UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

ADA: ASSISTENTE PESSOAL AUTOMATIZADO BASEADO EM LINGUAGEM NATURAL

HÉLINTON PEREIRA STEFFENS

BLUMENAU
2018

HÉLINTON PEREIRA STEFFENS

ADA: ASSISTENTE PESSOAL AUTOMATIZADO BASEADO EM LINGUAGEM NATURAL

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Profa. Joyce Martins, Mestre – Orientadora

BLUMENAU
2018

ADA: ASSISTENTE PESSOAL AUTOMATIZADO BASEADO EM LINGUAGEM NATURAL

Por

HÉLINTON PEREIRA STEFFENS

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Profa. Joyce Martins, Mestre – Orientadora, FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB
Membro:	Profa. Andreza Sartori, Doutora – FURB

Blumenau, 13 de julho de 2018.

Dedico este trabalho à minha mãe, que sempre sonhou e incentivou a conclusão do meu curso de graduação.

AGRADECIMENTOS

Agradeço primeiramente à minha mãe, Maria de Lourdes Pereira, que sempre me apoiou e incentivou nos meus estudos.

À minha namorada, Ana Sueli Kratz e família, pelo apoio e compreensão durante todo o período da graduação.

À minha orientadora, Joyce Martins, por sempre me ajudar e apoiar desde antes e durante o trabalho, com suas ideias, opiniões, comprometimento e dedicação.

Na travessia da vida, uma pessoa certamente terá de superar correntes críticas em muitos lugares. (...) Disposição para transpor obstáculos é necessária na travessia da vida – o que exige espírito preparado para superar quaisquer acontecimentos críticos.

Musashi

RESUMO

Este trabalho apresenta o desenvolvimento de um assistente pessoal automatizado para auxiliar o usuário a gerenciar atividades e necessidades, sendo capaz de dialogar em Português e Inglês, respondendo e/ou compreendendo indagações e afirmações do usuário. Quando um usuário inicia um diálogo, o assistente verifica se a interação foi realizada por voz ou por texto. Caso seja por voz, o sistema transforma a fala em texto, utilizando-se de uma biblioteca de Automatic Speech Recognition (ASR), e identifica o idioma da interação. A frase é então analisada pelo Processamento de Linguagem Natural (PLN) para extrair sujeito, verbo e complementos, bem como identificar se é uma afirmação, uma pergunta ou uma negação. As afirmações são armazenadas em formato de grafo pela web semântica assim permitindo ao usuário realizar consultas e buscas (perguntas), utilizando-se de regras de dedução e de raciocínio. As respostas às perguntas são inferidas dos modelos de conhecimento do usuário e podem ser retornadas por voz ou por texto. Caso a saída seja por voz, o assistente transforma as respostas geradas de texto para voz utilizando-se de uma biblioteca de síntese de voz ou Text To Speech (TTS). O trabalho proposto atingiu os objetivos elencados, com a restrição de que as frases obrigatoriamente necessitam ter sujeito e verbo e opcionalmente complementos.

Palavras-chave: Assistente pessoal automatizado. Processamento de linguagem natural. Reconhecimento e síntese de voz. Web semântica.

ABSTRACT

This monograph presents the development of an automated personal assistant to help users manage their activities, being able to converse in Portuguese and English, answering and/or understanding questions and user statements. Upon receiving a sentence, the assistant verifies if the interaction was carried out by text or voice. If it is by voice, the system translates the speech into text, using an Automatic Speech Recognition (ASR) library, and identifies the language of the interaction. Then, the sentence is analyzed by the Natural Language Processing (PLN) to extract subject, verb and complements, as well as identify if it is a statement, a question or a negation. The statements are stored in a graph format by the semantic web, thus allowing the user to perform queries and searches (questions), using rules of deduction and reasoning. The answers to questions are inferred from user knowledge models and can be returned by voice or text. In case the output is voice, the assistant transforms generated responses from text to speech using the Text To Speech (TTS) library. The proposed work reached the initials objectives, with the restriction that the phrases required need to have subject and verb and optionally complements.

Key-words: Automated personal assistant. Natural language processing. Voice recognition and synthesis. Semantic Web.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

ASR – Automatic Speech Recognition

HTTP – HyperText Transfer Protocol

JWT – JSON Web Tokens

OWL – Ontology Web Language

PLN – Processamento de Linguagem Natural

POS tagging – Part Of Speech tagging

RF – Requisito Funcional

RNF – Requisito Não Funcional

TTS – Text to Speech

UC – Use Case / Casos de Uso

UML – Unified Modeling Language

SUMÁRIO

1 INTRODUÇÃO

O gerenciamento do tempo e de atividades é um dos pontos chaves para se conseguir aumentar a produtividade humana, sendo que, de acordo com o “especialista em gerenciamento de tempo Christian Barbosa, 46% dos profissionais trabalham mais de 9 horas por dia, sendo que 80% deles afirmam gastar de 1 a 3 horas por dia em tarefas inúteis” (ESPINHA, 2016, p. 1). Esta perda de tempo em atividades inúteis é em decorrência de uma má gestão das atividades e do tempo de cada indivíduo. Algumas pessoas vivem com tantas atribuições e responsabilidades, que o senso de urgência passou a ser um eterno companheiro, sendo que viver ocupado parece ser inevitável, afinal todas as atividades exercidas parecem ser imprescindíveis: estudar, trabalhar, cuidar da casa, planejar viagens, entre outras. Como para as pessoas não é possível eliminar nenhuma dessas tarefas, elas têm a sensação de serem escravas do seu estilo de vida. No entanto, conforme Espinha (2016) cita, quando se consegue ordenar os afazeres e priorizar as atividades, sabe-se exatamente o que e quando deve ser feito, respeitando prazos e dando a atenção necessária àquilo que traz resultados. Ademais, ao fazer o gerenciamento de atividades, a pessoa programa seus afazeres para serem concluídos dentro de prazos, o que aumenta a produtividade do indivíduo.

Utilizar assistentes pessoais digitais para gerenciar as atividades é um instrumento para melhorar o gerenciamento do tempo, pois permitem acompanhar e priorizar as atividades e, por vezes, identificar alguma tarefa que possa ter a sua execução delegada a outra pessoa. Com isso, atribuições e responsabilidades são reduzidas, permitindo as pessoas focarem no que realmente importa. Então, um assistente pessoal digital pode ser definido, nas palavras de Rouse (2014), como sendo um programa que pode entender linguagem natural e executar tarefas pelo usuário tal qual poderiam ser executadas por um assistente pessoal ou uma secretária. Algumas das tarefas que os assistentes pessoais digitais podem fazer são: enviar e ler em voz alta mensagens de texto ou e-mail, pesquisar números de telefone, antecipar pedidos, fazer chamadas e lembrar o usuário sobre compromissos. Os assistentes digitais mais populares atualmente incluem Siri da Apple, Google Assistente da Google e Cortana da Microsoft.

Atualmente, os assistentes pessoais digitais são classificados em dois tipos (PREDICTIVE ANALYTICS TODAY, 2017): os assistentes pessoais automatizados e os agentes pessoais inteligentes. Os primeiros executam tarefas como um ajudante, podendo ser desde fazer reservas para jantar, comprar bilhetes de eventos, fazer arranjos de viagens a fornecer informações com base na entrada ou nos comandos de voz. Os segundos executam

automaticamente tarefas de gerenciamento ou tratamento de dados com base em informações e eventos online, muitas vezes sem inicialização ou interação do usuário.

Recentemente, em 2015 e 2016, a empresa Tractica LLC (2016) realizou um levantamento da utilização de assistentes pessoais digitais e fez também uma previsão para os anos seguintes. Verificou que o uso de assistentes digitais tem aumentado em vista das facilidades e funcionalidades crescentes e estimou que o número de usuários desses aplicativos aumentará de 390 milhões em 2015 para 1,8 bilhões em todo o mundo até o final de 2021. Ainda, o número de empresas que utilizam assistentes pessoais corporativos passará de 155 milhões em 2015 para 843 milhões até 2021. O número de consumidores e de empresas que utilizam assistentes digitais está crescendo rapidamente graças à inovação acelerada e à escalabilidade de tecnologias subjacentes, como Processamento de Linguagem Natural (PLN) e inteligência artificial (TRACTICA LLC, 2016).

Assim, é proposto neste trabalho o desenvolvimento de um assistente pessoal automatizado, capaz de fornecer informações para o usuário sobre suas atividades, necessidades, prazos e lembretes, além de alertá-lo sobre compromissos próximos. Para que o assistente possa compreender interações de cunho específico, como quando o usuário precisa ir a um determinado lugar, por exemplo, ele possui a capacidade de aprender a identificar novos termos, lugares e objetos.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver um assistente pessoal automatizado que interaja via comandos de voz e texto.

Os objetivos específicos são:

a) disponibilizar um aplicativo móvel na plataforma Android para a interação com o usuário;
b) permitir que o usuário interaja com o sistema nos idiomas Inglês e Português;
c) interagir com o usuário no mesmo idioma que ele esteja utilizando;
d) usar web semântica para identificar novos termos, lugares e objetos.

1.2 ESTRUTURA

A presente monografia encontra-se dividida em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusão. O capítulo 2 fornece um embasamento teórico a respeito dos principais assuntos abordados no presente trabalho, explanando conceitos como processamento de voz, PLN, extração de propriedades de uma frase e web

semântica. Também são apresentados três trabalhos correlatos a este. Na sequência, o capítulo 3 expõe os principais pontos do desenvolvimento, tais como: os requisitos, a arquitetura e os diagramas de casos de usos e de classes. Além disso, também são apresentados os *frameworks* e as Application Programming Interface (APIs) utilizadas, a implementação e operacionalidade da aplicação, bem como são analisados os resultados obtidos. O capítulo 4 encerra a monografia expondo as conclusões obtidas e sugere extensões que poderão ser feitas a partir deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo explorar os principais assuntos necessários para a realização do trabalho. Desta forma, o capítulo foi subdividido em três partes, onde a seção 2.1 trata do processamento de voz, incluindo a síntese e o reconhecimento. Já a seção 2.2 descreve o processamento de linguagem natural e como a extração de sujeito, verbo e complementos de sentenças, que neste trabalho é descrito como extração de triplos, é realizada. A seção 2.3 apresenta a utilização da web semântica para armazenar triplos e realizar consultas. Por fim, na seção 2.4 são descritos três trabalhos correlatos.

2.1 PROCESSAMENTO DE VOZ

As tecnologias de processamento de voz podem ser utilizadas para desenvolver aplicativos que facilitam a interação humano computador. Basicamente, tem-se o reconhecimento de voz ou Automatic Speech Recognition (ASR) e a síntese de voz ou Text To Speech (TTS).

Softwares ASR são um tipo de software que captura conteúdo de áudio e o transcreve em palavras escritas em um processador de texto ou outro destino de exibição (SPEECH..., 2018). Esse tipo de software de reconhecimento de voz é útil para quem precisa gerar conteúdo escrito sem digitação manual ou para pessoas com deficiências que tenham dificuldades no uso do teclado.

Saksamudre, Shrishrimal e Deshmukh (2015) afirmam que existem dois tipos de sistemas de reconhecimento de voz: o baseado em enunciados e o baseado no modelo de locutor. O modelo baseado em enunciados é classificado em três tipos de algoritmos: os de palavras isoladas, os de palavras conectadas e os de fala contínua. Os algoritmos de palavras isoladas são adequados para situações onde o usuário é obrigado a fornecer palavras ou comandos isolados. Os de palavras conectadas são similares aos de palavras isoladas, mas permitem comandos compostos de múltiplas palavras com um único significado. Os algoritmos de fala contínua, por sua vez, permitem ao usuário falar naturalmente. São similares a um sistema de ditado ou de fala espontânea, consistindo no reconhecimento natural de palavras, podendo compreender palavras pronunciadas de forma errada, falsos inícios e até não palavras. Já o modelo baseado no locutor pode ser implementado de três formas: (a) o dependente do locutor, que é desenvolvido para um tipo específico de falante, tendo uma boa taxa de acerto para esse tipo específico e baixa taxa de acerto quando usado para outros tipos; (b) o independente do locutor, que pode realizar o reconhecimento sem qualquer treinamento anterior, sendo utilizado em sistemas de respostas interativas por voz;

(c) o modelo de locutor adaptativo, que se utiliza de dados anteriores de outros locutores para tentar realizar o reconhecimento de voz de outros falantes, assim aumentado a sua taxa de acertos.

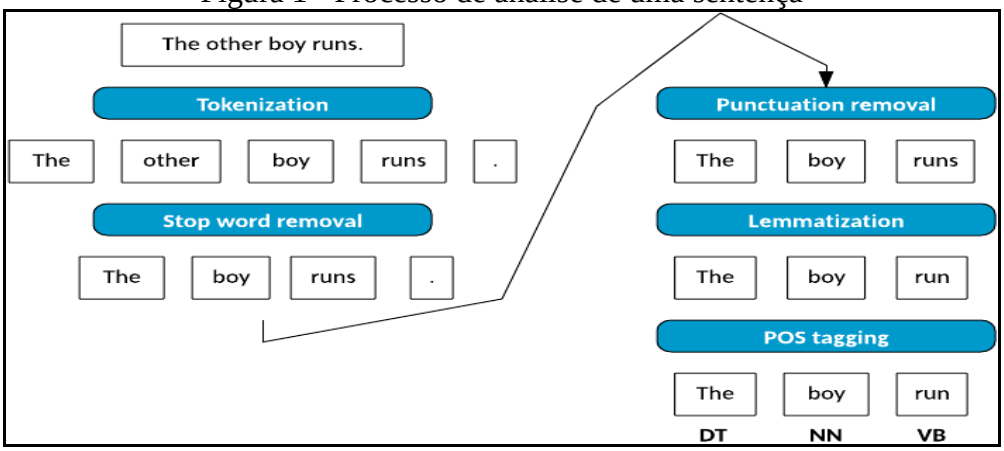
Softwares TTS realizam o processo inverso ao dos softwares ASR, ou seja, convertem textos escritos em unidades de fala para apresentação de áudio (TEXT..., 2018). O uso de TTS é comum em aplicações que buscam renderizar saída de áudio a partir de texto digital para ajudar aqueles que não conseguem ler ou falar, por exemplo. Para tanto, o programa tem que converter palavras em fonemas, as menores unidades de pronúncia de fala. Com o tempo, os especialistas observaram algumas práticas recomendadas para o desenvolvimento de TTS, incluindo bases de fonemas e abordagens concatenativas com análise preditiva, requisitos mínimos de memória e facilidades de configuração, além do tratamento de ambiguidades e de renderização mais precisa (TEXT..., 2018).

2.2 PROCESSAMENTO DE LINGUAGEM NATURAL

De acordo com Kiser (2016), o PLN é a maneira que os computadores analisam, compreendem e derivam significado da linguagem humana de uma forma útil e inteligente. Pode ser usado para organizar e estruturar o conhecimento para realizar tarefas, tais como: sumarização de textos, tradução automática, extração de relacionamentos, análise de sentimentos, reconhecimento de voz, entre outras.

Para realizar o processamento de uma sentença, Jones (2017) define que existem alguns passos que devem ser implementados, como mostrado na Figura 1. Para realizar a extração do significado de uma frase, tem-se cinco etapas: *tokenization*, *stop word removal*, *punctuation removal*, *lemmatization*, *part-of-speech* (POS) *tagging*.

Figura 1 - Processo de análise de uma sentença



Fonte: Jones (2017).

Jones (2017) define que *tokenization* é processo de separar a sentença em um conjunto completo de palavras individuais que a compõem, isto é, simplesmente divide-se a sentença em suas partes individuais (ou *tokens*). Conforme exemplificado na Figura 1, os *tokens* que compõem a sentença são as palavras *The*, *other*, *boy*, *runs* e o ponto final. O autor diz que o processo seguinte, *stop word removal*, é utilizado comumente para remover palavras subjacentes de forma a permitir que o foco seja nas palavras mais importantes da sentença. Não existe uma definição única do conjunto de palavras que compõem as *stop words*, mas existem palavras comumente aceitas a serem removidas, como preposições, artigos definidos ou adjetivos, por exemplo. Na próxima etapa é feita a remoção da pontuação (*punctuation removal*). Jones (2017) define que pontuação neste contexto não se refere unicamente a vírgulas e pontos, mas também a outros símbolos especiais usados, como parênteses, apóstrofos, aspas, pontos de exclamação, entre outros. A *lemmatization*, também conhecida por *stemming*, tem por objetivo reduzir as palavras a sua forma base ou raiz (JONES, 2017). Por exemplo, na Figura 1, a palavra *runs* será reduzida a *run.* Por último, tem-se a etapa de POS *tagging*, que é o processo de marcar uma palavra conforme seu contexto em uma frase. No Quadro 1 estão relacionadas as *tags* com as quais cada palavra pode ser marcada. Assim, na Figura 1, por exemplo, a palavra *the* é marcada como um artigo definido (DT), enquanto a palavra *boy* é marcada como um substantivo (NN) e a palavra *run* é marcada como um verbo (VB).

Quadro 1 - Lista de *tags* POS *tagging*

tag	**descrição**	***tag***	**descrição**
CC	coordinating conjunction	PRP$	possessive pronoun
CD	cardinal number	RB	adverb
DT	determiner	RBR	adverb, comparative
EX	existential *there*	RBS	adverb, superlative
FW	foreign word	RP	particle
IN	preposition or subordinating conjunction	SYM	symbol
JJ	adjective	TO	*to*
JJR	adjective, comparative	UH	interjection
JJS	adjective, superlative	VB	verb, base form
LS	list item marker	VBD	verb, past tense
MD	modal	VBG	verb, gerund or present participle
NN	noun, singular or mass	VBN	verb, past participle
NNS	noun, plural	VBP	verb, non-3rd person singular present
NNP	proper noun, singular	VBZ	verb, 3rd person singular present
NNPS	proper noun, plural	WDT	wh-determiner
PDT	predeterminer	WP	wh-pronoun
POS	possessive ending	WP$	possessive wh-pronoun
PRP	personal pronoun	WRB	wh-adverb

Fonte: Liberman (2003).

Após os passos anteriores, pode-se utilizar um *parser* para analisar sintaticamente a sentença de forma a estabelecer as relações entre as palavras e seus modificadores, bem como gerar uma *parse tree* (STANFORD NLP GROUP, 2018). Uma *parse tree*, segundo Jurafsky e Martin (2017), é um estágio intermediário da análise semântica, tendo papel fundamental para sistemas que necessitem responder perguntas ou extrair informações de um texto ou de uma entrada do usuário. Um problema existente na análise sintática é a ambiguidade das frases e palavras, visto que uma frase pode possuir mais que uma *parse tree*. Jurafsky e Martin (2017) definem que há três tipos de *parsers: syntactic parsing, statistical parsing* e *dependency parsing*. Um *syntactic parsing* gera a estrutura sintática de uma sentença a partir de gramáticas livres de contexto, sendo que a ambuiguidade é tratada pela utilização da abordagem de programação dinâmica. O *statistical parsing* parte do princípio que com conhecimento suficiente pode-se descobrir a probabilidade de quase tudo. Nesse caso, a ambiguidade é resolvida calculando a probabilidade de cada interpretação da sentença, sendo escolhida a mais provável. Já o *dependency parsing* é baseado em gramáticas de dependência que são importantes em sistemas contemporâneos de processamento de voz e fala. Nele a estrutura sintática de uma sentença é descrita unicamente em termos das palavras (ou lemas) que a compõem e um conjunto associado de relações gramaticais binárias dirigidas que se mantêm entre as palavras.

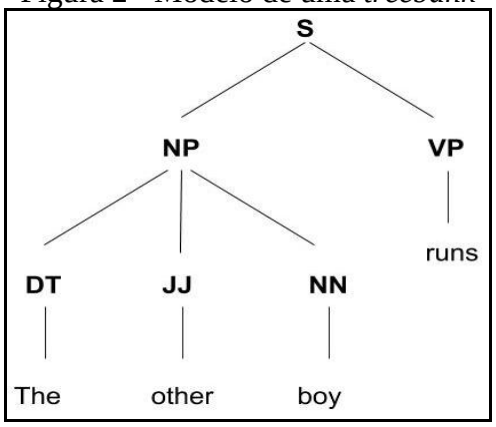
Depois de extraída a *parse tree* para uma sentença, pode-se identificar os triplos contidos nela. Leskovec, Grobelnik e Milic-Frayling (2005) definem que um triplo em uma sentença é representado pela relação entre sujeito, verbo e complementos. Rusu et al. (2007) apresentam um algoritmo para extração de triplos de sentenças baseados em uma *treebank*[1]. O algoritmo apresentado por Rusu et al. (2007) se baseia no *parser* de saída comum dos *frameworks* Stanford Parser e Apache OpenNLP. Nesses *parsers* uma sentença (S) é representada como uma árvore que tem três filhos: uma frase nominal (NP), uma frase verbal (VP) e o ponto final. A raiz da árvore sempre será S. Na Figura 2 é apresentado um exemplo de uma *treebank*, sendo que o ponto final não foi representado.

O processamento do algoritmo se dá da seguinte forma. Em primeiro lugar, precisa-se encontrar o sujeito da sentença. Para encontrá-lo, seleciona-se o nodo NP na subárvore. O sujeito será encontrado atráves de uma busca em largura e selecionando o primeiro

[1] Uma *treebank* é um corpus de texto onde cada sentença pertencente ao respectivo corpus possui uma estrutura sintática adicionada a ele (RUSU et al., 2007).

descendente de NP que é um substantivo ou pronome pessoal. Em segundo lugar, para determinar o verbo da sentença, uma pesquisa será realizada na subárvore VP. O descendente verbal mais profundo da frase verbal será o segundo elemento do trio. Em terceiro lugar, procura-se pelos complementos. Estes, se existirem, podem ser encontrados em três subárvores diferentes, todas irmãs da subárvore VP que contém o verbo. As subárvores são: PP (frase preposicional), NP e ADJP (frase adjetiva). Em NP e PP procura-se primeiro o substantivo, enquanto em ADJP procura-se primeiro o adjetivo. Com base nesse algoritmo é possível realizar a extração dos triplos de uma sentença, independente do *framework* escolhido (Stanford Parser ou Apache OpenNLP).

Figura 2 - Modelo de uma *treebank*



Fonte: elaborado pelo autor.

2.3 WEB SEMÂNTICA

A web semântica, segundo W3C Brasil (2011), dá às pessoas a capacidade de criarem repositórios de dados na web, construírem vocabulários e escreverem regras para interoperarem com esses dados. As tecnologias da web semântica fornecem um ambiente onde o aplicativo desenvolvido pode consultar os dados e fazer inferências usando vocabulários construídos.

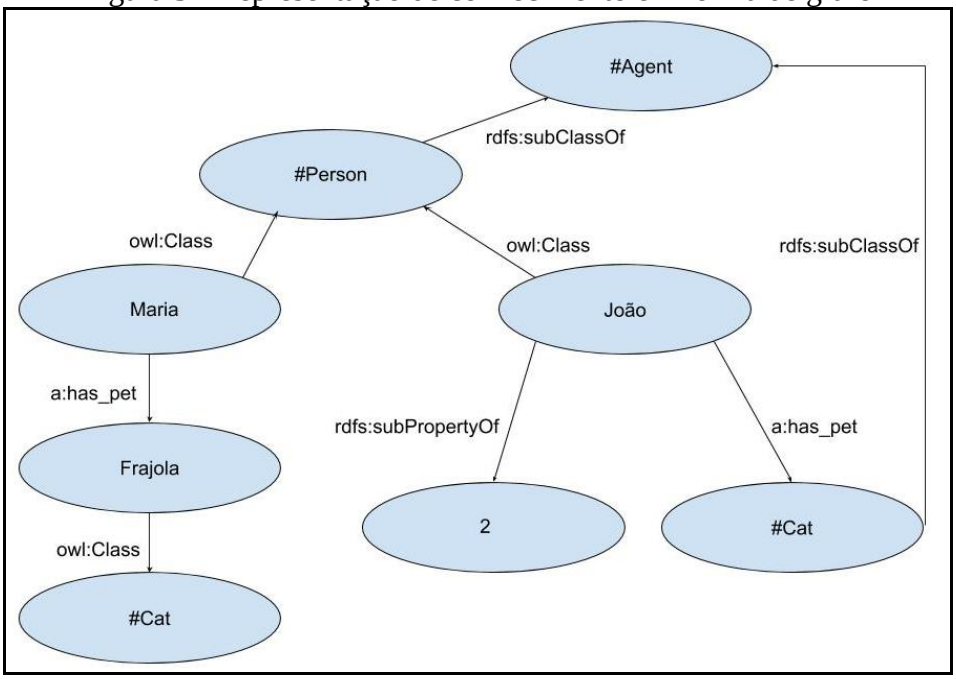
A Cambridge Semantics (2017) define a diferença entre web semântica e tecnologias relacionadas a dados, tais como bancos de dados relacionais ou a própria World Wide Web. A primeira preocupa-se com o significado, enquanto as segundas com a estrutura dos dados. Assim, a web semântica infere novos fatos a partir de dados existentes, permitindo não apenas o armazenamento e a recuperação de informações, mas também fornecendo informações inteiramente novas.

Outro ponto chave destacado pela Cambridge Semantics (2017) é o modelo de dados. Na web semântica o modelo de dados é flexível o suficiente para que novos fatos possam ser incorporados conforme necessário, incluindo novos tipos de dados não previstos no início,

contrastando com os bancos de dados relacionais onde o primeiro passo é ter uma definição de como será a modelagem do banco. Então, em aplicações onde é impossível identificar com antecedência todos os tipos de dados que devem ser armazenados – tais como sistemas de gerenciamento de conhecimento, sistemas de pesquisa e suporte ou sistemas com uma grande quantidade de dados imprevisíveis e não estruturados – a web semântica pode ser menos custosa para se implementar e manter ao longo do tempo.

Na Figura 3 é apresentado um modelo de dados para representar conhecimento na forma de um grafo. Este grafo representa o conhecimento adquirido pela web semântica, das interações com o usuário até o momento. Neste modelo, a informação é representada por uma coleção de triplos (sujeito, verbo, complementos), onde uma aresta entre dois nodos representa uma ligação entre sujeito e complemento, tendo o verbo unindo os nodos. Correia e Nunes (2002, p. 4) citam que este modelo "permite constituir uma rede de informação relacionada, onde será possível aos homens e às máquinas aplicar [sic] regras de inferência para criar deduções a partir dos significados descritos [... em] cada tripla de informação".

Figura 3 - Representação de conhecimento em forma de grafo



Fonte: elaborado pelo autor.

Uma vez organizada e estruturada a informação, é necessário relacionar os significados de palavras semelhantes, bem como agrupá-las por seus significados. A capacidade de descobrir os agrupamentos, conforme citam Correia e Nunes (2002), permite criar uma rede de conhecimentos mais ligada entre si, denominada ontologia, de onde é possível se extrair informações nem sempre visíveis no modelo, sendo utilizada para isso a Ontology Web Language (OWL). Ouksel e Sheth (1999) definem OWL como um vocabulário específico de

relacionamentos usados para descrever certos aspectos da realidade e um conjunto de suposições relativas ao significado das palavras. No Quadro 2 é apresentada a declaração das classes: `Agent`, `Person`, `Animal`, `Cat owner` e `Cat`, onde: (1) as classes `Person` e `Animal` são subclasses de `Agent`, (2) `Cat owner` é uma subclasse de `Person`, e (3) `Cat` é uma subclasse de `Animal`. Isto significa que todo recurso que for da classe `Person` ou `Animal` ou das suas subclasses poderá ser inferido como sendo também um `Agent`. Ouksel e Sheth (1999) afirmam que, entre vários outros esquemas e estruturas de classificação, incluindo técnicas baseadas em palavras chave e taxonomia, OWL é vista como sendo um esquema que fornece modelos de domínio precisos e mais completos.

Quadro 2 - Declaração de classes OWL

```
<owl:Class rdf:ID="Agent">
   <rdfs:label>Agent</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Agent"/>
</owl:Class>

<owl:Class rdf:ID="Person">
   <rdfs:label>Person</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Agent"/>
</owl:Class>
<owl:Class rdf:ID="Animal">
   <rdfs:label>Animal</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Agent"/>
</owl:Class>

<owl:Class rdf:ID="Cat_Owner">
   <rdfs:label>Cat owner</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>
<owl:Class rdf:ID="Cat">
   <rdfs:label>Cat</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Animal"/>
</owl:Class>
```

Fonte: adaptado de Orme, Yao e Etzkorn (2006).

Correia e Nunes (2002) adicionalmente citam que as ontologias fornecem um conjunto de regras de inferência, oferecendo ainda mais poder às relações entre os conceitos. Essas regras de inferência permitem que os agentes computacionais realizem deduções lógicas expressamente definidas em suas regras, conforme é mostrado no Quadro 3, onde pode-se inferir que donos de gatos tem gatos como animais de estimação e que ter um animal de estimação está associado a gostar de animais.

Quadro 3 - Regra de dedução OWL

```
Class(a:cat_owner complete intersectionOf(a:person
   restriction(a:has_pet someValuesFrom (a:cat))))

SubPropertyOf(a:has_pet a:likes)

Class(a:cat_liker complete intersectionOf(a:person
   restriction(a:likes someValuesFrom (a:cat))))
```

Fonte: adaptado de Bechhofer (2003).

2.4 TRABALHOS CORRELATOS

São apresentados três trabalhos correlatos, que possuem características semelhantes à proposta deste trabalho. Nas seções seguintes é detalhado o funcionamento do Dragon Mobile Assistant (NUANCE COMMUNICATIONS INC., 2017), do Braina (BRAINASOFT, 2017) e do Hound Voice Search & Mobile Assistant (SOUNDHOUND INC., 2017).

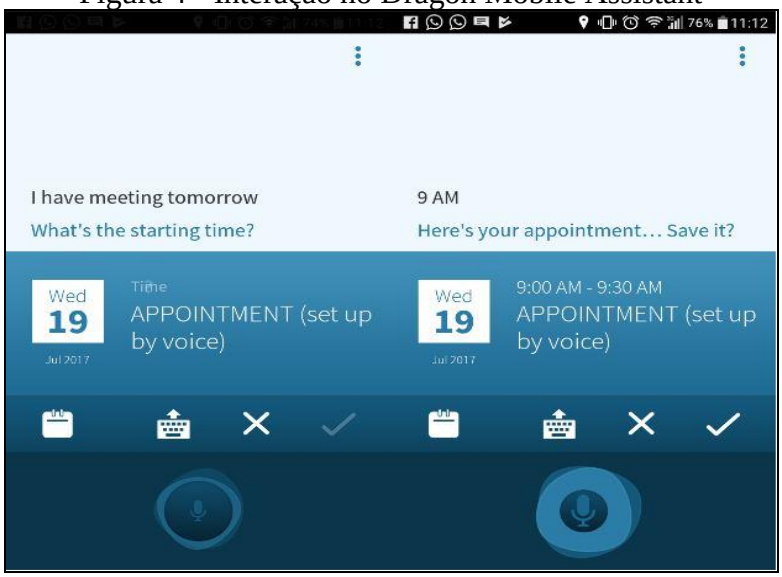
2.4.1 Dragon Mobile Assistant

Dragon Mobile Assistant é um agente pessoal inteligente desenvolvido pela Nuance Communications Inc. (2017) para o idioma Inglês, que pode enviar e receber mensagens de texto, publicar atualizações do Facebook e do Twitter e enviar e-mails, usando a tecnologia de reconhecimento de voz da Nuance. Outras funcionalidades incluem a capacidade de responder perguntas externas e funcionar como um *chatterbot*, apesar de ambas serem bem simples.

A partir de testes realizados pelo o autor deste trabalho, há alguns pontos a serem destacados que são: a qualidade de reconhecimento e síntese de voz, a capacidade de executar tarefas de forma autônoma, como enviar mensagens, fazer ligações e realizar a compra de produtos, além da automatização de tarefas, como, por exemplo, detectar que o usuário está se movendo em um veículo e então sintetizar notificações. No entanto, como ponto negativo cita-se o fato de não permitir a interação via texto, o que restringe os lugares em que se pode usar o aplicativo, devido à perda de privacidade do usuário. Além disso, não permite o gerenciamento das atividades, então não é possível alterar ou excluir uma atividade ou alerta. Por fim, como o aplicativo não possui capacidade de aprendizado, acaba errando na execução de alguns comandos simples. Por exemplo, o assistente não entende o comando "*I have a class tomorrow*" como um agendamento de atividade e tenta executar uma consulta externa.

A Figura 4 apresenta um exemplo de agendamento de atividade com o aplicativo Dragon Mobile Assistant. A interação consiste no usuário entrar com uma sentença afirmativa que contenha data ou hora ou indicação de tempo ("*I have meeting tomorrow*"). Na sequência, o aplicativo pode pedir e aguardar por outras informações (no exemplo, "*What's the starting time*?") ou confirmar a criação do compromisso ("*Here's your appointment... Save it*?").

Figura 4 - Interação no Dragon Mobile Assistant



Fonte: Nuance Communications Inc. (2017).

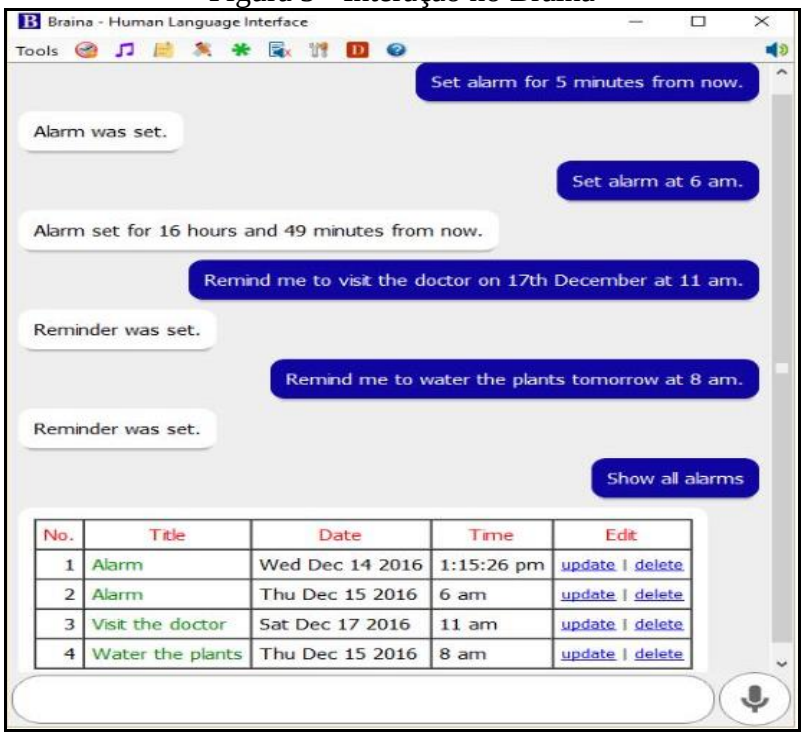
2.4.2 Braina

Braina é um agente pessoal automatizado, multi-plataforma, que permite ao usuário interagir via comandos de voz ou texto em Língua Inglesa, possibilitando assim que se tenha o controle do computador e do smartphone usando comandos em linguagem natural de forma a facilitar as tarefas diárias (BRAINASOFT, 2017). O funcionamento do assistente se dá na forma de comandos definidos em *templates* como: `Search <term> on Google`, `Remind me to <thing to do> at <time>`, `Synonyms of <word>`, `Set alarm at <time>`, `Set alarm for <time>`, além de muitos outros. A partir desta característica, a flexibilidade das interações se restringe aos *templates* pré-cadastrados.

A partir de testes realizados pelo o autor deste trabalho, há alguns pontos a serem destacados que são: a possibilidade de gerenciamento das atividades cadastradas, possibilitando alterar e/ou excluir um alerta ou atividade; a baixa capacidade de aprendizado; a capacidade de poder manter simples conversações, de responder algumas questões usando internet e de resolver equações matemáticas; além da síntese de voz. Um dos pontos negativos é o reconhecimento de voz que está disponível somente na versão paga.

Na Figura 5 é apresentado como cadastrar e consultar um alerta no aplicativo Braina, sendo que as sentenças da direita são as interações do usuário e as da esquerda são as do assistente. A interação consiste no usuário entrar com uma sentença, respeitando um dos *templates* pré-cadastrados, como `Set alarm at 6 am`. Para consultar os alertas cadastrados, basta utilizar a sentença `Show all alarms`.

Figura 5 - Interação no Braina



Fonte: Brainasoft (2017).

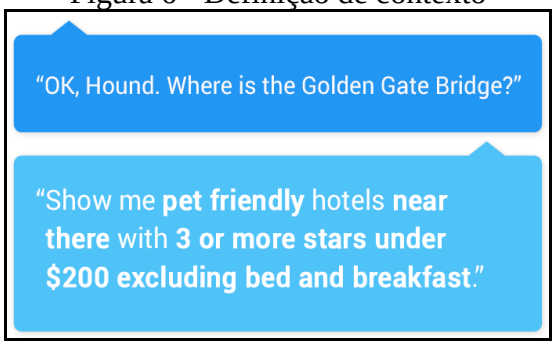
2.4.3 Hound Voice Search & Mobile Assistant

Hound Voice Search & Mobile Assistant, conforme descreve SoundHound Inc. (2017), é um assistente pessoal inteligente, desenvolvido para o idioma Inglês, que pode executar funções como: consultar a previsão do tempo, realizar ligações, enviar mensagens de texto, realizar pesquisas de objetos e lugares conforme os critérios informados, além de poder auxiliar na navegação para um endereço, verificar o mercado de ações, pesquisar e tocar músicas e até mesmo jogar jogos interativos.

O aplicativo é baseado em algoritmos de Speech-to-Meaning, que conforme SoundHound Inc. (2017), executa o processamento da linguagem natural de forma simultânea com o reconhecimento de voz, sendo, portanto, mais eficiente e preciso. Então, apesar do reconhecimento de voz não ser tão bom, normalmente se obtém bons resultados para consultas e comandos. Como características a serem destacadas, pode-se citar: a síntese de voz; a execução de consultas externas; a automatização de tarefas, como realizar chamadas, enviar mensagens e agendar atividades; e a execução de comandos levando em consideração o contexto atual. Nesse caso, como mostrado na Figura 6, quando o usuário executa uma consulta de uma localização por exemplo, no comando seguinte é possível pedir informações sobre o local sem ser necessário informá-lo novamente. Como não possui capacidade de aprendizado, o nível das conversações mantidas pelo aplicativo é muito baixo. Além disso,

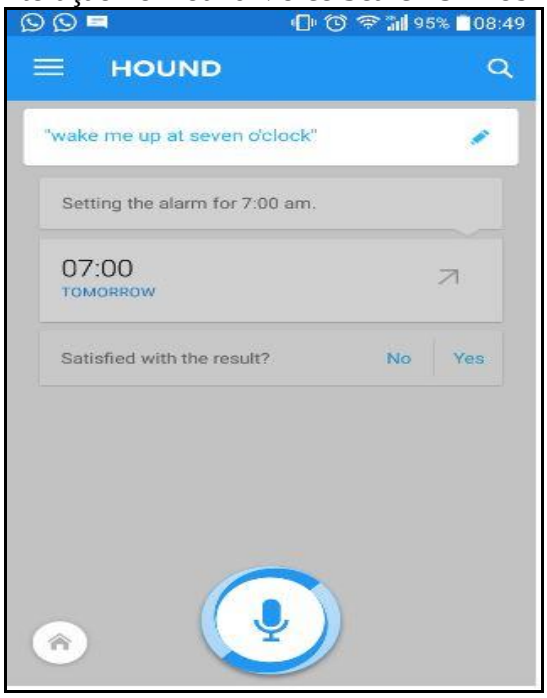
não permite o gerenciamento de atividades, alarmes e alertas cadastrados. Na Figura 7 é mostrado como é o registro de uma atividade com o aplicativo Hound Voice Search & Mobile Assistant. Deve-se entrar com uma sentença na forma imperativa ("*Wake me up at seven o'clock*"). Então o aplicativo pede para confirmar o registro da atividade.

Figura 6 - Definição de contexto



Fonte: SoundHound Inc. (2017).

Figura 7 - Interação no Hound Voice Search & Mobile Assistant



Fonte: SoundHound Inc. (2017).

3 DESENVOLVIMENTO DA ADA

O presente capítulo apresenta o desenvolvimento de um assistente pessoal automatizado chamado de Ada. Inicia com a descrição dos requisitos funcionais e não funcionais levantados para o desenvolvimento da aplicação (na seção 3.1), seguida da especificação (na seção 3.2), que apresenta a arquitetura do assistente desenvolvido, além dos diagramas de casos de uso e de classes. Já a seção 3.3 aborda assuntos referentes à implementação, listando as ferramentas e APIs utilizadas no desenvolvimento e explicando sobre a operacionalidade da aplicação. Por fim, a seção 3.4 analisa os resultados obtidos a partir do presente projeto.

3.1 REQUISITOS

O assistente pessoal automatizado proposto deve atender aos Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) apresentados no Quadro 4 e Quadro 5, respectivamente. Os requisitos funcionais estão relacionados aos casos de uso apresentados na Figura 8.

Quadro 4 - Requisitos funcionais

Requisitos funcionais	**Casos de uso (UC)**
RF01: permitir o cadastro de usuário	UC01
RF02: permitir o usuário definir se a interação será por comandos de voz ou por texto	UC02
RF03: permitir o usuário informar, alterar, visualizar e remover suas atividades, por comandos de voz ou texto	UC03, UC04, UC05
RF04: permitir o usuário informar, alterar, visualizar e remover suas necessidades, por comandos de voz ou texto	UC03, UC04, UC05
RF05: notificar o usuário sobre as atividades e necessidades diárias	UC06

Fonte: elaborado pelo autor.

Quadro 5 - Requisitos não funcionais

Requisitos não funcionais
RNF01: utilizar a API Apache OpenNLP para o processamento de linguagem natural
RNF02: utilizar a API Apache Jena para a web semântica
RNF03: ter o *backend* desenvolvido em Java
RNF04: ter o *frontend* desenvolvido para Android
RNF05: utilizar a API Cloud Speech to Text disponibilizada pela Google LLC no *frontend* para efetuar o reconhecimento de voz
RNF06: utilizar a *intent* android.speech.tts.engine.CHECK_TTS_DATA disponibilizada pelo Android no *frontend* para realizar a síntese de voz
RNF07: possuir confidencialidade, ou seja, as informações de cada usuário somente devem estar acessíveis ao mesmo
RNF08: permitir interação em Português e Inglês
RNF09: interagir com o usuário no mesmo idioma que ele esteja utilizando
RNF10: as frases a serem processadas devem conter sujeito, verbo e complementos nesta ordem
RNF11: não processar frases com erros ortográficos e gramaticais

Fonte: elaborado pelo autor.

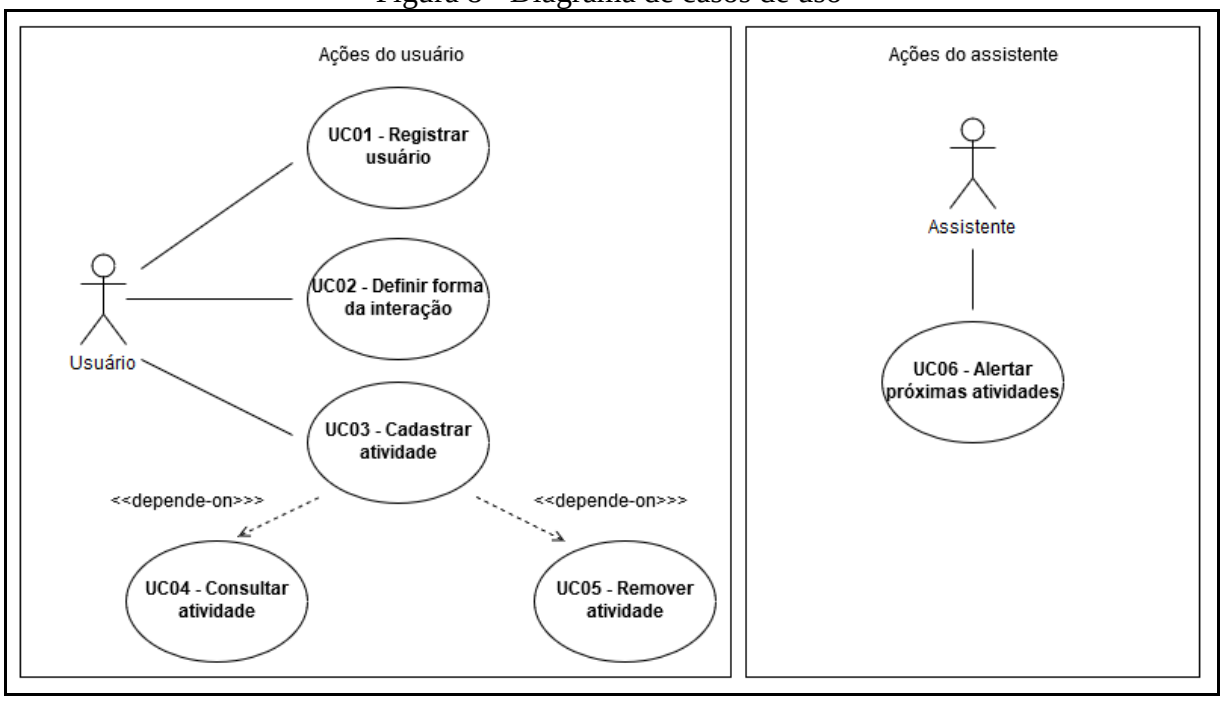
3.2 ESPECIFICAÇÃO

Nesta seção é descrita a especificação da Ada. Inicialmente é apresentada a arquitetura do assistente, seguida da explanação de como a geração da *parse tree* pela biblioteca Apache OpenNLP é utilizada pelo assistente para extração de triplos. Além disso, são expostos os diagramas de casos de uso e de classes, desenvolvidos com a ferramenta Enterprise Architect.

3.2.1 Casos de uso

O presente projeto contém seis casos de uso (em inglês Use Case - UC) que representam as principais funcionalidades do protótipo desenvolvido. O diagrama da Figura 8 foi feito utilizando a Unified Modeling Language (UML). Além disso, os UCs estão detalhados no Apêndice A.

Figura 8 - Diagrama de casos de uso



Fonte: elaborado pelo autor.

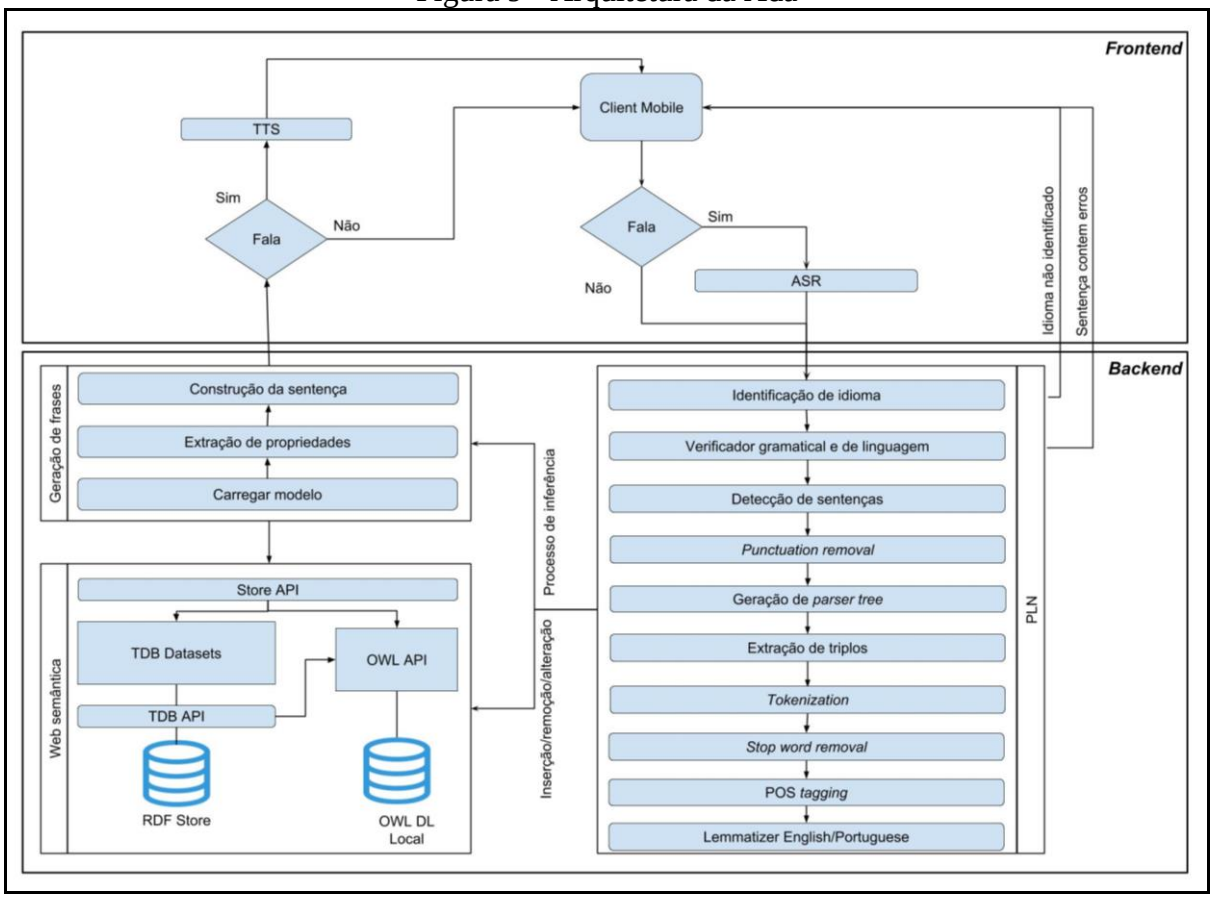
O caso de uso `Registrar usuário` representa a primeira interação do usuário com o assistente, onde é criado um nodo específico para o grafo do usuário, contendo suas informações privadas na web semântica. O caso de uso `Definir forma da interação` permite o usuário alterar a forma de dialogar com a Ada entre voz e texto. O cadastro de atividades ou de necessidades do usuário (UC03) é realizado quando o usuário informa uma determinada sentença e o algoritmo de extração de triplos identifica a frase como sendo uma afirmação. Então a mesma é salva no nodo específico do usuário. A partir do UC03, o usuário pode realizar consultas (UC04) e remover uma atividade ou uma necessidade (UC05). O usuário tem a possibilidade de realizar consultas utilizando-se das mesmas palavras ditas para

criar a atividade, tendo como exemplo a seguinte interação: “Eu preciso estudar hoje” e a consulta “Quando eu preciso estudar?”. Outra forma de realizar perguntas é utilizando-se das classes de palavras definidas na ontologia da Ada, onde para a frase dita anteriormente, pode-se efetuar uma consulta da seguinte maneira: “O que eu preciso fazer hoje?”. A remoção de atividades ou necessidades funciona de forma parecida, sendo que para removê-las, o algoritmo de extração de propriedades precisa identificar a sentença como sendo uma negação da atividade ou necessidade cadastrada. Cita-se como exemplo a afirmação “Eu tenho que arrumar a cama amanhã.” Para remover esta frase, pode ser dito “Eu não tenho que arrumar a cama”. O assistente também alerta o usuário sobre suas próximas atividades (UC06). Ou seja, todo dia a Ada envia uma notificação para o usuário avisando o que ele precisa fazer no dia de hoje.

3.2.2 Arquitetura

A arquitetura do assistente pessoal automatizado é dividida em *backend* e *frontend*, conforme mostra a Figura 9.

Figura 9 - Arquitetura da Ada



Fonte: elaborado pelo autor.

O *backend* possui três módulos: PLN, web semântica e geração de frases, buscando assim um menor acoplamento entre as camadas e visando o funcionamento independente de cada módulo. No módulo PLN tem-se: identificação do idioma, verificador gramatical e de linguagem, detecção de sentenças, *punctuation removal*, geração de *parse tree*, extração de triplos, *tokenization*, *stop word removal*, POS *tagging* e *lemmatizer English/Portuguese*. O módulo web semântica contém as lógicas de inserção, remoção, alteração e inferência de dados, as regras de dedução e a OWL. O módulo de geração de frases possui os seguintes processos: carregar o modelo, extração de propriedade e construção de sentenças. No *frontend* encontra-se a interface gráfica do assistente, ASR e TTS.

Quando um usuário inicializa um diálogo com a Ada, primeiramente é verificado se interação é realizada por voz ou por texto. Caso seja por voz, um *stream* de voz com o conteúdo dito pelo usuário é enviado via HyperText Transfer Protocol (HTTP) para a API Cloud Speech to Text para ser convertido em texto. Esta frase é então enviada via HTTP para o *backend* onde é realizada a identificação do idioma da sentença. Caso o idioma seja diferente de português ou inglês, uma mensagem informa ao usuário que somente frases em português ou inglês são aceitas. Caso contrário, é verificado se não há erros ortográficos ou gramaticais. Frases com erros não são processadas. Isso é necessário para evitar processamentos desnecessários e, consequentemente, erros inesperados. A seguir é realizada a detecção de sentenças contidas no texto informado, sendo que de cada sentença extraída é gerada uma *parse tree*. As *parse trees* alimentam o algoritmo de extração de triplos que identifica as propriedades contidas na sentença. Para cada uma das propriedades extraídas, é utilizado o *tokenization* para unir palavras compostas ou extrair palavras simples, sendo que as palavras identificadas como artigos definidos são removidas pelo processo de *stop word removal*. Posteriormente, dos *tokens* obtidos são determinadas as respectivas *tags* pelo processo de POS *tagging* para então serem extraídos os lemas de cada palavra pelo *lemmatizer*. Dessa forma, na ontologia são armazenadas apenas a forma base das palavras, evitando ter que tratar pessoas e tempos verbais. Com os triplos extraídos, utilizando a TDB API, é verificado o tipo da sentença. Caso seja uma inserção, o grafo com atividades e necessidades do usuário é alimentado. Caso seja uma remoção, busca-se no grafo do usuário o nodo contendo a informação a ser removida, sendo removido esse nodo e os nodos abaixo dele. Caso seja uma consulta, o processo é direcionado para o módulo de geração de frases para buscar a informação solicitada (carregar modelo). As propriedades do nodo contendo a informação solicitada e dos nodos abaixo são extraídas pelo processo de extração de propriedades e uma sentença com a resposta a ser dada ao usuário é construída pelo processo

de construção de sentenças. Verifica-se então se a interação com o usuário é por voz ou por texto. Caso seja por voz, é utilizada a *intent* CHECK_TTS_DATA para transformar o texto em voz e, por fim, apresentar o resultado ao usuário.

3.2.3 Geração de *parse tree* pelo Apache OpenNLP

A base da conversação realizada entre o assistente e o usuário se dá por triplos extraídos a partir de uma *parse tree*, obtida pelo Apache OpenNLP. Por isto, a geração de *parse tree* é passo essencial para o projeto. O Apache OpenNLP é um conjunto de ferramentas para PLN baseado em aprendizado de máquina, sendo que o funcionamento da biblioteca é livre de idioma. Isto significa que é possível realizar uma única implementação para diferentes idiomas. Para gerar uma *parse tree* com Apache OpenNLP deve-se implementar os seguintes passos: detecção de sentenças, *tokenization*, POS *tagging* e análise de sentenças.

A detecção de sentenças é o processo de separar as sentenças contidas em um texto. A forma que se dá este processo é exemplificada no Quadro 6, onde é apresentado um código exemplo. Este exemplo tem um texto como entrada (linhas 2 e 3) e quatro sentenças como saída: "*Hi.*", "*How are you?*", "*Welcome to Tutorialspoint.*" e "*We provide free tutorials on various technologies*". O Apache OpenNLP pode utilizar um modelo de dados pré-definidos, encontrado no próprio site da biblioteca, ou um modelo treinado manualmente.

Quadro 6 - Detecção de sentenças no Apache OpenNLP

```
 1  public static void main(String args[]) throws Exception{
 2    String sentence = "Hi. How are you? Welcome to Tutorialspoint. "
 3      + "We provide free tutorials on various technologies";
 4
 5    //Loading sentence detector model
 6    InputStream inputStream = new FileInputStream("C:/OpenNLP/ensent.bin");
 7    SentenceModel model = new SentenceModel(inputStream);
 8
 9    //Instantiating the SentenceDetectorME class
10    SentenceDetectorME detector = new SentenceDetectorME(model);
11
12    //Detecting the sentence
13    String sentences[] = detector.sentDetect(sentence);
14
15    //Printing the sentences
16    for(String s: sentences){
17      System.out.println(s);
18    }
19  }
```

Fonte: Tutorials Point (2018).

A etapa de *tokenization* contém métodos para dividir o texto bruto em partes menores (*tokens*). Ele usa a técnica de máxima entropia para decidir como realizar as divisões, sendo que a máxima entropia pode ser definida por Manning e Schutze (2002) como sendo uma estrutura para integrar informações de muitas fontes de informações heterogêneas para

classificação de dados. Essa etapa pode ser usada em tarefas como verificação ortográfica, processamento de pesquisas, identificação de partes do discurso e detecção de sentenças. No Quadro 7 é apresentado como é a implementação deste processo.

Quadro 7 - *Tokenization* no Apache OpenNLP

```
 1  public static void main(String args[]) throws Exception{
 2    String sentence = "Hi. How are you? Welcome to Tutorialspoint. "
 3      + "We provide free tutorials on various technologies";
 4
 5    //Loading the Tokenizer model
 6    InputStream inputStream = new FileInputStream("C:/OpenNLP/en-token.bin");
 7    TokenizerModel tokenModel = new TokenizerModel(inputStream);
 8
 9    //Instantiating the TokenizerME class
10    TokenizerME tokenizer = new TokenizerME(tokenModel);
11
12    //Tokenizing the given raw text
13    String tokens[] = tokenizer.tokenize(sentence);
14
15    //Printing the tokens
16    for (String t: tokens){
17      System.out.println(t);
18    }
19  }
```

Fonte: Tutorials Point (2018).

O Apache OpenNLP também trata POS *tagging* para detectar e marcar as partes do discurso de uma frase. Em vez do nome completo das partes do discurso, o Apache OpenNLP usa formas curtas de cada parte do discurso. No Quadro 8 é mostrado como o processo de POS *tagging* é realizado. Para a frase do exemplo "*Hi welcome to Tutorialspoint*" (linha 10), Tutorials Point (2018) obteve a seguinte saída: a palavra *hi* é um NNP, *welcome* é um JJ, *to* é TO e *Tutorialspoint* é um VB.

Quadro 8 - POS *tagging* no Apache OpenNLP

```
 1  public static void main(String args[]) throws Exception{
 2    //Loading Parts of speech-maxent model
 3    InputStream inputStream = new
 4                          FileInputStream("C:/OpenNLP_models/en-pos-maxent.bin");
 5    POSModel model = new POSModel(inputStream);
 6
 7    //Instantiating POSTaggerME class
 8    POSTaggerME tagger = new POSTaggerME(model);
 9
10    String sentence = "Hi welcome to Tutorialspoint";
11
12    //Tokenizing the sentence using WhitespaceTokenizer class
13    WhitespaceTokenizer whitespaceTokenizer= WhitespaceTokenizer.INSTANCE;
14    String[] tokens = whitespaceTokenizer.tokenize(sentence);
15
16    //Generating tags
17    String[] tags = tagger.tag(tokens);
18
19    //Instantiating the POSSample class
20    POSSample sample = new POSSample(tokens, tags);
21    System.out.println(sample.toString());
22  }
```

Fonte: Tutorials Point (2018).

Para conseguir analisar as sentenças e gerar uma *parse tree,* o Apache OpenNLP necessita de um modelo pré-definido, sendo que este modelo é treinado para analisar as frases fornecidas pelo usuário. No Quadro 9 é mostrado como a análise é realizada, onde a frase "*Tutorialspoint is the largest tutorial library*." tem a seguinte *parse tree*: (TOP (S (NP (NN Tutorialspoint)) (VP (VBZ is) (NP (DT the) (JJS largest) (NN tutorial) (NN library.))))).

Quadro 9 - *Parser* OpenNLP

```
 1  public static void main(String args[]) throws Exception{
 2    //Loading parser model
 3    InputStream inputStream = new FileInputStream(".../en-parserchunking.bin");
 4    ParserModel model = new ParserModel(inputStream);
 5
 6    //Creating a parser
 7    Parser parser = ParserFactory.create(model);
 8
 9    //Parsing the sentence
10    String sentence = "Tutorialspoint is the largest tutorial library.";
11    Parse topParses[] = ParserTool.parseLine(sentence, parser, 1);
12
13    for (Parse p : topParses)
14        p.show();
15  }
```

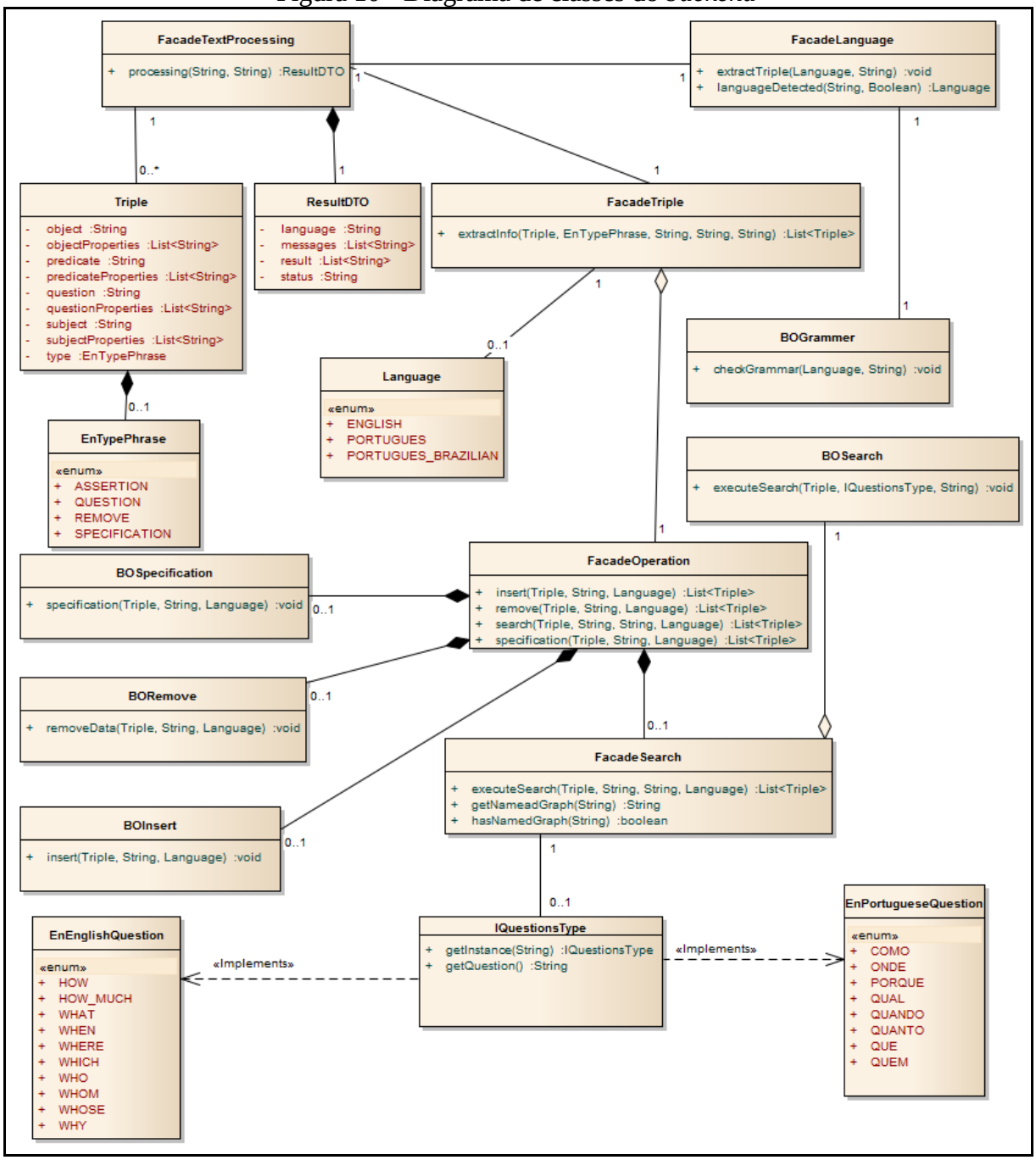
Fonte: Tutorials Point (2018).

3.2.4 Diagramas de classes

Na especificação do assistente desenvolveu-se dois diagramas de classes, um para o *backend* representado na Figura 10 e outro para o *frontend* ilustrado na Figura 11. A fim de garantir uma melhor visualização, optou-se por omitir construtores, `gets`, `sets` e demais métodos secundários.

No *backend*, a classe `FacadeTextProcessing` é a responsável por receber a sentença do usuário e orquestrar as ações necessárias para compreender e processar a entrada (método `processing`). Então, para realizar o processamento, é instanciada a classe `FacadeLanguage`, sendo utilizado o método `languageDetected` para identificar em qual idioma foi dito a frase. Também é verificado se a frase não possui erros ortográficos ou gramaticais utilizando o método `checkGrammar` da classe `BOGrammar`. O método `extractTriple` é chamado após a identificação do idioma para extrair as propriedades da frase. Essas propriedades extraídas são persistidas por objeto da classe `Triple`. Após a extração de propriedades, é instanciada a classe `FacadeOperation`, que, dependendo do tipo de `Triple`, é direcionada para uma implementação diferente. Por exemplo, se o tipo do objeto `Triple` for `QUESTION`, então será executado o método `executeSearch` da classe `BOSearch`. Os resultados obtidos da operação anterior são retornados à classe `FacadeTextProcessing` para converter a lista de triplos em sentenças a serem mostradas ao usuário.

Figura 10 - Diagrama de classes do *backend*

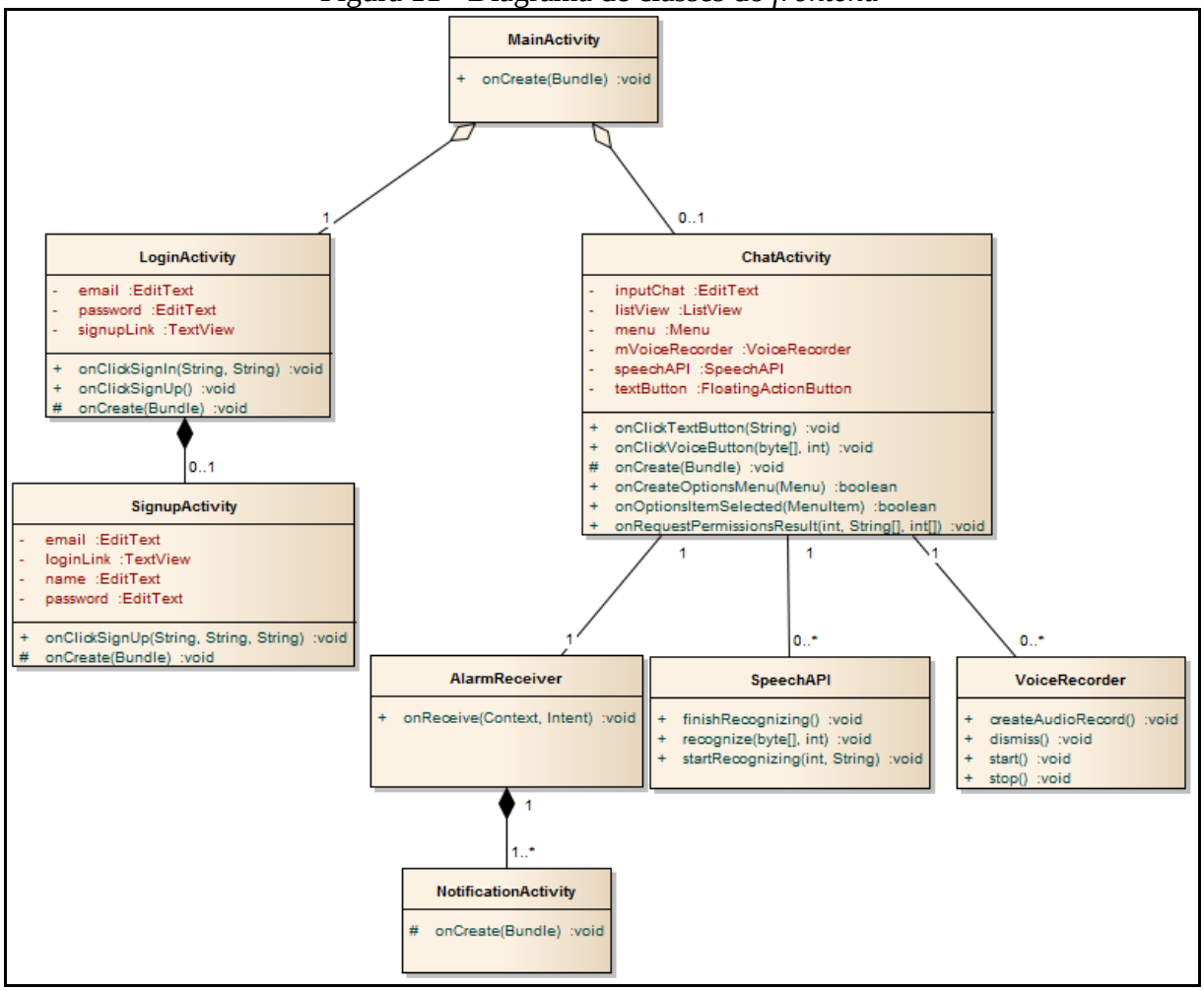


Fonte: elaborado pelo autor.

Por outro lado, no *frontend* a classe `MainActivity` é responsável por controlar os acessos a telas do assistente, sendo que ao inicializar o aplicativo verifica se o usuário possui um *token* de acesso válido gerado pelo *backend*. Em caso positivo, o usuário é direcionado para a tela de conversação que está vinculada à `ChatActivity`. Em caso negativo, o usuário é direcionado para a tela de *login* onde o vínculo é realizado com a `LoginActivity`. O usuário pode realizar o *login* pelo método `onClickSingIn` ou se registrar pelo método `onClickSingUp` que está vinculado à `SignUpActivity`. Após realizar o *login* ou se registrar,

o usuário é redirecionado a tela de conversação. Lá o usuário pode realizar a interação por texto ou por voz. Caso opte por interação por voz, o método `onClickVoiceButton` recebe o *stream* de voz do usuário até o mesmo soltar o botão com ícone de microfone. A voz do usuário é capturada pelos métodos `createAudioRecord`, `start` e `stop` da classe `VoiceRecorder`. `SpeechAPI` é responsável por realizar a comunicação com a API Cloud Speech to Text para obter a partir do *stream* de voz a fala do usuário convertida em formato de texto. A classe `AlarmReceiver` é o timer que alertar o usuário sobre suas próximas atividades e `NotificationActivity` é quem realiza a consulta das atividades.

Figura 11 - Diagrama de classes do *frontend*



Fonte: elaborado pelo autor.

3.3 IMPLEMENTAÇÃO

Nesta seção são mostradas as ferramentas e técnicas utilizadas para implementar o presente projeto. Também é explanado sobre como é feito o processamento das sentenças, como elas são compreendidas, como as frases são geradas pelo o assistente e como utilizar a aplicação desenvolvida.

3.3.1 Técnicas e ferramentas utilizadas

Para implementar este trabalho utilizou-se a linguagem de programação Java para desenvolver o *backend* e linguagem de programação Android para o *frontend*. Para realizar a comunicação entre *backend* e *frontend*, foi utilizado o protocolo HTTP para enviar e receber requisições, sendo que no *backend* optou-se por utilizar o *framework* Jersey RESTful Web Services, enquanto que no *frontend* foi utilizada a biblioteca `HttpClient` do Apache. Para a camada de segurança foi utilizado o padrão JSON Web Tokens (JWT). O PLN foi desenvolvido sobre o *framework* Apache OpenNLP, sendo que os dados utilizados para realizar os treinamentos foram os disponibilizados pela própria OpenNLP e pela Linguateca. O identificador de idiomas e os verificadores ortográficos e gramaticais foram implementados utilizando a API LanguageTool. Para construir o módulo de web semântica foi utilizado o *framework* Apache Jena, sendo que para a aplicação efetuar o reconhecimento de voz foi utilizada a API Cloud Speech to Text disponibilizada pela Google LLC e para realizar a síntese de voz foi utilizada a *intent* CHECK_TTS_DATA disponibilizada pela própria linguagem de programação Android.

3.3.2 Processamento das sentenças

Boa parte da complexidade do projeto se dá no processamento das sentenças ditas pelo usuário, sendo este processo responsável por identificar o tipo (QUESTION, ASSERTION, REMOVE, SPECIFICATION) e as propriedades contidas na frase. Para realizar esse processamento, foi implementado o algoritmo de extração de triplos de uma sentença na classe `FacadeLanguage`. Porém, para ser possível identificar quando o usuário está realizando uma pergunta, foi necessário incluir um passo a mais no algoritmo descrito por Rusu et al. (2007), qual seja: para uma frase ser classificada como QUESTION, a mesma deve possuir um pronome interrogativo. Em uma *parse tree*, estes pronomes ficam nas sub-árvores com as *tags* WRB e WP, descritas no Quadro 1 do capítulo anterior. Então antes de varrer a árvore em busca do sujeito, verbo e complementos, é preciso verificar se a primeira sub-árvore possui a *tag* WRB ou WP.

O algoritmo de extração de triplos foi implementado conforme apresentado no Quadro 10. Inicialmente, o método `getTree` separa a *parse tree* em três possíveis sub-árvores (NP, VP, WH). Após, o tratamento de cada sub-árvore se dá de forma distinta, sendo que o sujeito da frase será obtido do nodo NP, o verbo e o complemento serão encontrados no nodo VP e o pronome interrogativo será encontrado no nodo WH. Então para encontrar o sujeito da frase

foi realizada uma busca em profundidade conforme mostra o método `extractSubject`, sendo que o sujeito será o primeiro substantivo encontrado. A extração do verbo se dá pelo método `extractPredicate`, que busca o verbo no nodo VP mais profundo. A extração dos complementos da frase também ocorre a partir da sub-árvore VP, através do método `extractObject`, que procura pelas sub-árvores PP, NP e ADJP.

Quadro 10 - Algoritmo de extração de triplos

```
 1  private void getTree(Parse parses){
 2    Parse[] children = parses.getChildren();
 3    for(int i = 0; i < children.length; i++){
 4      if("NP".equals(children[i].getType()) && NP_subtree == null){
 5        this.NP_subtree = children[i]; continue; }
 6      else if("VP".equals(children[i].getType())
 7              || "VBP".equals(children[i].getType()) && VP_subtree == null){
 8             this.VP_subtree = children[i]; continue; }
 9      else if(WH_subtree==null && children[i].getType().startsWith("WH")){
10             this.WH_subtree = children[i]; }
11      getTree(children[i]); }
12  }
13  private void extractSubject(Parse parse, Parse parent, Triple triple){
14    if(parse == null){ return; }
15    for(Parse children : parse.getChildren()){
16      extractSubject(children, parse, triple); }
17    if(triple.getSubject() == null && grammaticalClass.isNoun(parse.getType())){
18      triple.setSubject(parse.getCoveredText());
19      triple.setSubjectProperties(extractProperties(parse, parent)); return; }
20  }
21  private void extractPredicate(Parse parse, Parse parent, Triple triple){
22    if(parse == null){
23      throw newIllegalArgumentException
24                           (ResourceUtils.getMessage("NEED_MORE_SPECIFY")); }
25    if(parse.getChildCount() > 0){
26      if(parse.getChildCount() > 1 ||
27         !parse.getChildren()[0].getType().equals ("TK")){
28         for(Parse children : parse.getChildren()){
29           extractPredicate(children, parse, triple); }
30      }
31    }
32    if(grammaticalClass.isVerb(parse.getType())){
33      triple.setPredicate(parse.getCoveredText());
34      triple.setPredicateProperties(extractProperties(parse, parent));
35      if(triple.getPredicateProperties()!=null &&
36         triple.getPredicateProperties().size() > 0){
37         triple.setType(EnTypePhrase.REMOVE); }
38      else { triple.setType(EnTypePhrase.ASSERTION); }
39    }
40  }
41  private void extractObject(Parse parse, Triple triple){
42    if(triple.getObject() == null && "NP".equals(parse.getType()) ||
43       "PP".equals(parse.getType()) || "WHNP".equals(parse.getType())){
44       for(Parse children : parse.getChildren()){
45         extractObject(children,parse, "NP|PP", triple); }
46    }
47    if(triple.getObject() == null && "ADJP".equals(parse.getType())){
48      for (Parse children : parse.getChildren()){
49        extractObject(children,parse, "ADJP", triple); }
50    }
51    for(Parse children : parse.getChildren()){
52      extractObject(children, triple); }
53  }
```

Fonte: elaborado pelo autor.

Outro ponto importante é o treinamento do PLN, pois por melhor que seja o algoritmo de extração de triplos, caso a *parse tree* não seja gerada corretamente, o algoritmo acima também não funcionará. O treinamento deve envolver frases que cubram boa parte dos cenários possíveis para que se obtenha uma boa taxa de acertos. É um processo manual, onde deve-se informar um conjunto de árvores, conforme mostrado no Quadro 11 e explicado na seção 3.2.2. Como o *framework* Apache OpenNLP é livre de idioma, pode-se criar uma única implementação para diferentes idiomas, bastando realizar o treinamento para diferentes idiomas (linhas 1 e 15 do Quadro 11).

Quadro 11 - Treinamento Apache OpenNLP

```
 1 (TOP (S (NP (PRP I)) (VP (NP (VBP have)) (NP (CD a) (NN meeting)) (ADVP
 2 tomorrow)) ))
 3 (TOP (S (NP (PRP I)) (VP (NP (ADVP don't) (VBP need)) (PP (CD 2) (NN
 4 bananas))) ))
 5 (TOP (S (NP (PRP$ My) (NN mom)) (VP (VBZ is) (VP (VBN called) (NP (NNP
 6 Maria)))) ))
 7 (TOP (SBARQ (WHNP (WP Who)) (SQ (VP (NP (VBZ needs) (TO to) (NN buy)) (PP (NN
 8 bananas))))))
 9 (TOP (SBARQ (WHNP (WP What)) (SQ (VBP do) (NP (PRP I)) (VP (NP (VBP need))))
10 ))
11 (TOP (S (NP (PRP$ My) (NN mom)) (VP (NP (ADVP doesn't) (VBP need) (DT to) (NN
12 buy)) (PP (NN food))) ))
13 (TOP (SBARQ (WHADVP (WRB When)) (SQ (MD will) (NP (PRP I)) (VP (NP (VB have))
14 (PP (CD a) (ADJP math) (NN test)))) ))
15 (TOP (S (NP (PRP Eu)) (VP (VBP tenho)(NP (CD uma) (NN reunião))(ADVP amanhã))
16 ))
17 (TOP (S (NP (PRP Eu)) (VP (NP (VBP vou) (VBP ter)) (NP (CD uma) (NN reunião))
18 (ADVP amanhã)) ))
19 (TOP (S (NP (PRP Eu)) (VP (NP (VBP necessito) (DT de)) (PP (CD 2) (NNS
20 maçãs))) ))
21 (TOP (S (NP (PRP Eu)) (VP (NP (VBP tenho) (DT que) (NN ir) (DT pra)) (PP (NN
22 aula))) ))
23 (TOP (S (NP (PRP Eu)) (VP (NP (ADVP não) (VBP tenho)) (PP (CD três) (NNS
24 abacaxis))) ))
25 (TOP (SBARQ (WHNP (WP Que) (VBP horas)) (SQ (NP (PRP eu)) (VP (NP (VBP tenho)
26 (NN aula) (DT de)) (PP (NN lógica)))) ))
27 (TOP (SBARQ (WHNP (WP Onde)) (SQ (NP (PRP eu)) (VP (NP (VBP preciso) (NN ir))
28 (PP (NN hoje))) ))
29 (TOP (S (NP (PRP Eu)) (VP (NP (VBP tenho)) (ADVP (CD 10) (NN metros)) (PP (DT
30 de) (NN terra)))))
31 (TOP (S (NP (PRP$ Minha) (NN mãe)) (VP (NP (VBP tem) (NN aula)) (PP (NN hoje)
32 (DT à) (NN noite))) ))
33 (TOP (S (NP (DT A) (PRP$ palavra) (NN chiclete)) (VP (NP (VBP significa)) (PP
34 (NN chiclete))) ))
```

Fonte: elaborado pelo autor.

3.3.3 Compreensão das frases

A outra parte da complexidade deste projeto se dá na compreensão das sentenças ditas pelo usuário, que podem resultar na inserção, remoção ou consulta de informações no modelo de dados do usuário. Após a extração do sujeito, verbo e complementos da frase, é necessário extrair as classes das palavras da ontologia, para que seja possível inserir, remover ou consultar as informações do usuário. A ontologia deste projeto possui classes de palavras em

português e inglês e suas respectivas subclasses e superclasses, conforme pode ser visualizado no Quadro 12 nas linhas 9, 10, 14 e 15.

Quadro 12 - Ontologia da Ada

```
 1  <?xml version="1.0" encoding="windows-1252"?>
 2    <rdf:RDF
 3      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 4      xmlns:owl="http://www.w3.org/2002/07/owl#"
 5      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 6      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
 7      xml:base="http://www.rainha_vermelha.com.br/technologies/ontology">
 8    <owl:Class rdf:about="#food">
 9      <rdfs:label xml:lang="pt">Comida</rdfs:label>
10      <rdfs:label xml:lang="en">Food</rdfs:label>
11      <rdfs:subClassOf rdf:resource="#thing"/>
12    </owl:Class>
13    <owl:Class rdf:ID="hot_dog">
14      <rdfs:label xml:lang="pt">Cachorro quente</rdfs:label>
15      <rdfs:label xml:lang="en">Hot dog</rdfs:label>
16      <rdfs:subClassOf rdf:resource="#food"/>
17    </owl:Class>
18  </rdf:RDF>
```

Fonte: elaborado pelo autor.

Observa-se que normalmente as ontologias são criadas de forma estática, ou seja, durante o desenvolvimento são levantadas todas as classes, as subclasses e as superclasses de palavras que poderão ser utilizadas. No entanto, para este projeto, isto seria uma séria limitação, pois como o usuário pode falar praticamente qualquer tipo de informação, seria impossível levantar todas as palavras que poderiam ser utilizadas sem limitar as frases que seriam aceitas. Desta forma, foi proposta e desenvolvida uma ontologia dinâmica, ou seja, além de um conjunto de palavras que foram levantadas durante o desenvolvimento, o usuário pode informar qualquer palavra, permitindo assim que o mesmo ensine à Ada novas palavras e como elas se relacionam com o modelo de conhecimento já existente. Este processo é mostrado no Quadro 13, sendo que para um maior entendimento é dado o seguinte exemplo: supondo que a palavra “feijão” não exista no modelo de conhecimentos da Ada. Ao processar uma frase como “Feijão é uma comida.”, o assistente verifica que o processo é uma especificação, pois a palavra “feijão” não existe no modelo de conhecimento. Ao extrair as propriedades da frase, é obtido que o sujeito é “feijão”, o verbo é “ser” e o complemento é “comida”. Este triplo é processado pelo método `specification`, que registra a palavra “feijão” no modelo com sendo uma palavra derivada do Português e uma subclasse da palavra “comida”.

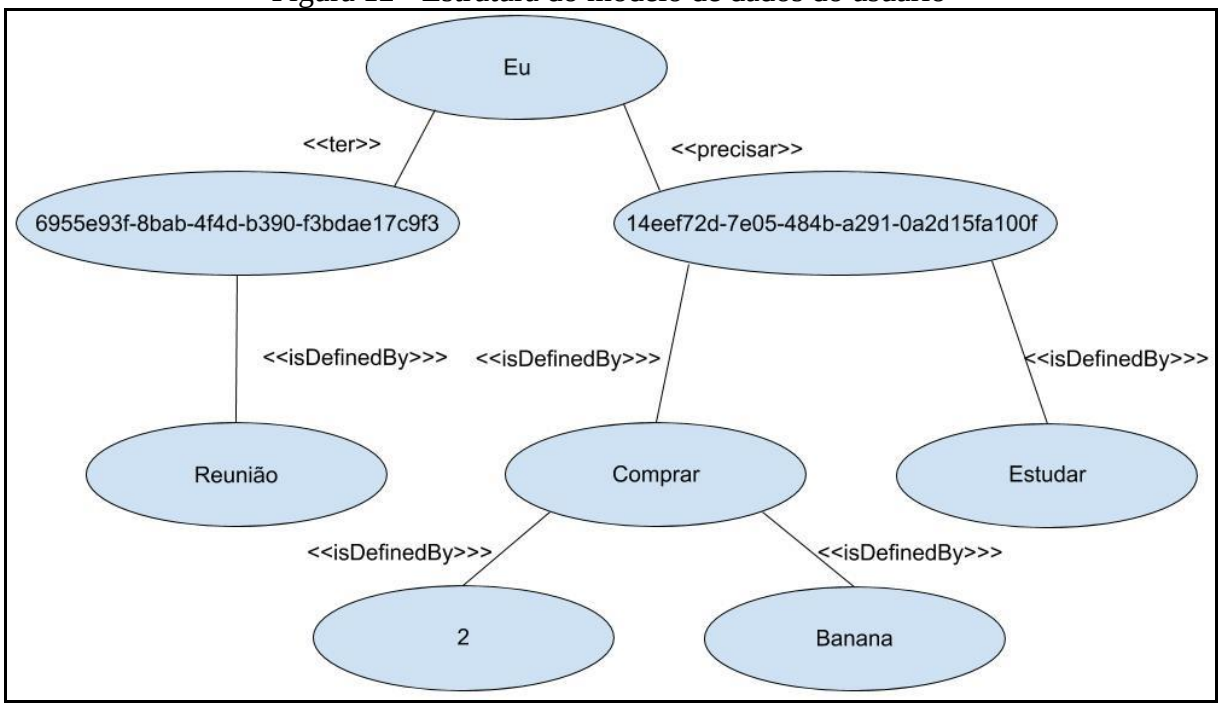
Quadro 13 - Especificação de novas palavras

```
 1  public static void specification(Triple triple, String key, String lang){
 2    if(triple != null){
 3      Dataset dataset = BODataSet.createInstance();
 4      dataset.begin(ReadWrite.WRITE);
 5      Model model = null;
 6      try {
 7        model = dataset.getNamedModel(key);
 8        model.setNsPrefix("rm", BOOntologie.BASE);
 9        model.add(triple.getSubject(), RDF.type, OWL.Class);
10        Resource resource = model.getResource(triple.getSubject().getURI());
11        resource.addProperty(RDFS.label,
12            triple.getSubject().getLocalName(),lang);
13        Resource object = (Resource) triple.getObject();
14        if (!triple.getSubject().getURI().equals(object.getURI())) {
15          Map<String, String> labels =
16            BOOntologie.getLabels(Arrays.asList(object.getURI()), key, lang);
17          if (labels == null || labels.size() == 0) {
18            resource.addProperty(RDFS.label, object.getLocalName(), lang);
19          } else {
20            resource.addProperty(RDFS.subClassOf, (Resource)
21  triple.getObject());
22          }
23        }
24        dataset.commit();
25      } catch (Exception e) {
26        dataset.abort();
27        throw new RuntimeException(e.getMessage());
28      } finally {
29        dataset.end();
30      }
31  }
```

Fonte: elaborado pelo autor.

A inserção é o processo onde uma frase do tipo ASSERTION é salva no modelo de dados do usuário. Antes de realizar a inserção, verifica-se se afirmação já não existe no modelo. Caso exista, o processo passa a ser uma complementação da informação. Por exemplo, considerando que o modelo contenha a informação que o usuário necessita comprar 1 banana. Caso o usuário informe que ele necessita de 2 bananas, o assistente determinará que o processo é uma complementação, pois no grafo do usuário já existe uma relação igual a afirmação que está sendo inserida. Assim, antes de inserir a informação, a relação antiga é removida e a quantidade é somada à nova quantidade informada de tal forma que o usuário passará a necessitar de 3 bananas. Caso a informação não exista no modelo, ela é inserida no grafo do usuário. Um grafo encontra-se exemplificado na Figura 12, onde é mostrado de que forma o conhecimento é estruturado no modelo de dados do usuário. Nessa figura é apresentado um grafo que contém três frases, sendo elas: “Eu tenho reunião.”, “Eu preciso estudar.” e “Eu preciso comprar 2 bananas.”. Conforme pode ser observado na Figura 12, o nodo “eu” se liga a dois outros nodos auxiliares por duas arestas “ter” e “precisar”. Estes nodos auxiliares são necessários para realizar o agrupamento dos complementos das frases.

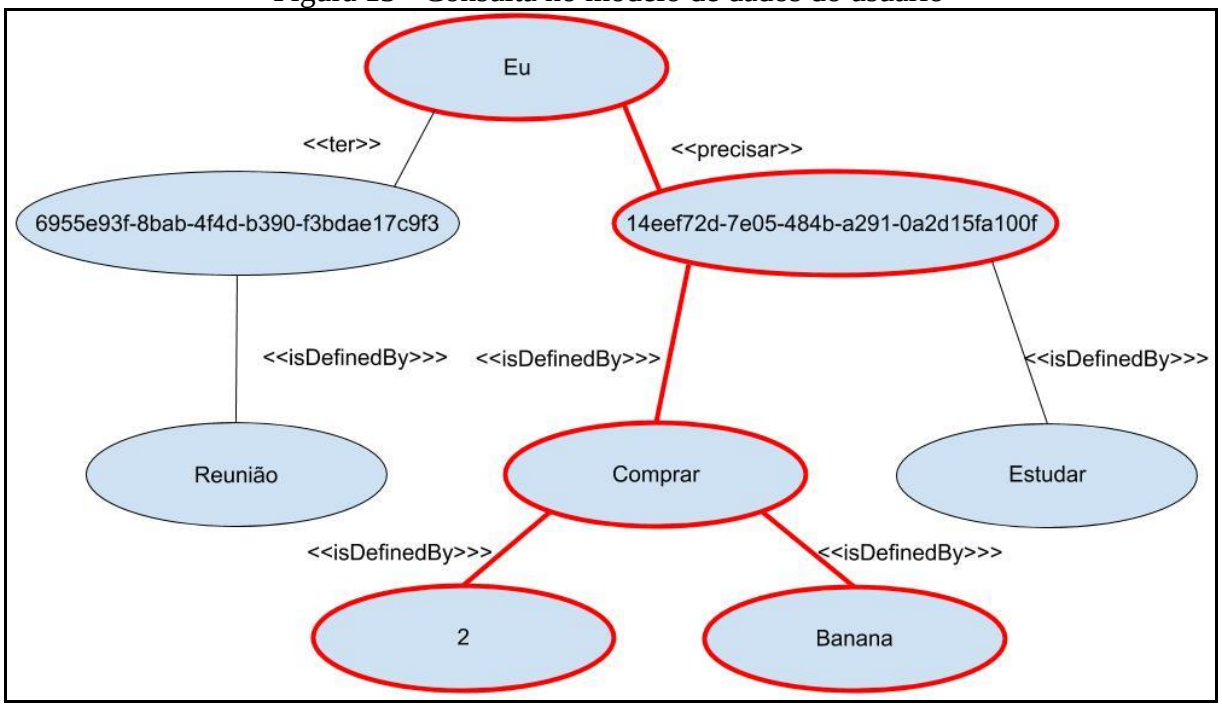
Figura 12 - Estrutura do modelo de dados do usuário



Fonte: elaborado pelo autor.

A inferência de dados é processo onde o usuário realiza uma pergunta para a Ada e ela retorna as informações encontradas, se existirem. Todas as perguntas que são compreendidas pelo assistente começam com um pronome ou advérbio interrogativo, sendo eles: que, quem, qual, quanto, quando, onde, como e porque (em português), *what*, *who*, *where*, *when*, *why*, *whose*, *how*, *how much* e *how many* (em inglês). Outras variações que também são aceitas em português são: que horas, que local e qual local, e em inglês: *what time*, *what place* e *which place*. A partir do pronome ou advérbio interrogativo, é descoberta a intenção da pergunta, ou seja, o que o usuário objetiva obter como resultado da sua pergunta. As intenções podem ser: tempo, lugar, modo, causa, coisa, pessoa, seletiva ou quantitativa. Para cada uma das intenções, a forma da consulta é diferente. Tendo como exemplo o modelo de dados da Figura 12, caso o usuário tivesse perguntado “O que eu preciso comprar?”, Ada identificaria que, por possuir o pronome interrogativo que, a intenção do usuário é buscar alguma coisa, representada pela classe *thing* da OWL. Assim, Ada responderia para o usuário que ele precisa comprar 2 bananas, conforme ilustrado na Figura 13.

Figura 13 - Consulta no modelo de dados do usuário



Fonte: elaborado pelo autor.

O usuário também pode realizar consultas utilizando-se de regras de dedução. Com isto, é possível melhorar a qualidade e variedade de suas inferências. A partir da regra de dedução do Quadro 14, o usuário pode realizar consultas utilizando superclasses das palavras. Utilizando como exemplo o modelo de dados da Figura 12, o usuário poderia fazer perguntas como “O que eu preciso fazer?”, obtendo como resposta, como ilustrado na Figura 14, que precisa comprar 2 bananas e estudar. Isso só é possível devido à regra de dedução mostrada no Quadro 14.

Quadro 14 - Regra de dedução da Ada

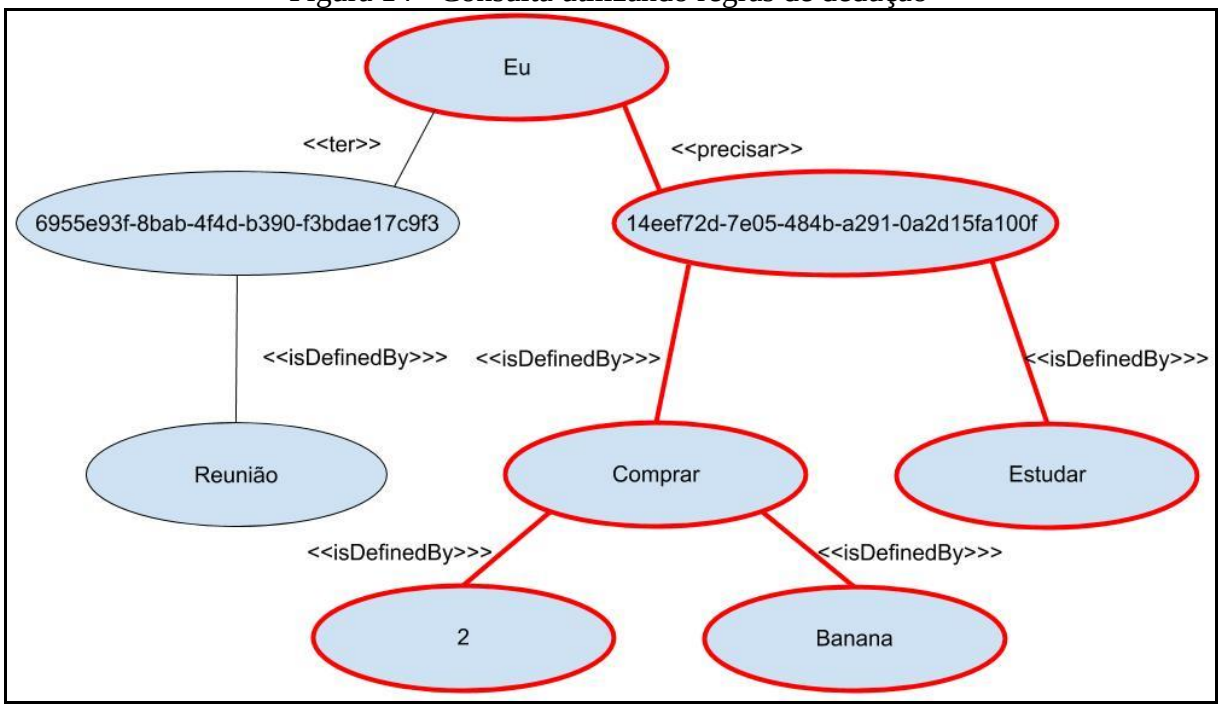
```
1  @prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#>.
2  #[rule1: (?a drc:p drc:b )  -> (?a drc:p drc:c)]
3  #[rule2: (?b drc:p drc:c )  -> (?b drc:p drc:d)]
4
5  [rule1:(?a rdfs:subClassOf ?b )(?b rdfs:subClassOf ?c ) -> (?a rdfs:subClassOf
6  ?c)]
```

Fonte: elaborado pelo autor.

O processo de remoção de informações do modelo de dados do usuário é semelhante às consultas. A remoção se dá quando a sentença for marcada como REMOVE. Para exemplificar uma remoção, considerando novamente o modelo de dados da Figura 12, o usuário poderia remover os dados lá contidos com as seguintes frases: “Eu não preciso estudar.” ou “Eu não tenho reunião.”. Ao contrário da inserção onde as frases podem ser complementadas, na remoção propriedades contidas no modelo são removidas. Para deixar mais claro, ainda utilizando o modelo de dados da Figura 12, caso o usuário diga a seguinte frase, “Eu não preciso comprar 1 banana.”, o assistente verificará que no modelo existe a

informação que o usuário necessita comprar 2 bananas, então do valor contido no modelo será subtraído 1, conforme a frase dita pelo o usuário.

Figura 14 - Consulta utilizando regras de dedução



Fonte: elaborado pelo autor.

3.3.4 Geração de frases

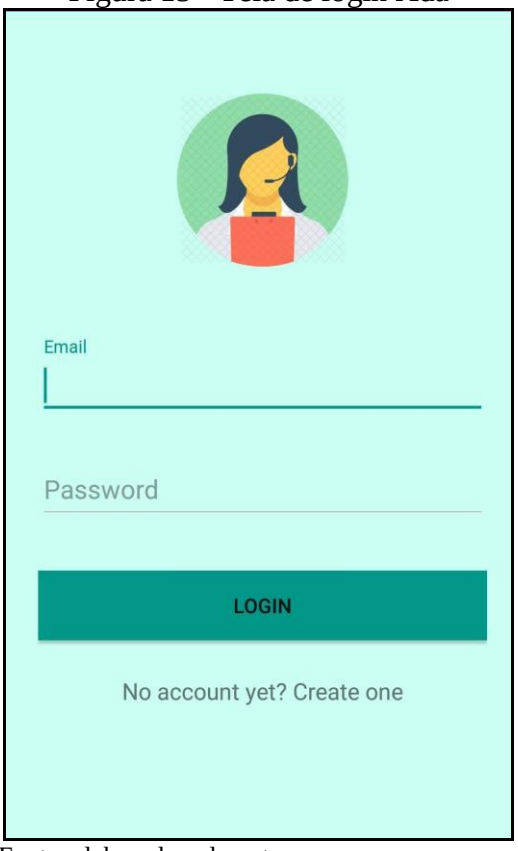
O processo de geração de frases ocorre quando o usuário realiza uma consulta sobre o seu modelo de dados e a Ada retorna as informações encontradas. Inicialmente, o modelo contendo as informações do usuário e a base de conhecimento interna da Ada com o significado das palavras e suas relações com outras palavras são carregados. Em seguida, são extraídas as propriedades, conforme descrito na seção anterior, assim obtendo as informações contidas na web semântica. A partir disso, é possível realizar o processo de construção de sentenças, ou seja, as propriedades extraídas são transformadas em sentenças. Para tanto, dada uma lista de triplos, é realizada uma iteração sobre a mesma para analisar cada estrutura. Caso na pergunta realizada pelo usuário o sujeito seja oculto, a frase a ser construída começará com o sujeito contido no triplo. Caso o verbo da inferência realizada pelo usuário seja a superclasse do verbo que foi retornado no triplo encontrado, ele é adicionado a sentença. Caso o verbo seja o mesmo do triplo encontrado ou já tenha sido adicionado à sentença, o mesmo não será adicionado novamente. Da mesma forma são processados os complementos. Por fim, como cada triplo representa uma frase, é necessário realizar a conjunção das frases, adicionando e ou *and*, dependendo do idioma da sentença que está sendo construída. Para tornar mais claro os passos descritos acima, usando a Figura 12 como exemplo, caso o usuário

pergunte “O que eu preciso fazer?”, ele terá a resposta “Estudar”. No entanto, caso o usuário pergunte “O que eu preciso?”, obterá a resposta “Comprar 2 bananas” e “Estudar”.

3.3.5 Operacionalidade da implementação

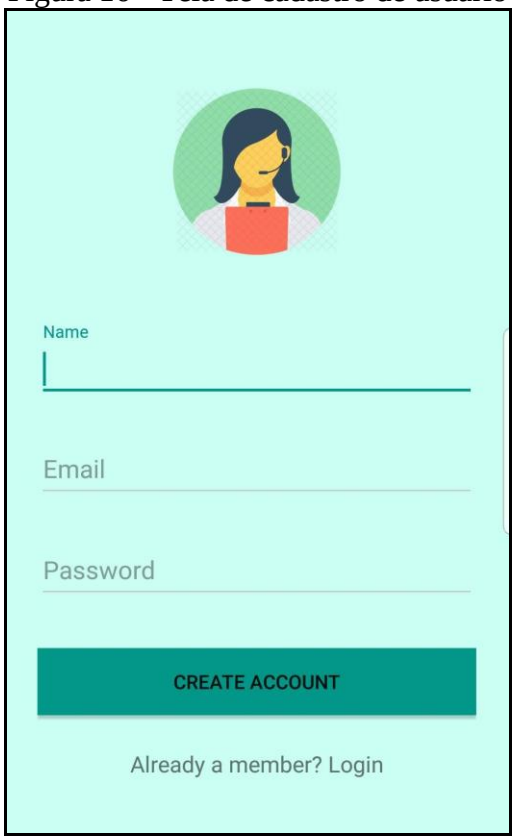
A Ada foi desenvolvida como um aplicativo móvel para a plataforma Android e deve ser instalado no smartphone para poder utilizá-lo. Também é necessária conexão com a internet para ser possível realizar a comunicação com o *backend*, que se encontra no AWS da Amazon, e com a API Cloud Speech to Text para realizar a conversão de fala em texto. A tela inicial do aplicativo é a tela de login, mostrada na Figura 15, onde o usuário pode realizar o login informando seu e-mail e senha. Caso o usuário não possua uma conta, ele pode se cadastrar ao clicar em `No account yet? Create one`. Na Figura 16 é apresentada a tela de cadastro de usuário. Para se cadastrar o usuário deve informar um e-mail ainda não cadastrado no assistente, seu nome e uma senha. Ao clicar no botão `CREATE ACCOUNT`, o usuário será redirecionado para a tela principal do aplicativo, mostrada na Figura 17.

Figura 15 - Tela de login Ada



Fonte: elaborado pelo autor.

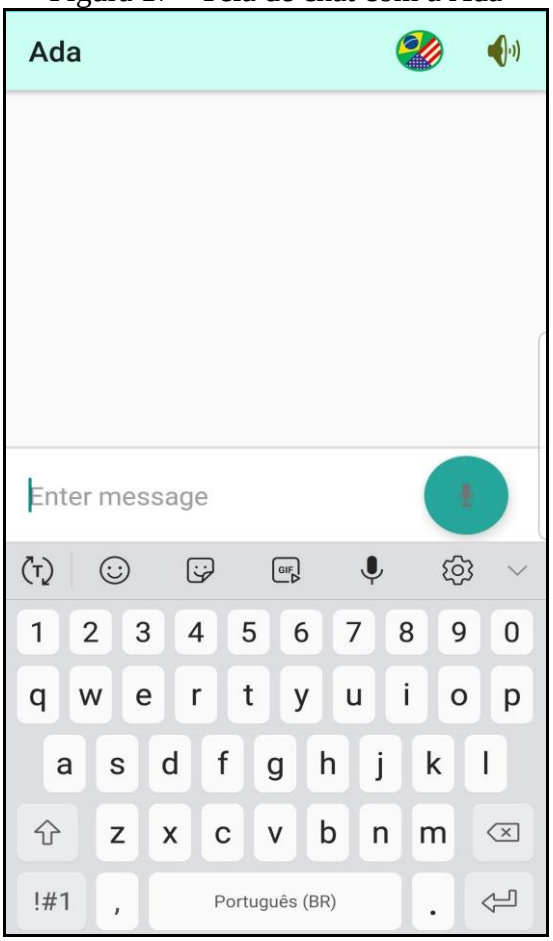
Figura 16 - Tela de cadastro de usuário



Fonte: elaborado pelo autor.

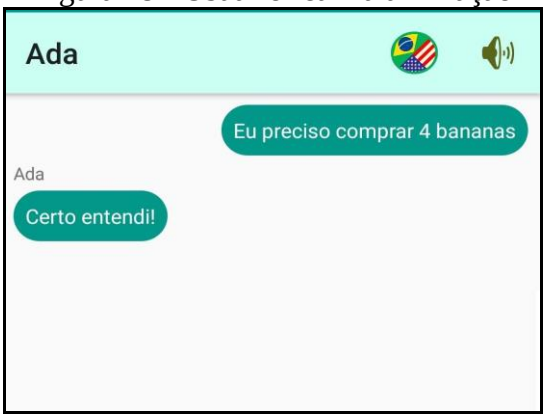
Na tela da Figura 17, o usuário poderá selecionar a forma de interação (por voz ou por texto) e dialogar com a Ada. Para selecionar a forma de interação, o usuário deve clicar no botão com ícone de alto-falante no canto superior direito, alternando entre interação por voz, como mostrado na figura, ou por texto. Ao falar ou digitar uma sentença sem erros ortográficos e gramaticais e clicar no botão com ícone de enviar, o assistente dará um feedback conforme mostrado na Figura 18. Na sequência, observa-se na Figura 19 que, apesar dos dados terem sidos armazenados em português, o usuário poderá consultar suas informações em inglês. Na Figura 20 é apresentado como é o processo para remover informações, enquanto na Figura 21 é mostrado como o aprendizado de novas palavras é realizado pela Ada.

Figura 17 - Tela de chat com a Ada



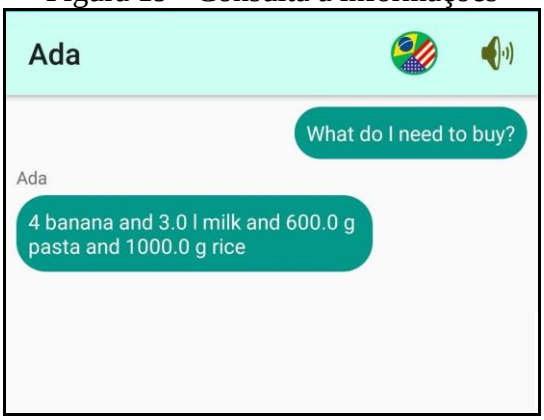
Fonte: elaborado pelo autor.

Figura 18 - Usuário realiza afirmação



Fonte: elaborado pelo autor.

Figura 19 - Consulta a informações

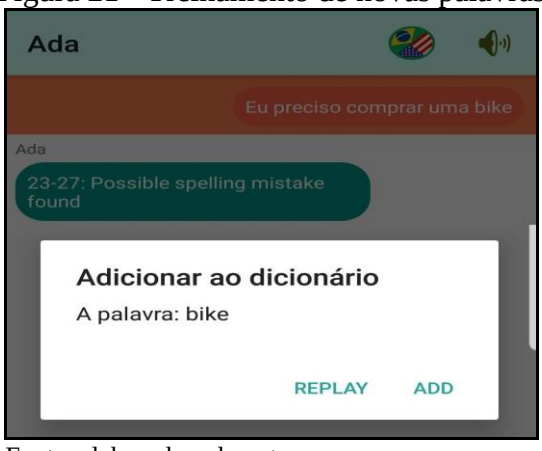


Fonte: elaborado pelo autor.

Figura 20 - Remoção de informações

Fonte: elaborado pelo autor.

Figura 21 - Treinamento de novas palavras



Fonte: elaborado pelo autor.

3.4 ANÁLISE DOS RESULTADOS

Nesta seção são mostrados os resultados obtidos com o desenvolvimento do assistente. Inicialmente é apresentada uma comparação entre as formas de realizar o reconhecimento de voz para aplicativos móveis pela *intent* ACTION_RECOGNIZE_SPEECH, disponibilizada nativamente pelo Android, e pela API Cloud Speech to Text, disponibilizada pela Google LLC. Em seguida, tem-se um comparativo entre os *frameworks* de PLN: Stanford Parser e

Apache OpenNLP. A última subseção traz uma comparação entre os trabalhos correlatos e o desenvolvido.

3.4.1 Comparativo entre as bibliotecas de reconhecimento de voz

A implementação da funcionalidade de reconhecimento de voz exigiu algumas alterações no decorrer do desenvolvimento do trabalho para que fosse possível realizar o reconhecimento de forma alternada entre frases ditas em Inglês e em Português. Inicialmente, o reconhecimento de voz foi implementado usando a *intent* ACTION_RECOGNIZE_SPEECH. No entanto, foi verificado que o reconhecimento somente funcionava no idioma em que o smartphone do usuário estivesse configurado. Assim, caso o usuário tentasse alternar o diálogo entre os idiomas inglês e português, o mesmo teria que trocar o idioma de seu smartphone. Por este motivo, optou-se por utilizar a API Cloud Speech to Text, que trabalha com múltiplos idiomas e demonstrou uma performance e um nível de confiabilidade semelhante à forma nativa, conforme mostrado no Quadro 15, onde são apresentadas sentenças em inglês e português que foram testadas em ambas as soluções. Todas as sentenças foram reconhecidas em no máximo duas tentativas.

Quadro 15 - Comparativo entre as bibliotecas de reconhecimento de voz

sentenças	API Cloud Speech to Text	*intent* ACTION_RECOGNIZE_ SPEECH
Eu preciso estudar hoje.	1ª tentativa	1ª tentativa
Eu tenho aula de matemática amanhã.	1ª tentativa	1ª tentativa
Eu preciso comprar 3 litros de leite.	1ª tentativa	1ª tentativa
Eu preciso limpar a casa esta semana.	1ª tentativa	1ª tentativa
I need to go out this weekend.	1ª tentativa	2ª tentativa
I have to buy two apples this week.	2ª tentativa	2ª tentativa
I need to study for the math test.	2ª tentativa	2ª tentativa
I have three books at home.	1ª tentativa	1ª tentativa
O que eu preciso fazer esta semana?	1ª tentativa	1ª tentativa
Quando eu tenho aula?	1ª tentativa	1ª tentativa
When do I need to buy?	1ª tentativa	1ª tentativa
What time do I need to go out?	1ª tentativa	1ª tentativa

Fonte: elaborado pelo autor.

3.4.2 Comparativo entre os *frameworks* de PLN

No início do desenvolvimento do aplicativo, buscou-se por *frameworks* de PLN que atendessem os requisitos funcionais e não funcionais do projeto, ou seja, o *framework* escolhido deveria poder ser utilizado em Java e ser capaz de trabalhar com sentenças em português e inglês. As informações obtidas do trabalho de Rusu et al. (2007) foram fundamentais para a escolha do *framework*. Rusu et al. (2007) apresentaram as métricas

obtidas da utilização do algoritmo de extração de triplos a partir de diferentes *frameworks*, sendo que no trabalho referenciado foi utilizada uma máquina com as seguintes configurações: CPU 2.80GHz e 2.00GB de RAM. Além disso, em todas as análises, foram utilizadas 100 sentenças extraídas de artigos da época. Assim, Rusu et al. (2007) registraram que com o Stanford Parser as 100 sentenças foram analisadas em 178.1 segundos gerando 118 triplos diferentes, enquanto que com o Apache OpenNLP as mesmas 100 sentenças foram analisadas em 29.95 segundos gerando um total de 168 diferentes triplos. Então, com base nos registros de Rusu et al. (2007), devido a maior performance na extração de triplos, optou-se por utilizar o Apache OpenNLP.

3.4.3 Comparativo entre os trabalhos correlatos

O Quadro 16 apresenta de forma comparativa as características dos trabalhos correlatos e da ferramenta desenvolvida.

Quadro 16 - Comparativo entre os trabalhos correlatos e o assistente desenvolvido

características / trabalhos	Dragon Mobile Assistant (NUANCE COMMUNICATIONS INC., 2017)	Braina (BRAINASOFT, 2017)	Hound Voice Search & Mobile Assistant (SOUNDHOUND INC., 2017)	Ada (trabalho desenvolvido)
entrada via comando de voz	x	x	x	x
entrada via texto	x	x	x	x
capacidade de aprendizado		x		x
automatização de tarefas	x		x	
gerenciamento das atividades		x		x
idioma de interação	inglês	inglês	inglês	inglês português
capacidade de responder a perguntas externas	x	x	x	
chatterbot	x	x	x	

Fonte: elaborado pelo autor.

Conforme pode ser observado no Quadro 16, todos os aplicativos analisados permitem que a interação seja tanto por comandos de voz quanto por texto, sendo que aplicativos que não possuem esta característica estão restritos a lugares em que o usuário julgar que não afetem sua privacidade. Todos os aplicativos analisados possuem uma boa taxa de acerto em reconhecer as entradas do usuário por comandos de voz e, em casos de erros, o usuário ainda possui a possibilidade de interagir por comandos de texto.

A capacidade de aprendizado está relacionada com a possibilidade do software aprender coisas a respeito do usuário e do mundo a sua volta. Como o Braina trabalha com a ideia de *templates*, somente pode aprender coisas com relação a seus *templates*, o que é limitado. A Ada, por sua vez, por representar os conhecimentos do mundo em uma OWL, é capaz de extrair e adicionar novas relações à sua base de conhecimento. Neste processo realizado pela Ada, o usuário pode inferir, por exemplo, que a palavra "refri" significa refrigerante e que é uma bebida. Assim, apesar da palavra "refri" não existir no português, pode ser aprendida pelo assistente, enquanto os outros aplicativos não possuem essa característica.

A automatização de tarefas realizada pelo Dragon Mobile Assistant e pelo Hound Voice Search & Mobile Assistant é muito semelhante, executando de forma autônoma tarefas agendadas ou de forma explícita tarefas como navegar para determinada rota. Observa-se que a automatização de tarefas é o ponto chave dos aplicativos Dragon Mobile Assistant e Hound Voice Search & Mobile Assistant, embora não permitam o gerenciamento de atividades. Os únicos aplicativos que permitem o usuário consultar os seus compromissos agendados são a Braina e a Ada. Por outro lado, os dois primeiros facilitam a utilização dos aplicativos do smartphone do usuário, tais como realizar ligações e enviar e-mails. Observa-se também que dentre os assistentes digitais descritos, somente a Ada funciona em dois idiomas, enquanto os demais interagem somente no idioma Inglês.

A capacidade de responder perguntas externas, é uma característica importante. Além de tornar os diálogos mais interessantes, é capaz de suprir problemas do processamento das sentenças proferidas pelo o usuário, pois ao não conseguir realizar um processamento, ainda existe a possibilidade de buscar na internet uma possível para o mesmo, sendo que somente a Ada não possui esta característica. Ter a funcionalidade de *chatterbot* garante, além de um maior engajamento do usuário, que ele permanece por mais tempo utilizando o assistente, sendo que também somente a Ada não possui esta característica.

No Quadro 17 foi estabelecida uma comparação entre as ferramentas quanto à funcionalidade de compreensão de sentenças ditas pelo usuário. É possível constatar que os aplicativos vistos possuem dificuldades na interpretação das sentenças afirmativas, o que pode inviabilizar uma interação ou no mínimo gerar um retrabalho por parte do usuário em tentar adaptar uma sentença de uma forma que o assistente a compreenda. Observa-se que somente a Ada foi capaz de compreender todas as afirmações, mas, por outro lado, os demais aplicativos obtiveram melhores resultados em compreender as perguntas realizadas. Isto se deve à capacidade de responder perguntas externas.

Quadro 17 - Comparativo entre o poder de compreensão de sentenças

sentenças \ trabalhos	Dragon Mobile Assistant (NUANCE COMMUNICATIONS INC., 2017)	Braina (BRAINASOFT, 2017)	Hound Voice Search & Mobile Assistant (SOUNDHOUND INC., 2017)	Ada (trabalho desenvolvido)
I need to go to school this week	sim	não	não	sim
I have to buy 2 shirts at mall	sim	não	não	sim
I need to study for math	não	sim	sim	sim
I have a meeting tomorrow	sim	não	não	sim
My mom has to go home today	não	não	não	sim
My father has to buy food	não	não	não	sim
I don't want to receive calls	sim	não	não	sim
I don't have money	não	sim	não	sim
I want to eat	sim	não	sim	sim
Who was the last usa presidente?	sim	sim	não	não
How much cust a banana?	não	não	não	não
What is the current temperature?	sim	sim	sim	não
Where can i get a haircut?	sim	sim	sim	não
How do I get to Golden Stare bridge?	sim	não	sim	não

Fonte: elaborado pelo autor.

4 CONCLUSÕES

Durante o desenvolvimento desse trabalho, criou-se um assistente pessoal automatizado móvel que interage com o usuário nos idiomas Português e Inglês. Ada permite o usuário informar atividades, necessidades e lembretes da forma mais natural possível, assim como também permite realizar consultas sobre as informações salvas. Todo diálogo é realizado no idioma de preferência do usuário, ou seja, se o usuário iniciar um diálogo em Português, Ada responderá em Português, caso seja em Inglês, ela responderá em Inglês. A ontologia foi desenvolvida de tal forma que o usuário possa informar a classe das palavras que ele está falando e não são inicialmente compreendidas pela Ada. Com isso, tem-se a capacidade de aprendizado de maneira que o uso de novas palavras incrementa o conhecimento do assistente. Portanto, todos os objetivos elencados foram atingidos.

Durante a escolha das ferramentas para o desenvolvimento deste trabalho, existiram alguns percalços principalmente referente à escolha do *framework* PLN e da ferramenta de reconhecimento de voz. Referente ao *framework* Apache OpenNLP, a escolha deu-se a partir dos resultados apresentados por Rusu et al. (2007). Ainda assim, foi necessário, inicialmente, encontrar os arquivos de treinamento do *parser* para a Língua Portuguesa e, posteriormente, após o treinamento, adicionar cada nova sentença proferida pelo usuário no modelo de treinamento. No entanto, em função do Apache OpenNLP ser um *parser* estatístico, com o aumento do modelo de treinamento, a taxa de acertos na geração das *parse trees* foi aumentando consideravelmente ao longo do desenvolvimento. Quanto ao reconhecimento de voz, inicialmente, foi testada a utilização da *intent* CHECK_TTS_DATA. Como durante na fase de testes foi verificado que o reconhecimento somente funcionava para o idioma que estava configurado o smartphone do usuário, assim impossibilitando o usuário de alternar os idiomas durante o diálogo, optou-se por usar a API Cloud Speech to Text. Ainda, pensou-se em utilizar o *framework* SimpleNLG para o assistente gerar e fornecer respostas mais complexas e de maior qualidade ao usuário. No entanto, foram encontrados diversos problemas na utilização desse *framework* para a Língua Portuguesa, principalmente no que diz respeito à geração de artigos e preposições para as frases. Optou-se então por não utilizar a geração de linguagem natural, mas sim por gerar as frases de respostas ao usuário conforme descrito na seção de geração de frases.

Ao final do desenvolvimento do projeto, acredita-se que este possa contribuir para a área de processamento de linguagem natural, visto que os trabalhos sobre PLN pesquisados fora desenvolvidos focados em um único idioma, ou seja, para se trabalhar com mais de um

idioma é necessário realizar uma nova implementação, enquanto que para a Ada somente é necessário criar os arquivos de treinamento referentes ao idioma que se deseja trabalhar. Outra contribuição trata-se da extensão do algoritmo de extração de triplos apresentado por Rusu et al. (2007). O algoritmo proposto por eles extrai sujeito, verbo e complementos de frases afirmativas ou negativas. A partir deste trabalho também é possível extrair, além do sujeito, verbo e complementos, o pronome interrogativo da frase interrogativas. Por fim, diferentemente dos trabalhos pesquisados onde as ontologias sempre eram estáticas, neste trabalho foi desenvolvimento uma ontologia dinâmica que é capaz de aprender novos termos e palavras e suas relações com os demais conhecimentos já armazenados, assim resolvendo a limitação de não ser possível levantar todas as palavras (e suas relações) que podem ser ditas por um usuário.

4.1 EXTENSÕES

O projeto apresentado nesta monografia atingiu os objetivos propostos. Entretanto, existem pontos que podem ser melhorados e novas funcionalidades que podem ser implementadas. São eles:

a) buscar formas de otimizar a performance das consultas na web semântica;
b) utilizar algoritmos de geração de linguagem natural para gerar as frases de retorno ao usuário;
c) implementar a capacidade de respostas externas utilizando web semântica;
d) incorporar *chatterbot* ao assistente para ter um maior grau de engajamento com o usuário;
e) incorporar a capacidade de automação de tarefas;
f) desenvolver diferentes tipos de clientes, tais como páginas web e smartwatch, para o assistente estar disponível em outras plataformas;
g) realizar testes do assistente em ambientes específicos como consultórios médicos e escritórios.

REFERÊNCIAS

BECHHOFER, Sean. **OWL reasoning examples**. [S.l.], 2003. Disponível em: <http://owl.man.ac.uk/2003/why/latest/>. Acesso em: 16 maio 2018.

BRAINASOFT. **Braina**: artificial intelligence (AI) virtual assistant software. [S.l.], 2017. Disponível em: <https://www.brainasoft.com/braina/>. Acesso em: 22 jun. 2017.

BURLESON, Cody. **What is the semantic web?** [S.l.], 2014. Disponível em: <https://www.quora.com/What-is-the-Semantic-Web>. Acesso em: 16 maio 2018.

CAMBRIDGE SEMANTICS. **Introduction to the semantic web**. [S.l.], 2017. Disponível em: <http://www-stage.cambridgesemantics.com/semantic-university/introduction-semantic-web>. Acesso em: 01 mar. 2017.

CORREIA, Ana; NUNES, Sérgio. **Semantic web**. Porto, 2002. Disponível em: <https://web.fe.up.pt/~mgi01016/ari/mcr_final.pdf>. Acesso em: 16 maio 2018.

ESPINHA, Roberto. **Gerenciamento de atividades**: ganhe tempo organizando suas atividades no dia a dia. [S.l.], 2016. Disponível em: <http://artia.com/blog/gerencie-melhor-seu-tempo/>. Acesso em: 28 out. 2017.

IWASAWA, Shun'ya et al. A Collaborative annotation between human annotators and a statistical parser. In: LINGUISTIC ANNOTATION WORKSHOP, 5th, 2011, Portland. **Proceedings...** Stroudsburg: Association for Computational Linguistics, 2011. p. 56-64. Disponível em: <https://www.semanticscholar.org/paper/A-Collaborative-Annotation-between-Human-Annotators-Iwasawa-Hanaoka/837c98f7eba03e83c2855add58a3732e15fbfddf/figure/2>. Acesso em: 13 maio 2018.

JONES, Tim. **Speaking out loud**: an introduction to natural language processing. [S.l.], 2017. Disponível em: <https://www.ibm.com/developerworks/library/cc-cognitive-natural-language-processing/index.html>. Acesso em: 17 maio 2018.

JURAFSKY, Dan; MARTIN, James. **Speech and language processing**. 3th ed. California: Stanford University Press, 2017.

KISER, Matt. **Introduction to natural language processing (NLP)**. [S.l.], 2016. Disponível em: <https://blog.algorithmia.com/introduction-natural-language-processing-nlp/>. Acesso em: 10 abr. 2017.

LESKOVEC, Jurij; GROBELNIK, Marko; MILIC-FRAYLING, Natasa. Impact of linguistic analysis on the semantic graph coverage and learning of document extracts. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 25th, 2005, Pennsylvania. **Proceedings...** [S.l.]: Association for the Advancement of Artificial Intelligence, 2005. Não paginado. Disponível em: <https://www-cs.stanford.edu/people/jure/pubs/nlpspo-aaai05.pdf>. Acesso em: 17 maio 2018.

LIBERMAN, Mark. **Introduction to linguistics.** [S.l.], 2003. Disponível em: <http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html>. Acesso em: 20 maio 2018.

MANNING, Christopher; SCHÜTZE, Hinrich. **Foundations of statistical natural language**. 8th ed. United Kindown: MIT Press, 2002.

NUANCE COMMUNICATIONS INC. **Dragon Mobile Assistant**: your personal assistant for life. [S.l.], 2017. Disponível em: <http://www.dragonmobileapps.com/>. Acesso em: 22 jun. 2017.

ORME, Anthony M.; YAO, Haining; ETZKORN, Letha H. Coupling metrics for ontology-based systems. **IEEE Software**, Alabama, v. 23, n. 2, p. 102-108, Mar./Apr. 2006. Disponível em: <https://www.computer.org/csdl/mags/so/2006/02/s2102-abs.html>. Acesso em: 17 maio 2018.

OUKSEL, Aris M.; SHETH, Amit. Semantic interoperability in global information systems: a brief introduction to the research area and the special section. **ACM SIGMOD Record**, New York, v. 28, n. 1, p. 5-12, Mar. 1999. Disponível em: <https://dl.acm.org/citation.cfm?id=309849>. Acesso em: 14 jun. 2018.

PREDICTIVE ANALYTICS TODAY. **Top 22 intelligent personal assistants or automated personal assistants**. [S.l.], 2017. Disponível em: <https://www.predictiveanalyticstoday.com/top-intelligent-personal-assistants-automated-personal-assistants/>. Acesso em: 05 jul. 2017.

ROUSE, Margaret. **Digital assistant**. [S.l.], 2014. Disponível em: <http://whatis.techtarget.com/definition/digital-assistant>. Acesso em: 28 out. 2017.

RUSU, Delia et al. Triplet extraction from sentences. In: CONFERENCE ON DATA MINING AND DATA WAREHOUSES, 6th, 2007, Ljubljana. **Proceedings...** Ljubljana : Jozef Stefan Institute, 2007. Não paginado. Disponível em: <http://ailab.ijs.si/dunja/SiKDD2007/Papers/Rusu_Trippels.pdf>. Acesso em: 27 mar. 2017.

SAKSAMUDRE Suman K.; SHRISHRIMAL, Pukhraj P.; DESHMUKH, Ratnadeep R. A Review on different approaches for speech recognition system. **International Journal of Computer Applications**, [S.l.], v. 115, n. 2, p. 23-28, Apr. 2015. Disponível em: <https://www.researchgate.net/publication/276136270_A_Review_on_Different_Approaches_for_Speech_Recognition_System>. Acesso em: 14 jun. 2018.

SOUNDHOUND INC. **Hound Voice Search & Mobile Assistant**. Santa Clara, 2017. Disponível em: <https://www.soundhound.com/hound>. Acesso em: 22 de jun. 2017.

SPEECH-to-Text software. In: TECHOPEDIA. [S.l.]: Techopedia Inc., 2018. Disponível em: <https://www.techopedia.com/definition/23767/speech-to-text-software>. Acesso em: 14 maio 2018.

STANFORD NLP GROUP. **The Stanford parser**: a statistical parser. Stanford, 2018. Disponível em: <https://nlp.stanford.edu/software/lex-parser.shtml>. Acesso em: 17 maio 2018.

TEXT to Speech (TTS). In: TECHOPEDIA. [S.l.]: Techopedia Inc., 2018. Disponível em: <https://www.techopedia.com/definition/23843/text-to-speech-tts>. Acesso em: 14 maio 2018.

TRACTICA LLC. **The virtual digital assistant market will reach $15.8 billion worldwide by 2021**. Colorado, 2016. Disponível em: <https://www.tractica.com/newsroom/press-releases/the-virtual-digital-assistant-market-will-reach-15-8-billion-worldwide-by-2021/>. Acesso em: 02 set. 2017.

TUTORIALS POINT. **OpenNLP tutorial**. [S.l.], 2018. Disponível em: <https://www.tutorialspoint.com/opennlp/index.htm>. Acesso em: 01 jun. 2018.

W3C BRASIL. **Web semântica**. [S.l.], 2011. Disponível em: <http://www.w3c.br/Padroes/WebSemantica>. Acesso em: 14 out. 2017.

APÊNDICE A – Detalhamento dos casos de uso

A seguir é apresentado o detalhamento dos casos de uso: `UC01: Registrar usuário` (Quadro 18), `UC02: Definir forma da interação` (Quadro 19), `UC03: Cadastrar atividade` (Quadro 20), `UC04: Consultar atividade` (Quadro 21), `UC05: Remover atividade` (Quadro 22) e `UC06: Alertar próximas atividades` (Quadro 23).

Quadro 18 - Caso de uso `UC01: Registrar usuário`

UC01 – Registrar usuário	
Descrição	Permite que o usuário crie uma conta para utilizar o assistente pessoal automatizado, denominado Ada.
Ator	Usuário
Cenário principal	1. O usuário informa um e-mail e uma senha. 2. O assistente cria uma conta específica para o usuário. 3. O usuário acessa suas informações a partir do e-mail e da senha informados.
Pré-condições	O e-mail informado deve ser único.
Pós-condições	Conta específica para o usuário registrada no assistente.

Fonte: elaborado pelo autor.

Quadro 19 - Caso de uso `UC02: Definir forma da interação`

UC06 – Definir forma da interação	
Descrição	Permite que o usuário defina se o diálogo a ser realizado com o assistente pessoal automatizado será por voz ou texto.
Ator	Usuário
Cenário principal	1. O usuário seleciona a opção para emitir som. 2. O assistente passa a interagir por voz.
Fluxo alternativo 01	Após o passo 2 do fluxo principal: 1. O usuário desseleciona a opção para emitir som. 2. O assistente passa a interagir somente por texto.
Pré-condições	O usuário estar registrado, conforme UC01.
Pós-condições	Interação por som selecionada (ou desselecionada, conforme o caso). Próximas respostas do assistente por voz (ou por texto).

Fonte: elaborado pelo autor.

Quadro 20 - Caso de uso `UC03: Cadastrar atividade`

UC03 – Cadastrar atividade	
Descrição	Permite que o usuário registre atividades.
Ator	Usuário
Cenário principal	1. O usuário informa uma atividade. 2. O assistente verifica se a atividade não está cadastrada. 3. O assistente cadastra a atividade. 4. O assistente informa que cadastrou uma nova atividade.
Fluxo alternativo 01	No passo 2 do fluxo principal, se a atividade informada já está cadastrada: 1. O assistente complementa as informações da atividade. 2. O assistente informa que complementou as informações da atividade já cadastrada.
Pré-condições	O usuário deve estar registrado, conforme UC01. O usuário deve informar a atividade em uma frase contendo sujeito, verbo e complementos, nessa ordem. A frase não deve conter erros ortográficos e gramaticais.
Pós-condições	Atividade cadastrada ou informações da atividade já cadastrada complementadas, conforme o caso.

Fonte: elaborado pelo autor.

Quadro 21 - Caso de uso `UC04: Consultar atividade`

UC04 – Consultar atividade	
Descrição	Permite que o usuário consulte suas atividades.
Ator	Usuário
Cenário principal	1. O usuário informa termos específicos que possam identificar uma atividade, tais como data, hora ou palavras-chave (reunião, aula, congresso, compras, entre outros). 2. O assistente verifica se existe alguma atividade específica cadastrada. 3. O assistente informa a atividade encontrada.
Fluxo alternativo 01	No passo 2 do fluxo principal, se a atividade informada não está cadastrada como uma atividade específica: 1. O assistente verifica se a atividade não é a superclasse de outras atividades. 2. O assistente informa as atividades encontradas.
Pré-condições	O usuário deve estar registrado, conforme UC01. O usuário deve informar a atividade em uma frase contendo sujeito, verbo e complementos, nessa ordem. A frase não deve conter erros ortográficos e gramaticais. A atividade deve estar cadastrada, conforme UC02.
Pós-condições	Informações da atividade apresentadas.

Fonte: elaborado pelo autor.

Quadro 22 - Caso de uso `UC05: Remover atividade`

UC05 – Remover atividade	
Descrição	Permite que o usuário remova atividades cadastradas.
Ator	Usuário
Cenário principal	1. O usuário informa a atividade a ser removida. 2. O assistente verifica se a atividade está cadastrada como uma atividade específica. 3. O assistente remove a atividade. 4. O assistente informa que removeu a atividade.
Fluxo alternativo 01	No passo 2 do fluxo principal, se a atividade informada não está cadastrada como uma atividade específica: 1. O assistente verifica se a atividade não é a superclasse de outras atividades. 2. O assistente remove as atividades encontradas.
Pré-condições	O usuário deve estar registrado, conforme UC01. O usuário deve informar a atividade em uma frase contendo sujeito, verbo e complementos, nessa ordem. A frase não deve conter erros ortográficos e gramaticais. A atividade deve estar cadastrada, conforme UC02.
Pós-condições	Atividade removida.

Fonte: elaborado pelo autor.

Quadro 23 - Caso de uso `UC06: Alertar próximas atividades`

UC06 – Alertar próximas atividades	
Descrição	Alerta o usuário sobre atividades a serem realizadas no dia de hoje.
Ator	Assistente
Cenário principal	1. O assistente verifica se o usuário possui atividades registradas para o dia de hoje. 2. O assistente gera uma notificação no celular do usuário informando as atividades do dia. 3. O usuário visualiza as atividades do dia.
Pré-condições	O usuário deve estar registrado, conforme UC01. As atividades devem estar cadastradas, conforme UC02.
Pós-condições	Atividades do dia apresentadas.

Fonte: elaborado pelo autor.