte da coleção correspondente. Em caso negativo, pede para o usuário uma confirmação.
Remova * da lista *	Remove um item da lista.
Leia a lista *	Apenas retorna a lista pedida pelo usuário.
Será que preciso ir *	Comando que verifica se o usuário tem alguma lista pendente que ainda não foi utilizada.
Está faltando *	Ao identificar essa categoria, o *bot* irá perguntar para o usuário se o item informado precisa ser incluído em uma lista e prosseguir apropriadamente.

Fonte: autoria própria.

O *bot* fará listas gerais (para tarefas, por exemplo) e listas específicas (mercado, farmácia, papelaria e compras). O importante das listas específicas é que os itens possuem coleções separadas para que o *chatterbot* trate de forma mais eficiente as particularidades associadas a cada lista, além evitar a inserção errada de itens. As coleções específicas definidas são: “colecao-mercado.txt”, “colecao-farmacia.txt” e “colecao-papelaria.txt”.

No caso da inclusão de um novo item desconhecido, o *bot* solicitará confirmação do usuário e então incluirá o novo item também à coleção. Por exemplo, para “Lista do mercado”, todos os itens que forem sendo adicionados à lista de mercado irão incrementar a coleção “colecao-mercado”. Esse processo visa abreviar a identificação das categorias no processo de criação e manipulação de listas.

São consideradas notas, pequenas porções de informação que o *bot* deve armazenar e para estes casos, o protótipo irá armazená-las na forma de lista de tarefas. Os comandos para que as notas sejam produzidas estão apresentados no Quadro 18. A palavra “nota” também pode ser substituída por “anotação”.

Quadro 18 - Categorias padrão para criação e manipulação de notas

Categoria	Comentário
Faça uma nota	Inicia o processo de criação de uma anotação.
Inclua uma nota	
Leia as notas *	Lê para o usuário as notas salvas.

Fonte: autoria própria.

3.5.4 Agenda e Lembretes

É por meio da API *Calendar* que o protótipo tem acesso a toda a rotina do usuário, assim como feriados e datas especiais, como aniversários, que foram cadastradas. A partir disso, cabe ao protótipo monitorar e lembrar ao usuário os eventos que compõe sua agenda.

Para que o protótipo não se torne uma ferramenta desagradável, ele deve tentar entender a rotina do usuário, para isso, foram definidos quatro valores para saber o grau de importância dos lembretes que ele precisa retornar ao usuário no futuro.

No Quadro 19 estão definidas as prioridades dos avisos.

Quadro 19 - Prioridades dos lembretes

Prioridade	Descrição
Indiferente	Não é levado em consideração
Baixa	É lembrado apenas no relatório diário ou quando solicitado pelo usuário.
Média	Além de aparecer no relatório diário, será lembrado durante a primeira conversa do dia e/ou quando solicitado pelo usuário o lembrete do evento. São lembretes casuais e que não sugerem tanto compromisso. Esses eventos podem ser cadastrados diretamente pelo comando "ME LEMBRE..."
Alta	Aparece no relatório diário, é lembrado durante a primeira conversa do dia e algum período antes do evento. Esses eventos podem ser cadastrados diretamente pelo início "NÃO POSSO ESQUECER...." e "É IMPORTANTE..." Por padrão, os avisos começam a ser enviados a partir de 1 hora antes do início do evento.

Fonte: autoria própria.

3.5.4.1 Agenda

Foram definidos cinco predicados que caracterizam um evento da agenda. O Quadro 20 descreve e apresenta as informações padrão esperadas.

Quadro 20 - Informações padrão de um evento da agenda

Informação	Predicado	Descrição
Título	agenda-titulo	Um nome para o evento. Por exemplo: "Reunião da empresa", "Aniversário".
Descrição	agenda-desc	Uma descrição sucinta para o evento.
Quando	agenda-quando	A data do evento. A entrada pode ser informada em forma de dias da semana ou data.

Hora	agenda-hora	Indica a hora e/ou o momento do dia (Manhã, Tarde, Noite, Dia Todo).
Local	agenda-local	Opcional, é o local onde o evento irá ocorrer.

Fonte: autoria própria.

3.5.4.1 Lembretes

No Quadro 21 estão definidas quatro informações que o *bot* precisa coletar para cadastrar novos lembretes.

Quadro 21 - Informações padrão de um evento de lembrete

Informação	Predicado	Descrição
Descrição	lembrete	Predicado que contém o texto que precisa ser lembrado.
Quando	lembrete-quando	O usuário será perguntado o dia em que a lista será utilizada. A entrada pode ser informada na forma de dias da semana ou data.
Hora	lembrete-hora	Indica a hora e/ou o momento do dia (Manhã, Tarde, Noite, Dia Todo).
Prioridade	lembrete-prioridade	Define o grau de importância do lembrete.

Fonte: autoria própria.

O usuário pode cadastrar lembretes pelos comandos definidos no Quadro 22. A parte do comando "Me lembre" ainda pode ser substituído pelas alternativas: "Me ajude a lembrar", "Me avise", "Não posso esquecer de" e "É importante lembrar que".

Quadro 22 - Categorias padrão para criação e manipulação de lembretes

Categoria	Comentário
Me lembre que *	O texto capturado no coringa será o corpo do lembrete, o *bot* ainda deve perguntar quando e qual a prioridade do aviso.
Me lembre às * que *	Para os dois casos o primeiro coringa refere-se às horas ou período do dia em que o aviso precisa ser emitido. Por exemplo, "Me lembre pela manhã que preciso tomar remédio".
Me lembre pela * que *	
Me lembre * que *	Cria um lembrete na data e hora que foi captada pelo *bot*. O primeiro coringa da categoria irá corresponder à: dia da semana, data ou número de dias para o aviso.
Me lembre hoje que *	Será criado um lembrete com a data atual na agenda do usuário.
Me lembre de * em * que *	Também é possível cadastrar eventos periódios. Os dois primeiros coringas irão tratar a entrada para definir o período. Mais informações podem ser solicitadas pelo *bot*.

Fonte: autoria própria.

3.5.5 Informações Gerais

O *chatterbot* deve fornecer informações breves sobre os mais diferentes assuntos. Por meio do suporte à RDF no formato de triplas (sujeito-objeto-propriedade), é possível que o *bot* responda a perguntas básicas de “O que é?”, “Pra que serve?". Além do suporte a RDF, pequenos resumos podem ser obtidos no site Wikipedia, referente aos mais diversos assuntos. O Quadro 23 apresenta as categorias padrão para busca de informações gerais.

Quadro 23 - Categorias padrão para busca de informações gerais

Categoria	Descrição
Busque na Wikipedia por *	Explicitamente irá buscar no site Wikipedia pelo termo capturado pelo coringa e retornar um breve sumário sobre o assunto. Quando o termo possuir mais de uma página, o bot deve solicitar ao usuário e tratar a desambiguação.
O que é *	Para este tipo de perguntas o *bot* primeiro busca um coringa correspondende na base RDF e, em caso negativo, tanta buscar na Wikipedia pelo termo.
Pra que serve *	Busca exclusivamente nos arquivos RDF.

Fonte: autoria própria.

Outro tipo de informação que o *bot* fornecerá são informações sobre a previsão do tempo, que podem ser solicitadas explicitamente por meio dos comandos principais apresentados no Quadro 24.

Quadro 24 - Categorias padrão para busca de informações referentes à previsão do tempo

Categoria	Descrição
Previsão do tempo para *	Explicitamente irá buscar no site Wikipedia pelo termo capturado pelo coringa e retornar um breve sumário sobre o assunto. Quando o termo possuir mais de uma página, o *bot* deve solicitar ao usuário e tratar a desambiguação.
Previsão do tempo para * em *	Para este tipo de perguntas o *bot* primeiro busca um coringa correspondente na base RDF e, em caso negativo, tanta buscar na Wikipedia pelo termo.

Fonte: autoria própria.

O protótipo possui ainda uma variedade de alternativas para a busca de informações de previsão do tempo, conforme apresentadas no Quadro 25, em sua maioria, iniciando pela expressão “SERA QUE...”.

Quadro 25 - Categorias alternativas para a previsão do tempo

Categoria	Descrição
Será que preciso levar guarda-chuva	Utilizando dados de previsão do tempo o *bot* deve indicar uma resposta adequada.
Será que chove *	O comando pode ser seguido do dia da semana ou período ("final de semana", por exemplo) e retornará uma breve previsão referente ao que foi perguntado. Exemplo: **Interlocutor**: Será que chove final de semana **Robô**: Parece que o final de semana será de sol entre nuvens, com 2% de chance de chover. É um bom sinal?
Será que esquenta *	
Estou congelando hoje	A resposta esperada para essa entrada é que o protótipo indique se a temperatura irá continuar baixa ou aumentar nos próximos dias.

Fonte: autoria própria.

As respostas para as perguntas envolvendo previsão do tempo irão se adaptar conforme a pergunta, fornecendo apenas as informações relevantes aos assuntos presentes nas perguntas, diferente do comando principal, que gera uma resposta mais completa e envolvendo todos os aspectos de uma previsão do tempo.

4 CONCLUSÃO

Robôs de conversação e aplicativos que visam melhorar a produtividade já existem há um tempo, no entanto, a tecnologia de conversão de fala em texto ainda não havia alcançado uma maturidade suficiente para que a voz pudesse compor interfaces completas. O panorama mudou e a demanda por assistentes controlados por voz tem crescido com os anos, fruto do avanço na tecnologia e estudos nas áreas. Pensando nisso, surgiu a ideia de combinar as técnicas de construção de bases AIML dos *chatterbots* com uma interface comandada por voz para formar um dispositivo que fornecesse assistência virtual para um determinado usuário.

Inicialmente a conversão de áudio em texto seria de forma *offline* com a API Coruja combinando o decodificador Julius com um modelo acústico em português. Porém, as limitações de *hardware* do Raspberry Pi para processar a voz em texto nos testes inviabilizaram a questão e foi preciso recorrer à uma alternativa online que se mostrou mais eficiente. Apesar disso, o Raspberry Pi 3 Modelo B combinado com o interpretador AIML desenvolvido em Python se mostraram bem eficazes durante a implementação do protótipo.

A base padrão do *chatterbot* A.L.I.C.E e a linguagem AIML mostram-se de grande importância para trabalhos na área por serem consistentes e uma alternativa rápida e fácil para o desenvolvimento de robôs de conversação. No entanto, a linguagem ainda esbarra na forma em que a base aprende e cresce, visto que depende principalmente de um administrador para o conhecimento seja acurado e o cérebro do robô cresça. Para trabalhos com foco em língua portuguesa existe ainda o problema, pois, as bases oficiais mais completas são todas em inglês.

A combinação da estrutura de conversação e das funcionalidades especificas do protótipo conseguem fornecer o auxílio esperado para o usuário, ainda que muitos dos temas tenham sido simplificados em virtude das limitações da entrada em formato de voz. O *bot* ainda se perde em muitos momentos, pois a linguagem natural é muito rica sendo quase impossível para um robô de conversação que busca por padrões de sentenças, conseguir abranger toda a gama de possibilidades existentes. A solução é ir aumentando gradativamente com o tempo a base para que o cérebro consiga identificar o máximo possível. As funcionalidades cumprem o papel proposto, com a brecha de poderem ser escaladas por algoritmos de *machine learning* para fornecer previsão e sugestão de ações, ou ainda, servir de interface para controlar outras funções da residência, como luzes e eletrodomésticos.

REFERÊNCIAS

BIRD, Steven; KLEIN, Ewan; LOPER, Edward. **Natural language processing with Python**. Sebastopol: O'Reilly Media, Inc., 2009.

BRAGA, Petrônio de Luna. **Reconhecimento de voz dependente de locutor utilizando Redes Neurais Artificiais**. 2006. Trabalho de Conclusão de Curso (Graduação) – Graduação em Engenharia da Computação, POLI-UPE, Recife.

BRANDÃO, César Filipe Monteiro. **Avaliação de agentes de conversação**: a influência de elementos multimédia. 2012. Dissertação (Mestrado em Multimédia) - Programa de Mestrado, Faculdade de Engenharia do Porto, Porto.

CASTANHO, Carla Lisiane de Oliveira. **A avaliação do uso de chatterbots no ensino através de uma ferramenta de autoria**. 2002. Dissertação (Mestrado em Ciência da Computação, Área de Concentração: Sistemas de Conhecimento) - Programa de Pós-Graduação em Ciência da Computação, UFSC, Florianópolis.

CHOWDHURY, Gobinda G. Natural language processing. **Annual Review of Information Science and Technology,** São Paulo, v.37, n.1, p. 51–89, 2003.

COMARELLA, Rafaela Lunardi; CAFÉ, Ligia Maria Arruda. Chatterbot: conceito, características, tipologia e construção. **Informática & Sociedade: Estudos (Inf. & Soc.:Est.),** João Pessoa, v.18, n.2, p. 55–67, 2008.

GALVÃO, Adjamir de Moura. **Persona-AIML**: uma arquitetura para desenvolver *chatterbots* com personalidade. 2003. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, UFPE, Recife.

HAYKIN, Simon. **Redes neurais**: princípios e práticas. Porto Alegre: Bookman, 2001.

KATSAMANIS, Athanasios et al. Robust far-field spoken command recognition for home automation combining adaptation and multichannel processing. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), 2014, Florence. **Anais eletrônicos**... Florence: University of Pisa, 2014. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6854664>. Acesso em: 8 set. 2014.

LEE, Kwang; GRICE, Roger. The Design and Development of User Interfaces for Voice Application in Mobile Devices. In: IEEE INTERNATIONAL PROFESSIONAL COMMUNICATION CONFERENCE, 2006, Saratoga Springs. **Anais eletrônicos**... Saratoga Springs: New York, 2006. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4114175> Acesso em: 8 set. 2014.

LEITÃO, Daniel Almeida. **Um *chatterbot* para um ambiente de ensino de gerência de projetos**. 2004. Trabalho de Conclusão de Curso (Graduação) – Graduação em Ciência da Computação, UFPE, Recife.

LEONHARDT, Michelle Denise. **Doroty**: um chatterbot para treinamento de profissionais atuantes no gerenciamento de redes de computadores. 2005. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Computação, UFRGS, Porto Alegre.

LEONHARDT, Michelle D. et al. ELEKTRA: Um *chatterbot* para uso em ambiente educacional. **Renote**, Porto Alegre, v. 1, n. 2, ago. 2003. Disponível em: < http://seer.ufrgs.br/renote/article/view/14336> Acesso em: 15 dez. 2014.

LOUZADA, Jailton Alkimin. **Reconhecimento automático de fala por computador**. 2007. Trabalho de Conclusão de Curso (Graduação) – Graduação em Ciência da Computação, PUC, Goiás.

MACHADO, Thiago José Marques. **Integrando chatterbot e agente animado de interface em um ambiente virtual de aprendizagem**. 2005. Monografia (Especialização em MBA e Banco de Dados) - Programa de Pós-Graduação Especialização em MBA e Banco de Dados, UNESC, Criciúma.

MILHORAT, Pierrick et al. Building the next generation of personal digital Assistants. In: INTERNATIONAL CONFERENCE ON ADVANCED TECHNOLOGIES FOR SIGNAL AND IMAGE PROCESSING (ATSIP), 2014, Sousse. **Anais eletrônicos**... Sousse: Tunisia, 2014. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6834655> Acesso em: 8 set. 2014.

MOURA, Thiago José Marques. **Um chatterbot para aquisição automática de perfil do usuário**. 2003. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, UFPE, Recife.

NETO, Nelson; BATISTA, Pedro; KLAUTAU, Aldebaro. VOICECONET: a collaborative framework for speech-based computer accessibility with a case study for brazilian portuguese. **Modern Speech Recognition Approaches with Case Studies**, p. 303–326, 2012.

NEVES, André Menezes Marques das. **iAIML: um mecanismo para o tratamento de intenção em *chatterbots***. 2005. Tese (Doutorado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, UFPE, Recife.

OLIVEIRA, Rafael et al. Recursos para desenvolvimento de aplicativos com suporte a reconhecimento de voz para desktop e sistemas embarcados. In: XII WORKSHOP DE SOFTWARE LIVRE (WSL), 2011, Porto Alegre. **Anais eletrônicos**... Porto Alegre: Centro de Eventos PUCRS, 2011. Disponível em: <http://wsl.softwarelivre.org/2011/> Acesso em: 15 dez. 2014.

OLIVEIRA, Hilário Tomaz Alves de et al. Dr. Pierre: um chatterbot com intenção e personalidade baseado em ontologias para apoiar o ensino de psiquiatria. In: XXI SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇão (SBIE), 2010, João Pessoa. **Anais eletrônicos**... João Pessoa: Hotel Tropical Tambaú, 2010. Disponível em: <http://www.br-ie.org/pub/index.php/sbie/article/view/1468> Acesso em: 15 dez. 2014.

PANDORABOTS. **AIML reference**. 2019. Disponível em <https://pandorabots.com/docs/aiml-reference/> Acesso em: 08 jul. 2019.

PAULA, Leonam João Leal de. **Desenvolvimento de aplicativo para dispositivos móveis para coleta de dados georreferenciados através de reconhecimento de voz**. 2013. Dissertação (Mestrado em Ciências. Área de concentração: Engenharia de Sistemas Agrícolas) - Programa de Pós-Graduação em Engenharia de Sistemas Agrícolas, Universidade de São Paulo, Piracicaba.

PERICO, Alisson; SHINOHARA, Cindi Sayumi; SARMENTO, Cristiano Dellani. **Sistema de reconhecimento de voz para automatização de uma plataforma elevatória**. 2014. Trabalho de Conclusão de Curso (Graduação) - Graduação em Engenharia Industrial Elétrica, Universidade Tecnológica Federal do Paraná, Curitiba.

PREECE, Jennifer; ROGERS, Yvonne, SHARP, Helen. **Design de interação**: além da interação humano-computador. Porto Alegre: Bookman, 2013.

RABELLO, R. S.; MARCHI, Ana Carolina Bertoletti de; FOSSATTI, Matheus. **AGEBOT: um chatterbot em AIML voltado a responder questões sobre epilepsia**. In: XI WORKSHOP DE INFORMÁTICA MÉDICA, 2011, Natal. Anais do WIM, 2011.

RUSSEL, Stuart; NORVIG, Peter. **Inteligência artificial**: tradução da terceira edição. Rio de Janeiro: Elsevier, 2013.

SGARBI, Julio A. **Domótica inteligente**: automação residencial baseada em comportamento. 2007. Dissertação (Mestrado em Engenharia Elétrica) - Programa de Mestrado em Engenharia Elétrica, Centro Universitário da FEI, São Bernardo do Campo.

SILVA, Carlos Patrick Alves da. **Um software de reconhecimento de voz para português brasileiro**. 2010. Dissertação (Mestrado em Engenharia Elétrica. Área de concentração: Engenharia de Telecomunicações) - Programa de Pós-Graduação em Engenharia Elétrica, UFPA, Belém.

STERLING, Keith. **Program-Y**: background. 2019. Disponível em <https://github.com/keiffster/program-y/wiki/Background>. Acesso em: 08 jul. 2019.

THE MAGPI MAGAZINE. **Raspberry Pi 3**: specs, benchmarks & testing. 2016. Disponível em <https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/>. Acesso em: 08 jul. 2019.

VIEIRA, Renata; LOPES, Lucelene, Gobinda G. Natural Language Processing. **Linguagens especializadas em *Corpora***: modos de dizer e interfaces de pesquisa**,** Porto Alegre, n.1, p. 183–201, 2010.

WALLACE, Richard S. **AIML**: Artificial Intelligence Markup Language. 2014. Disponível em: <http://www.alicebot.org/aiml.html/> Acesso em: 15 dez. 2014.