UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE TESTE DE PROFICIÊNCIA EM LÍNGUA ALEMÃ

CARLOS HENRIQUE STAPAIT JUNIOR

BLUMENAU
2017

CARLOS HENRIQUE STAPAIT JUNIOR

PROTÓTIPO DE TESTE DE PROFICIÊNCIA EM LÍNGUA ALEMÃ

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.

Prof. Roberto Heinzle, Doutor - Orientador

BLUMENAU
2017

PROTÓTIPO DE TESTE DE PROFICIÊNCIA EM LÍNGUA ALEMÃ

Por

CARLOS HENRIQUE STAPAIT JUNIOR

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:	Prof. Roberto Heinzle, Doutor – Orientador, FURB
Membro:	Prof(a). Grabriele Jennrich Bambineti, Especialista – FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB

Blumenau, 12 de dezembro de 2017

Dedico este trabalho de conclusão de curso a minha família, amigos e colegas, que me incentivaram e apoiaram, sem isso não seria possível a concretização deste trabalho.

AGRADECIMENTOS

À minha família...

Aos meus amigos e colegas...

A todos os professores do curso por me ensinarem tudo o que sei hoje.

À professora Joyce que me orientou na primeira etapa do pré-projeto.

Ao meu orientador Roberto Heinzle por ter acreditado em mim.

O seu tempo é limitado, então não o gaste vivendo a vida de um outro alguém. Não fique preso pelos dogmas, que é viver com os resultados da vida de outras pessoas. Não deixe que o barulho da opinião dos outros cale a sua própria voz interior.

Steve Jobs

RESUMO

Este trabalho descreve o desenvolvimento do protótipo de teste de proficiência em língua alemã. É um programa desktop desenvolvido em Linguagem Java com a finalidade de testar os conhecimentos de uma pessoa na língua alemã. Para automatizar o teste de proficiência acrescentou-se ao protótipo recursos como o processamento da linguagem natural. Para isso foram utilizados a biblioteca Mary Text To Speech (TTS) na versão 5.2 e Sphinx versão 4-5prealpha. A ferramenta envolvida no armazenamento de dados foi a API Java Architecture for XML Binding (JAXB), presente no Java Development Kit (JDK). O protótipo foi validado através de sua utilização por professores e alunos estudantes do idioma alemão. Os relatos dos alunos atestam que o protótipo atende as expectativas de avaliar os conhecimentos de um estudante do idioma alemão. No entanto, mostra-se necessário realizar estudos mais aprofundados para fazer o protótipo atender ao requisito de testar a pronúncia de uma pessoa no idioma alemão.

Palavras-chave: Teste de proficiência. Teste de nivelamento. Processamento de linguagem natural.

ABSTRACT

This work describes the development of the Proficiency Test Prototype of German Language. It is a desktop program developed in Java Language with the purpose of testing a person’s knowledge in the German language. To automate the proficiency test, the prototype resources like as natural language processing, were added. For this we used the Mary Text To Speech (TTS) library in version 5.2 and Sphinx version 4-5prealpha. The tool involved in storing data was the Java Architecture for XML Binding (JAXB) API, present in the Java Development Kit (JDK). The prototype was validated through its use by German language teachers and students. The Student reports attest that the prototype meets the expectations of assessing a German language student's knowledge. However, further studies are needed to make the prototype meet the requirement to test a person's pronunciation in the German language.

Key-words: Proficiency test. Placement test. Natural language processing.

LISTA DE FIGURAS

Figura 1 – Teste *online* 14
Figura 2 – Questão de gramática do teste 15
Figura 3 – Questão de interpretação de texto do teste 16
Figura 4 – Questão auditiva do teste 16
Figura 5 – Resultado do teste 17
Figura 6 – Teste *online* 18
Figura 7 – Questões de interpretação de texto do teste 18
Figura 8 – Questão de compreensão auditiva do teste 19
Figura 9 – Questão de interpretação de texto do teste 20
Figura 10 – Resultado do teste 21
Figura 11 – Exemplo de diagrama de bloco de um sistema TTS 22
Figura 12 – Diagrama de Pipeline de Reconhecimento de voz. 22
Figura 13 – Diagrama de caso de uso 25
Figura 14 – Diagrama e classes de estrutura dos dados 27
Figura 15 – Diagrama de classes do teste de proficiência 29
Figura 16 – Trecho de código de seleção e ordenação de perguntas 30
Figura 17 – Trecho de código montando teste de pronúncia 31
Figura 18 – Diagrama de classes do relatório 32
Figura 19 – Trecho de código contagem de respostas 33
Figura 20 – Trecho de código geração de relatório 34
Figura 21 – Método que gera a voz sintetizada 36
Figura 22 – Método para reconhecimento de voz 37
Figura 23 – Anotações em Java 38
Figura 24 – método marshall 39
Figura 25 – método unmarshall 40
Figura 26 – Estrutura XML 40
Figura 27 – Janela de Login 41
Figura 28 – Menu da Janela principal do protótipo 41
Figura 29 – Tela de cadastro de níveis 42
Figura 30 – Tela de cadastro de habilidades 43
Figura 31 – Tela de cadastro de perguntas 45

Figura 32 – Janela de consulta.....46
Figura 33 – Inserção do nome 47
Figura 34 – Mensagem 47
Figura 35 – Teste de proficiência 48
Figura 36 – Janela de relatório 49
Figura 37 – Perguntas respondidas erradas na janela de relatório.....50
Figura 38 – Teste de pronúncia 51
Figura 39 – Janela do relatório da pronúncia 51
Figura 40 – Descrição dos níveis do CEFR.....57
Figura 41 – trecho de código 59
Figura 42 – trecho de código da gravação do áudio 60

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

ASR – Automated Speech Recognition

CEFR – Common European Framework of Reference for Languages

HTML – Hyper Text Markup Language

JAXB – Java Architecture for XML Binding

JDK – Java Development Kit

RF – Requisito Funcional

RNF – Requisito Não Funcional

TTS – Text To Speech

UC – Casos de Uso

UML – Unified Modeling Language

XML – eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVOS 13
1.2 ESTRUTURA 13
2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 TESTE DE NIVELAMENTO DA DEUTSCHE WELLE 14
2.2 TESTE DE PROFICIÊNCIA DO GOETHE INSTITUT 17
2.3 O PROCESSAMENTO DA LINGUAGEM NATURAL 21
3 DESENVOLVIMENTO 24
3.1 REQUISITOS 24
3.2 ESPECIFICAÇÃO 24
3.2.1 Caso de uso 25
3.2.2 Diagramas de classes 26
3.3 IMPLEMENTAÇÃO 35
3.3.1 Técnicas e ferramentas utilizadas 35
3.3.2 O processamento da Linguagem Natural 35
3.3.3 A Estrutura dos dados 37
3.3.4 Operacionalidade da implementação 41
3.4 ANÁLISE DOS RESULTADOS 52
3.4.1 Experimento de Validação 52
3.4.2 Comparativo com Trabalhos Correlatos e Considerações Finais 52
4 CONCLUSÕES 54
4.1 EXTENSÕES 54
REFERÊNCIAS 56
APÊNDICE A – QUADRO EUROPEU COMUM DE REFERÊNCIA PARA LÍNGUAS 57
APÊNDICE B – NIVELAMENTO 59
APÊNDICE C – CAPTURA DE ÁUDIO PELO MICROFONE 60

1 INTRODUÇÃO

No aprendizado de idiomas, segundo Post (2016), todo estudante precisa desenvolver quatro habilidades: ouvir, falar, ler e escrever. Para exercitar estas habilidades, tem-se atividades de interpretação de texto, dissertação, compreensão auditiva e conversação. As habilidades precisam ser exercitadas continuamente e o esforço empregado depende do nível de conhecimento do estudante.

Antes de se matricular num curso de idiomas, normalmente é feito o teste de proficiência ou nivelamento que, de acordo com o FURB Idiomas (2016, p. 1), "[...] avalia os conhecimentos do idioma e é fundamental para saber em qual nível se está apto a cursar. É recomendado para quem já teve contato com o idioma a ser estudado [...]". Portanto, o objetivo deste teste é medir os conhecimentos do estudante e classificar em níveis usando alguma avaliação de referência[1].

No FURB Idiomas o teste de nivelamento "é realizado de forma autônoma nos computadores do Laboratório de Línguas" (FURB IDIOMAS, 2016, p. 1), mas está disponível apenas para a língua inglesa. A avaliação de proficiência em outras línguas é feita por um professor através de testes escritos ou orais. Estes testes de nivelamento autônomos, que também são oferecidos *online* por instituições similares ao FURB Idiomas, são constituídos, na sua maioria, apenas por questões objetivas de leitura e interpretação de texto e não avaliam todas as habilidades que o estudante de línguas precisa ter para definir seu nível de conhecimento em determinado idioma. Assim, pode-se considerar que as provas escritas ou orais, efetuadas nas próprias instituições, ainda são a melhor alternativa para quem deseja um certificado formal em alguma língua, uma vez que nos testes *online* não há garantias de que o participante não realizou nenhum tipo de consulta.

Mas existem recursos, como o processamento de linguagem natural, que podem melhorar o modo como os testes autônomos são feitos. A interpretação de texto pode ser avaliada com uma questão de múltipla escolha, por exemplo. A escrita e o vocabulário podem ser avaliados a partir de palavras isoladas ou, mais sofisticadamente, por um corretor ortográfico e gramatical para analisar o relacionamento entre as palavras de um texto. Já atividades de compreensão auditiva, que dependem de filmes, músicas e áudios, podem

[1] O Common European Framework of Reference for Languages (CEFR) é o modelo mais usado. Segundo Council of Europa (2014), ele foi projetado para fornecer uma base para o ensino, aprendizagem e avaliação de línguas. O quadro classifica a proficiência em seis níveis: A1, A2, B1, B2, C1 e C2.

também ser realizadas com síntese de voz, enquanto a pronúncia pode ser avaliada usando o reconhecimento de voz.

Diante do exposto, propõe-se estender o uso dos computadores na área de ensino de idiomas, mais especificamente, em testes de nivelamento, acrescentando o processamento de linguagem natural, através de síntese e reconhecimento de voz. Buscar-se-á avaliar as habilidades de compreensão auditiva e pronúncia na proficiência de língua alemã.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver um protótipo que realize um teste de proficiência em língua alemã.

Os objetivos específicos do protótipo são:

a) avaliar o conhecimento no idioma alemão quanto a gramática;
b) averiguar a habilidade de leitura através da interpretação de texto;
c) verificar a compreensão auditiva usando a síntese de voz;
d) avaliar a pronúncia com o uso de reconhecimento de voz.

1.2 ESTRUTURA

Esta monografia está dividida em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusão. O primeiro capítulo, a introdução apresenta o tema e a justificativa, seguida pelos objetivos e a estrutura da monografia. O segundo capítulo detalha a fundamentação teórica do teste de língua alemã e os trabalhos correlatos. O terceiro capítulo é destinado a mostrar o desenvolvimento do protótipo, apresentando a especificação, ferramentas utilizadas e resultados obtidos. O quarto e último capítulo apresenta as conclusões e sugestões para extensão do protótipo.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentados dois trabalhos correlatos e introduzidos os conceitos do processamento da linguagem natural. Os trabalhos correlatos são testes de nivelamento *online* oferecidos por duas instituições conhecidas pelo rico material disponibilizado para o ensino e divulgação da língua alemã (DEUTSCHE WELLE, 2015; GOETHE INSTITUT, 2016), as metodologias que usam para avaliar as capacidades linguística dos seus usuários assemelham-se com o protótipo aqui proposto. O processamento da linguagem natural são recursos computacionais que simulam ou interpretam a voz humana e serão usadas junto do protótipo.

2.1 TESTE DE NIVELAMENTO DA DEUTSCHE WELLE

A Deutsche Welle oferece um site com muitos recursos para o ensino da língua alemã, inclusive *podcasts*, mas o objeto desse estudo é o teste de nivelamento *online*. A Figura 1 mostra a página inicial do teste da Deutsche Welle, com *links* informativos que o antecedem, bem como os níveis que o teste engloba.

Figura 1 – Teste *online*

Fonte: Deutsche Welle (2015).

Antes de começar o teste, o site pede o registro do participante, informando nome completo, ano de nascimento, língua materna, nacionalidade e e-mail. Em seguida é solicitada a escolha do nível do teste. Existem quatro opções, conforme pode ser observado na Figura 1. Cada nível é composto por trinta questões, sendo que cada questão pode ter uma ou mais perguntas objetivas. O nível do teste escolhido determina a complexidade das questões que o participante terá que responder. Alguém que detenha conhecimentos mais elevados no idioma terá mais facilidade para interpretar e responder as questões do que um iniciante. Os níveis de complexidade do teste correspondem aos níveis do CEFR até o B2.

É após a escolha do nível que o teste efetivamente começa. O teste é constituído por questões de leitura, compreensão auditiva, vocabulário e gramática. A Figura 2 apresenta uma questão de gramática do nível B1. O participante deve preencher quatro campos com os pronomes interrogativos citados no final da questão. Isso não significa que os estudantes de alemão tenham que aprender estes pronomes interrogativos apenas no nível B1, isso significa que estes pronomes devem fazer parte do vocabulário de nível B1.

Figura 2 – Questão de gramática do teste

Fonte: Deutsche Welle (2015).

A Figura 3 traz um exemplo de questão de interpretação de texto também do nível B1. O texto que deve ser interpretado está à esquerda, enquanto à direita são feitas duas perguntas.

Figura 3 – Questão de interpretação de texto do teste

Fonte: Deutsche Welle (2015).

A Figura 4 mostra uma questão de compreensão auditiva também do nível B1. Uma característica dos testes auditivos da Deutsche Welle é que os áudios possuem um som ambiente e não somente as vozes dos interlocutores. Isso obriga o participante a imergir no diálogo para entendê-lo. Além disso, mesmo nos testes dos níveis A1 ou A2, os diálogos têm sempre a velocidade normal da fala cotidiana. Observa-se que os áudios só podem ser reproduzidos três vezes.

Figura 4 – Questão auditiva do teste

Fonte: Deutsche Welle (2015).

Percebe-se que as instruções das questões apresentadas nas figuras estão todas em alemão, obrigando o participante a interpretar o que está escrito. No entanto, nos testes de complexidade dos níveis A1 e A2 as instruções contam com uma tradução para o inglês. As questões cujas respostas devem ser informadas em campos de texto só aceitam palavras chaves, ou seja, não há teste dissertativo. Além disso, em nenhum momento foi preciso usar o microfone, isto é, o teste não avalia a pronúncia.

Na última página do teste (Figura 5) são mostradas quantas perguntas foram respondidas e em quanto tempo o teste foi concluído. Se o participante conseguir mais de 80% das respostas corretas será encorajado a experimentar o próximo nível.

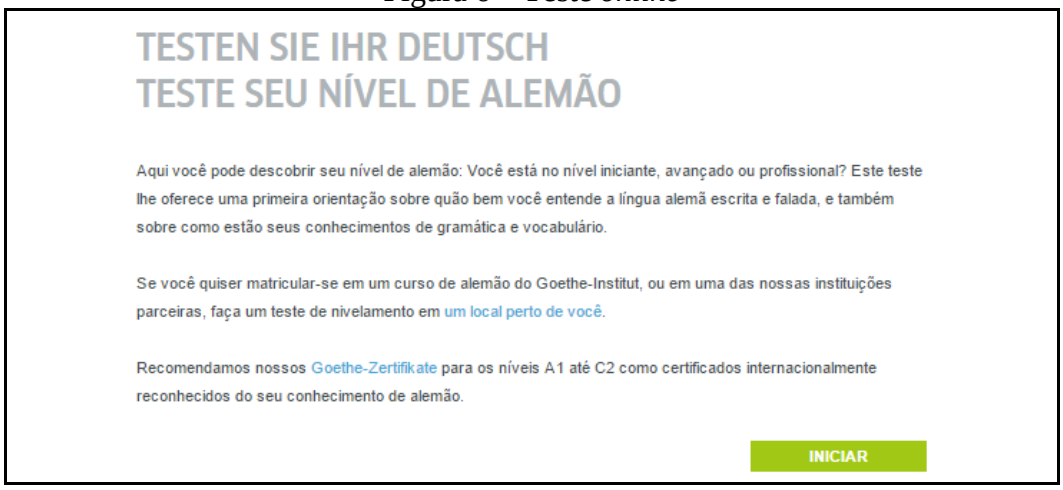
Figura 5 – Resultado do teste

Fonte: Deutsche Welle (2015).

2.2 TESTE DE PROFICIÊNCIA DO GOETHE INSTITUT

O Goethe Institut oferece conteúdo *online* similar ao da Deutsche Welle. O teste de proficiência, por sua vez, é diferente: é formado por doze questões e trinta perguntas. Diferente do teste da Deutsche Welle, o participante não seleciona o nível de competência para o qual deseja verificar seus conhecimentos, é o teste que determina o nível do participante pelas respostas corretas. A Figura 6 mostra a página inicial do teste do Goethe Institut com as orientações disponíveis em vários idiomas.

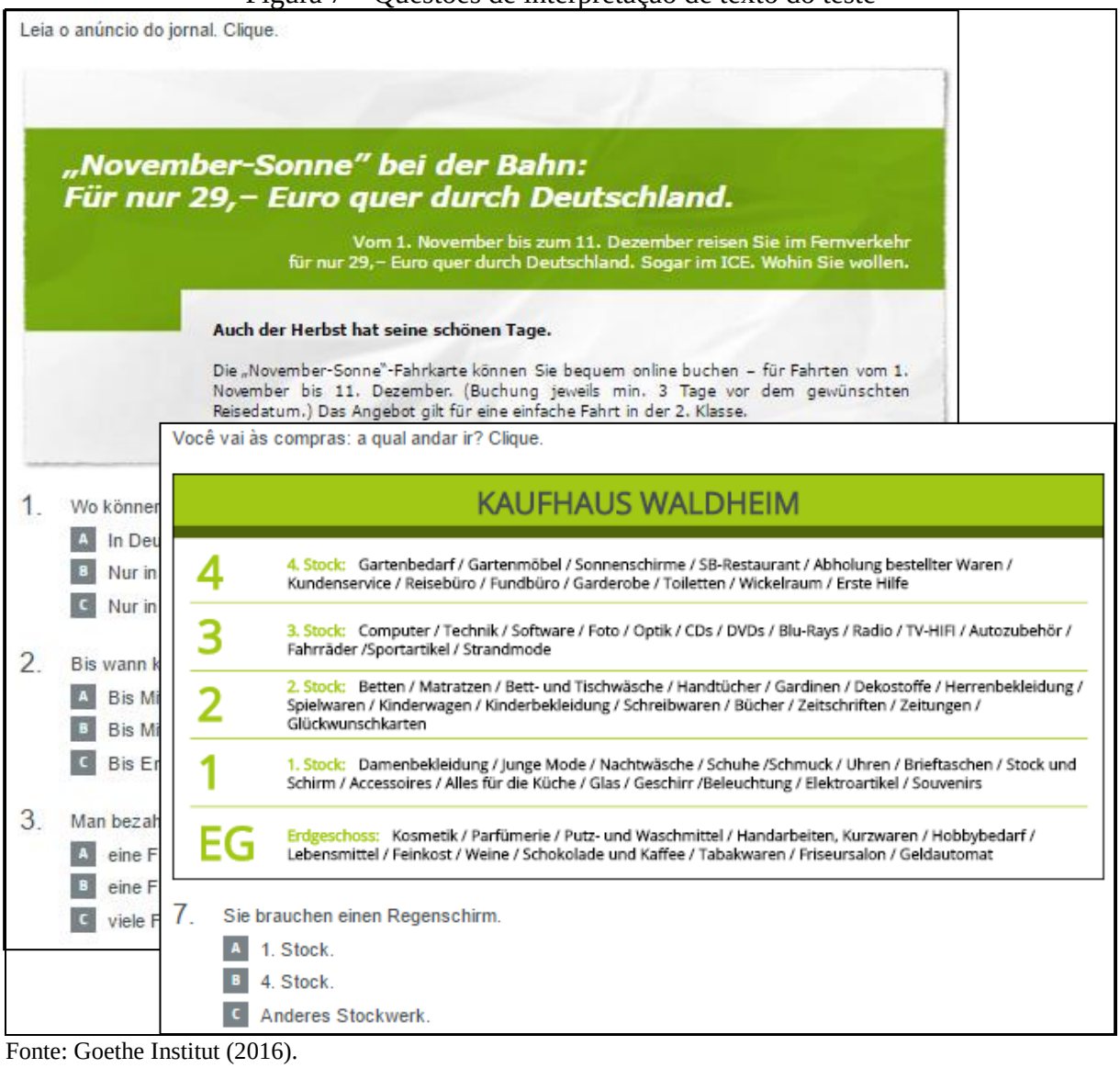
Figura 6 – Teste *online*



Fonte: Goethe Institut (2016).

O teste do Goethe Institut avalia a competência do participante principalmente pela interpretação de texto e compreensão auditiva.

Figura 7 – Questões de interpretação de texto do teste

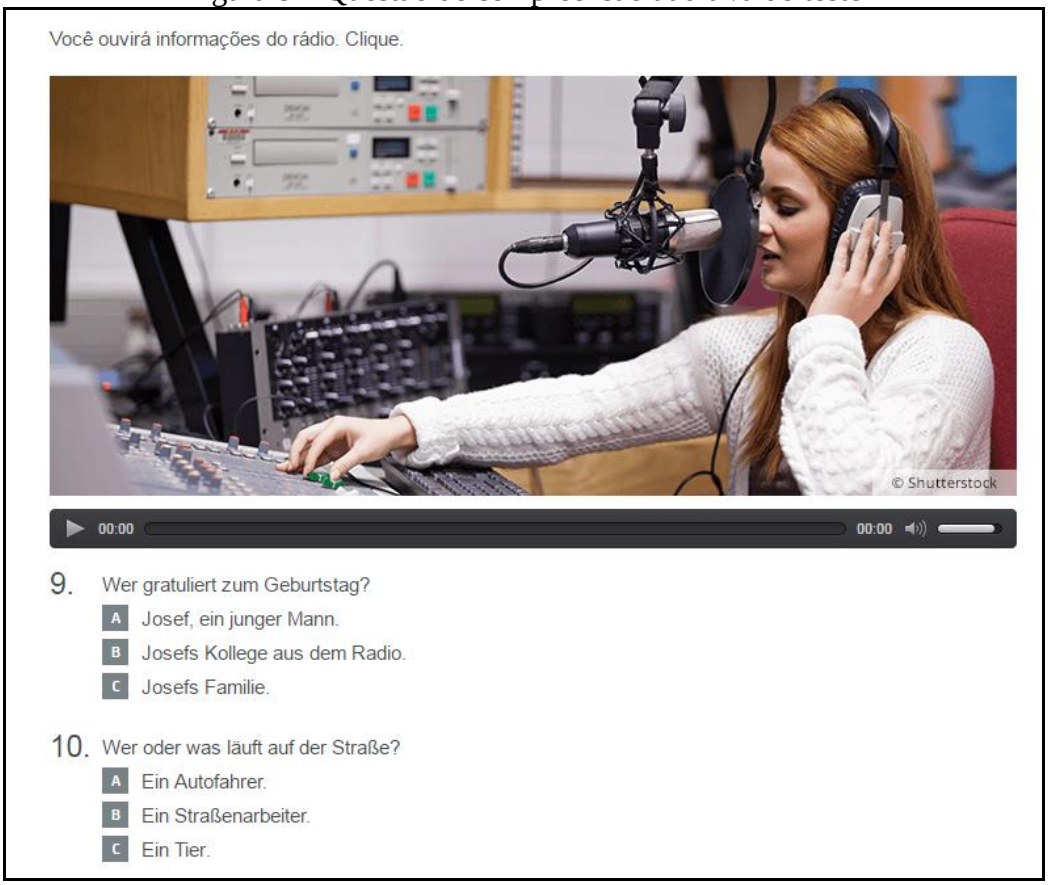


Fonte: Goethe Institut (2016).

A Figura 7 mostra duas questões de interpretação de texto. Na primeira, à esquerda, a página começa com uma breve instrução em português: “Leia o anúncio do jornal. Clique.”. Abaixo da imagem que representa o anúncio do jornal, estão três perguntas todas em alemão. Já a questão apresentada na Figura 7 à direita faz alusão a uma placa informativa de uma loja com cinco andares, sendo que o participante deve selecionar em quais andares estão os objetos que ele precisa comprar.

Percebe-se que as questões apresentadas nos testes normalmente fazem alusão a algumas situações reais enfrentadas no dia a dia. A Figura 8 apresenta uma questão de compreensão auditiva do teste do Goethe Institut. A principal diferença é que o áudio pode ser reproduzido quantas vezes o participante desejar.

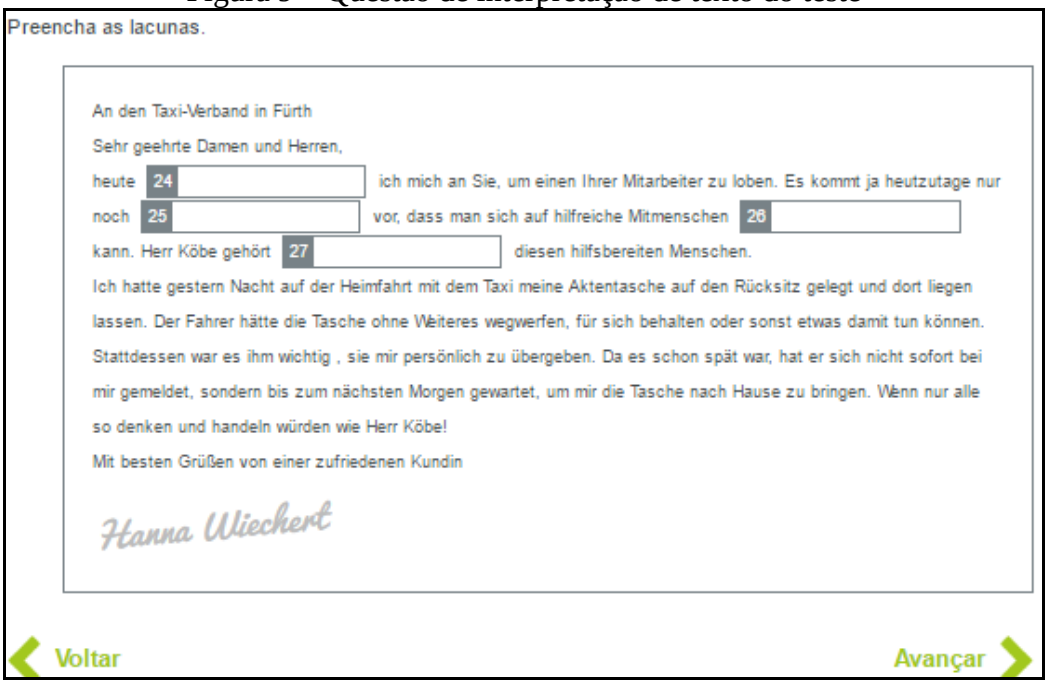
Figura 8 – Questão de compreensão auditiva do teste



Fonte: Goethe Institut (2016).

A Figura 9 apresenta uma questão de interpretação de texto com gramática e vocabulário de um nível um pouco mais avançado. O texto é uma carta e não há nenhuma informação de quais palavras devem ser colocadas nos campos. O teste da Deutsche Welle também possui algumas questões parecidas com este modelo, que foram omitidas na seção anterior por se assemelhar a esta questão. Observa-se que o teste do Goethe Institut possui menos questões que o teste da Deutsche Welle, tornando-o menos extenso.

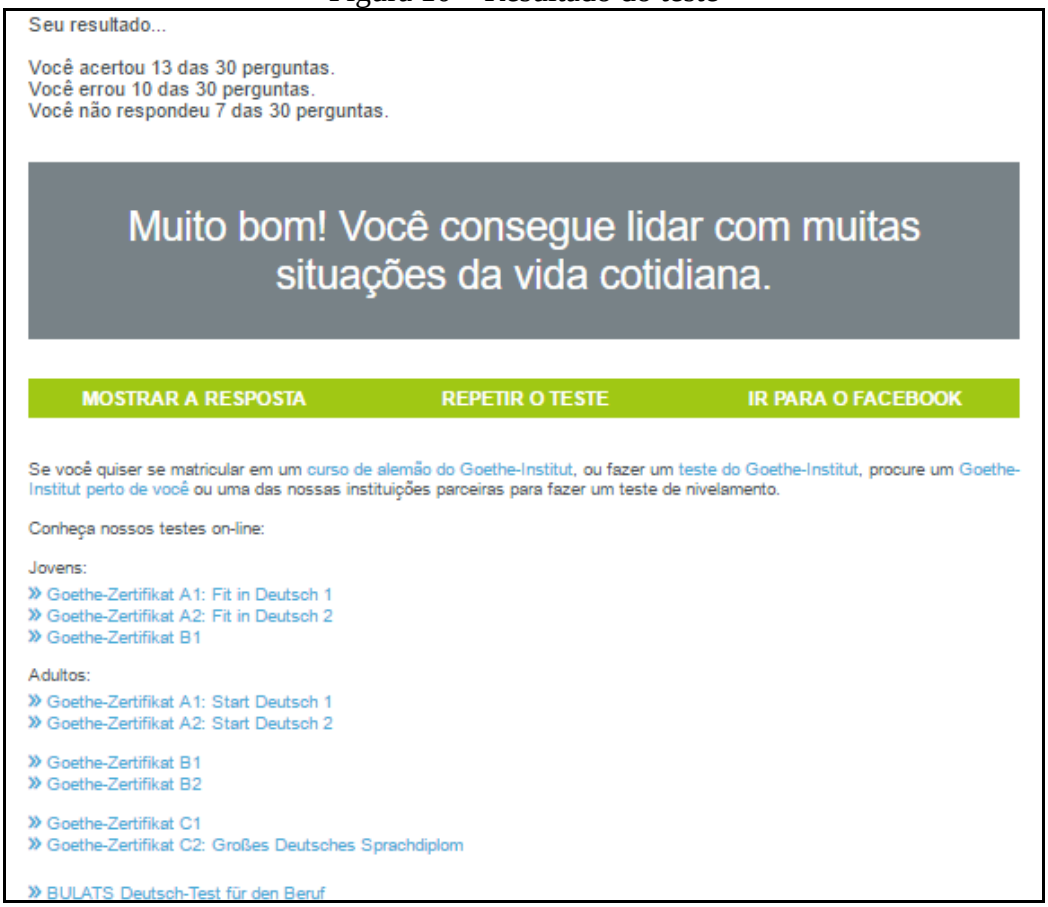
Figura 9 – Questão de interpretação de texto do teste



Fonte: Goethe Institut (2016).

Por fim, na Figura 10 é mostrado o resultado do teste com a quantidade de perguntas que foram corretamente respondidas, quantas estavam erradas e quantas foram deixadas sem resposta. O resultado do teste é apresentado como: “Muito bom! Você consegue lidar com muitas situações da vida cotidiana.”, mas não é informado em qual nível do CEFR o participante está. A Universidade de Coimbra (2014, p. 1) descreve que um usuário de nível B1 “É capaz de lidar com a maioria das situações encontradas na região onde se fala a língua-alvo.”. Portanto, pode-se sugerir que o resultado apresentado na Figura 10 classifica o participante com conhecimentos do nível B1.

Figura 10 – Resultado do teste



Fonte: Goethe Institut (2016).

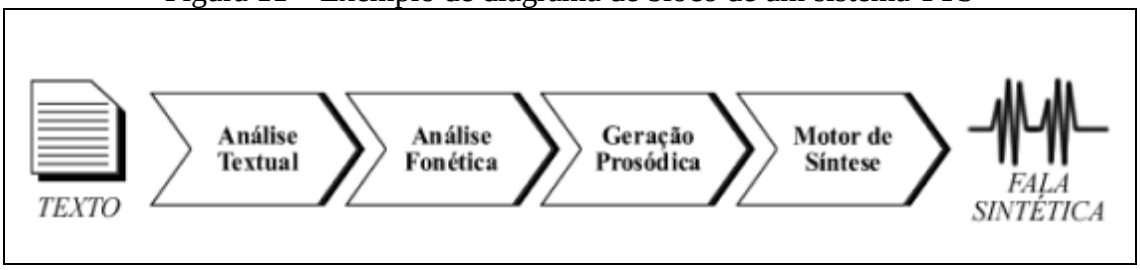
2.3 O PROCESSAMENTO DA LINGUAGEM NATURAL

Oliveira et al. (2011) afirmam que o processamento de linguagem natural pode fazer uso de duas tecnologias computacionais: a síntese de voz ou Text-To-Speech (TTS), que consiste em converter texto em voz sintetizada, e reconhecimento de voz ou Automated Speech Recognition (ASR), que converte o sinal de voz digitalizado em texto.

Costa et al (2012) descrevem que um sistema TTS é comumente composto por duas partes: *Front-End* constituído por módulos de processamento da linguagem natural e *Back-End* constituído por módulos de processamento de voz para a geração de voz sintetizada.

O *Front-End* é formado por um conjunto de algoritmos que devem normatizar um texto (KINOSHITA et al. (2006) apud COSTA et al. (2012)). Ele transforma o texto em uma sequência de fonemas e assim determina as características prosódicas da língua. (COSTA et al, 2012). *Back-End* possui um conjunto de filtros ou reconhecimento de padrões que recebem parâmetros amostrais de voz e os rótulos de contexto prosódico, para formar a onda sonora correspondente a pronúncia do texto. (COSTA et al., 2012).

Figura 11 – Exemplo de diagrama de bloco de um sistema TTS



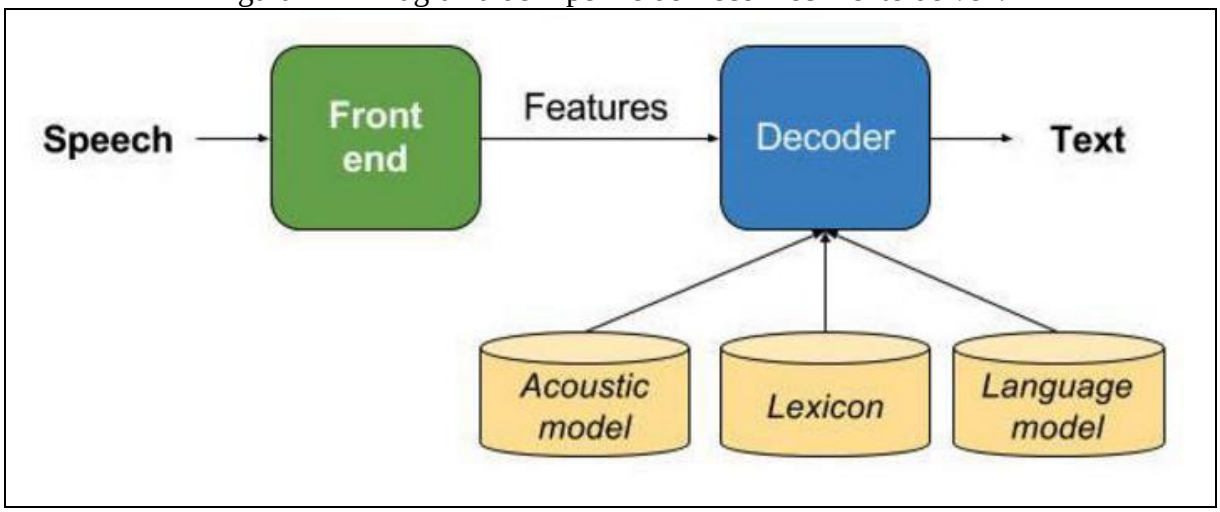
Fonte: Costa et al. (2012).

Para efetuar o reconhecimento da voz é preciso entender como a fala funciona (MCLELLAN, 2016), mas para o contexto deste projeto é mais conveniente começar explicando como a linguagem funciona.

A unidade básica da linguagem é o fonema, que seria então a menor parte de uma palavra, podem ser vogais, consoantes, ditongos e outros existentes em diferentes linguagens. O reconhecimento da fala envolve a digitalização do fluxo analógico de fonemas e enunciados, aplicando uma bateria de técnicas estatísticas para identificar possíveis palavras que estão sendo faladas. (MCLELLAN, 2016).

Da mesma forma que o sistema TTS o ASR também pode ser dividido em duas partes: *Front-end* o módulo responsável pela digitalização dos sinais de áudio analógicos, e o *Back-end* que Mclellan (2016) o chama de *Decoder* (Figura 12).

Figura 12 – Diagrama de Pipeline de Reconhecimento de voz.



Fonte: Mclellan (2016).

O *Back-end* possui três componentes: O Modelo Acústico (*Acoustic model*) é construído por uma base compilada de representação de fonemas, ele seleciona a provável cadeia de fonemas que podem formas palavras. O Léxico (*Lexicon*) é formado por um dicionário de palavras e o Modelo de Linguagem (*Language model*) restringe a escolha de palavras que fazem mais sentido na frase. Os sistemas ASR precisam ter esse refinamento por

causa de ruídos ou sotaques que podem interferir na identificação de uma palavra. (MCLEAN, 2016).

Para efetuar o processamento da linguagem natural, é necessário usar uma *Application Programming Interface* (API) que suporta aplicações TTS e ASR, com métodos que permitem ao programador abstrair requisitos de baixo nível do processamento. A seguir são descritas duas APIS para o processamento da linguagem natural utilizadas no protótipo.

O MaryTTS é um sintetizador de voz multilíngue de código aberto feito em Java. Foi originalmente desenvolvido pela DFKI (Deutsches Forschungszentrum für künstliche Intelligenz) e o Institute of Phonetics da Universidade de Saarland. Atualmente é mantido pelo Multimodal Speech Processing da Cluster of Excellence e DFKI. Sua arquitetura é similar a exposta na Figura 11 com quatro partes: pré-processamento ou normalização do texto, analise linguística, cálculo dos parâmetros acústicos com e síntese de voz (DFKI GMBH, 2016).

A segunda tecnologia é o CMUSphinx, um kit de ferramentas de reconhecimento de fala e modelos acústicos. No protótipo foi usada a biblioteca de reconhecimento de voz Sphinx 4 – 5prealpha feita totalmente em Java. Ela fornece uma maneira de converter gravação de fala em texto com a ajuda de modelos acústicos. Sendo compatível com servidores e aplicação desktop e com suporte a várias línguas. (CMUSPHINX, 2017).

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas de desenvolvimento do protótipo. A seção 3.1 apresenta os requisitos funcionais e não funcionais do protótipo. A seção 3.2 descreve a especificação através de diagramas da Unified Modeling Language (UML). A seção 3.3 apresenta os detalhes da implementação, técnicas, ferramentas e bibliotecas utilizadas, e a funcionalidade do protótipo. Por fim, a seção 3.4 apresenta os resultados obtidos com a elaboração deste trabalho.

3.1 REQUISITOS

O protótipo proposto neste trabalho deve:

a) efetuar o cadastro de níveis do teste de proficiência conforme definido pelo CEFR (RF);
b) permitir para cada nível o cadastro de perguntas com suas respectivas respostas (RF);
c) permitir associar textos e imagens para avaliação nas perguntas (RF);
d) possibilitar a associação de áudios e síntese de voz para avaliar a habilidade de compreensão auditiva (RF);
e) validar a pronúncia através do reconhecimento de voz (RF);
f) efetuar o teste de nivelamento considerando as perguntas cadastradas (RF);
g) emitir o resultado do teste de nivelamento com base nas respostas corretas e incorretas (RF);
h) permitir a persistência dos dados cadastrados em formato eXtensible Markup Language (XML) (RNF);
i) ser desenvolvido em Java no ambiente de desenvolvimento Eclipse (RNF).

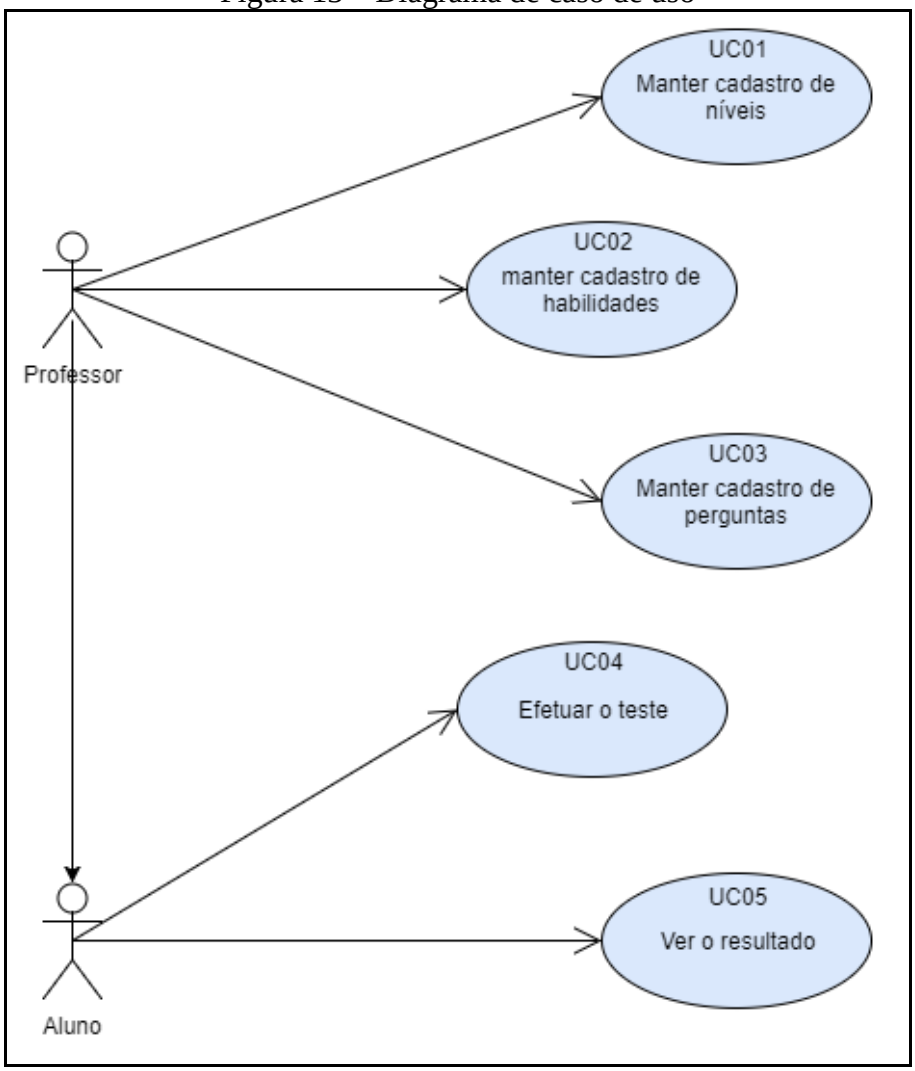
3.2 ESPECIFICAÇÃO

Para a especificação do protótipo foram utilizados diagramas em formato UML, desenvolvidos no Draw.io uma ferramenta de diagramação gratuita usada junto do Google Drive. Usou-se a mesma para a elaboração do diagrama de caso de uso e do diagrama de classes.

3.2.1 Caso de uso

Nesta seção são especificadas todas as ações executadas pelo ator `Professor` (Administrador) do protótipo e pelo ator `Aluno` (participante) do teste de proficiência. A Figura 13 mostra o diagrama de caso de uso com as ações dos dois atores.

Figura 13 – Diagrama de caso de uso



Fonte: elaborado pelo autor.

O Ator professor pode cadastrar dados que serão carregados pelo protótipo. O aluno pode apenas ver esses dados na forma de perguntas do teste de proficiência. No caso de uso `UC01 – Manter cadastro de níveis`, o professor pode cadastrar os níveis das perguntas do teste, similar aos níveis do CEFR. Também podem ser cadastrados níveis personalizados dependendo do formato do curso. No caso de uso `UC02 – Manter cadastro de habilidades`, o professor pode cadastrar uma área de habilidade, e associar elementos de texto, áudio ou imagens às perguntas. No caso de uso `UC03 – Manter cadastro de perguntas`, o professor pode cadastrar uma pergunta, mas para isso é necessário associá-la a um nível e a uma habilidade previamente cadastrada.

As ações do ator aluno também estão presentes na Figura 13. Vale ressaltar que o professor também pode efetuar essas mesmas ações no seu nível de acesso. No caso de uso `UC04 – Efetuar o teste`, o aluno dá início ao teste de proficiência e deve de preferência responder as perguntas até o final do teste. No caso de uso `UC05 – Ver o resultado`, o aluno verá um relatório do seu desempenho no teste, junto do detalhamento das perguntas respondidas e não respondidas e respondidas erradas também.

3.2.2 Diagramas de classes

Nesta seção são mostrados os diagramas de classes do protótipo. Eles especificam a estrutura de relacionamento das principais classes do protótipo.

3.2.2.1 Diagrama da estrutura dos dados

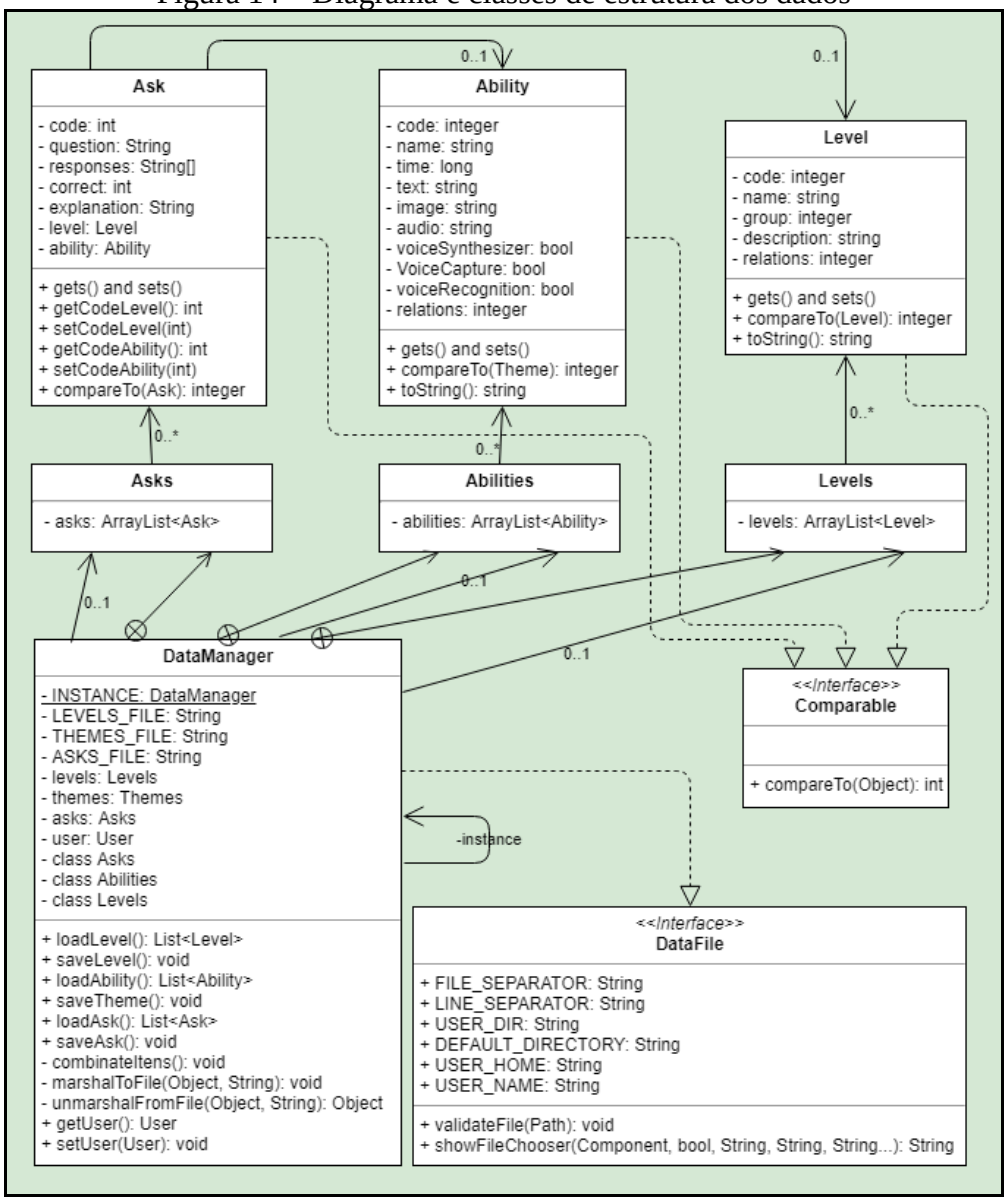
A Figura 14 ilustra o diagrama de classes de estrutura dos dados. Essas classes são utilizadas para manter o cadastro e a persistências dos dados do protótipo. Abaixo são descritos a utilidade das classes presentes na Figura 14:

a) classe `Level` – classe responsável por armazenar o nível de uma pergunta, essa classe associada a pergunta, serve para indicar a complexidade da pergunta e é usada como referência no teste de nivelamento;
b) classe `Ability` – classe responsável por armazenar uma habilidade. Essa classe especifica uma área de habilidade a ser avaliada pela pergunta, como por exemplo a habilidade de interpretar um texto;
c) classe `Ask` – classe responsável por armazenar uma pergunta. A classe pergunta precisa estar associada a um nível (`Level`) e uma habilidade (`Ability`) para funcionar;
d) Interface `Comparable` – interface do Java responsável por declarar o método `compareTo` que serve para implementar um critério de ordenação de listas. As classes `Level`, `Ability` e `Ask` implementam essa interface;
e) classe `DataManager` – classe responsável pelo gerenciamento da persistência dos dados no sistema de arquivos do computador;
f) classes `Asks`, `Abilities` e `Levels` – classes internas da classe `DataManager`, e são responsáveis por armazenar uma lista dos objetos `Ask`, `Ability` e `Level` e também são usadas como modelo para a conversão dos objetos Java para o formato XML;

g) interface `DataFile` – interface que disponibiliza alguns métodos e atributos para acesso e validação de endereços do sistema de arquivos.

As classes mais importantes do protótipo são: Ask (pergunta), Ability (habilidade) e Level (nível). Essas três classes juntas formam a estrutura básica de uma pergunta do teste.

Figura 14 – Diagrama e classes de estrutura dos dados



Fonte: elaborado pelo autor.

Para o protótipo lidar com um número indeterminado de perguntas (`Ask`), habilidades (`Ability`) e níveis (`Level`) registrados, foram utilizadas listas presentes nas classes internas `Asks`, `Abilities` e `Levels` da classe `DataManager`. Por isso as classes `Ask`, `Ability` e `Level` implementam da interface `Comparable` o método `compareTo` que estabelece o critério de ordenação.

No protótipo o atributo `code` (código) presente nas três classes ligadas a interface `Comparable` se tornou o atributo padrão do critério de ordenação desses elementos. Embora

um critério de ordenação diferenciado fosse especialmente importante para a classe `Level`, porquê o protótipo precisa distinguir um nível avançado de um nível mais básico. O atributo `code` da dessa classe também acabou por permanecer como elemento de ordenação. Assim, o cadastro dos níveis no protótipo deve ser feito de modo ordenado, do mais básico para o mais avançado, sem essa regra o protótipo ordenaria as perguntas erroneamente na construção do teste de proficiência, começando pela pergunta mais complexa. Além do atributo `code`, as classes `Ability` e `Level` possuem um atributo chamado `relations` para indicar a quantidade de perguntas a que estão relacionadas. Se o valor desse atributo for igual a zero significa que o nível ou habilidade não estão associados a nenhuma pergunta e podem ser deletados pelo professor.

A classe `Ask` possui um atributo associado um ao objeto `Level` e outro ao objeto `Ability`. Mas também existe dentro da classe `Ask` dois elementos do tipo `int`, chamados `codeLevel` e `codeAbility` que armazenam o valor do atributo `code` das classes `Level` e `Ability`. A justificativa da existência destes atributos tipo `int` é esclarecido na página 40.

3.2.2.2 Diagrama do teste de proficiência

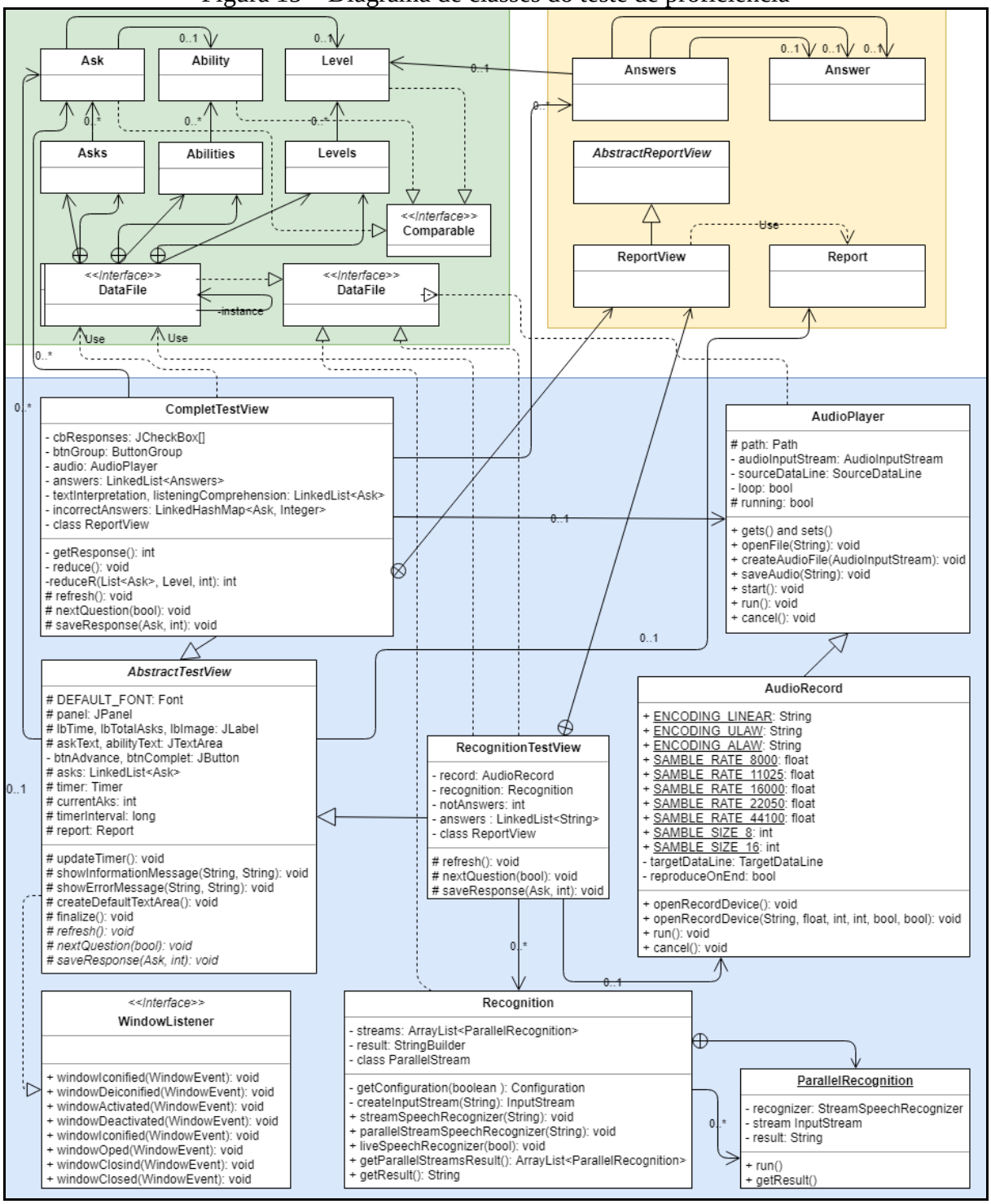
A Figura 15 mostra o diagrama de classes do teste de proficiência. As classes apresentadas na Figura 14 também estão presentes neste diagrama, mas com menos detalhes, assim como as classes do relatório que são apresentadas na Figura 18. Abaixo são descritos alguns detalhes sobre cada classe presente na Figura 15:

a) classe `CompletTestView`: classe responsável pelo teste de proficiência quanto em questões de gramática, interpretação de texto e compreensão auditiva, estas questões são identificadas através dos valores contidos nos atributos da classe `Ability` associada a classe `Ask`;
b) classe `RecognitionTestView`: classe responsável pelo teste de pronúncia;
c) classe `AbstractTestView`: as classes `CompletTestView` e `RecognitionTestView` são herdeiras dessa classe abstrata que implementa a interface `WindowListener`;
d) interface `WindowListener`: interface que responde aos eventos das Janelas;
e) classe `AudioPlayer`: classe responsável pela reprodução de arquivos de som, inclusive os áudios gerados por síntese de voz que são salvos em arquivo;
f) classe `AudioRecord`: classe responsável pela gravação do áudio do microfone;

g) classe `Recognition`: classe responsável por fazer a comunicação com o interpretador de voz;

h) classe `ParallelRecognition`: classe interna da classe `Recognition` e é responsável por fazer a interpretação da voz em paralelo.

Figura 15 – Diagrama de classes do teste de proficiência



Fonte: elaborado pelo autor.

O protótipo possui duas classes de teste de proficiência. Uma chamada `CompletTextView` e outra `RecognitionTestView` que realiza o teste de pronúncia em janela

separada em razão de performance e funcionalidade. Para essas duas classes de teste realizarem o teste de proficiência elas precisam carregar os objetos `Ask` do disco do computador. Isso é feito através da classe `DataManager` que possui internamente três classes com listas de objetos. O método que retorna a lista de perguntas para as classes de teste é chamado `loadAsk`.

As duas classes de teste do protótipo avaliam habilidades diferente. Assim é necessário que essas classes de testes saibam distinguir os diferentes tipos de habilidades a que as perguntas estão associadas. A Figura 16 mostra um trecho de código do método `WindowOpened` da classe `CompletTestView`.

Figura 16 – Trecho de código de seleção e ordenação de perguntas

```
219    private void reduce() {
220        HashSet<Level> levels = new HashSet<Level>();
221        asks.forEach(a -> {
222            if (!levels.contains(a.getLevel()))
223                levels.add(a.getLevel());
224        });
225        textInterpretation.forEach(a -> {
226            if (!levels.contains(a.getLevel()))
227                levels.add(a.getLevel());
228        });
229        listeningComprehension.forEach(a -> {
230            if (!levels.contains(a.getLevel()))
231                levels.add(a.getLevel());
232        });
233
234        Iterator<Level> i = levels.iterator();
235        while (i.hasNext()) {
236            Level level = i.next();
237            int value1 = reduceR(listeningComprehension, level, 5);
238            int value2 = reduceR(textInterpretation, level, 5);
239            reduceR(asks, level, 20 - (value1 + value2));
240            i.remove();
241        }
242    }
243
244    private int reduceR(List<Ask> list, Level level, int size) {
245        List<Ask> reduce = list.stream().filter(a -> a.getCodeLevel() == level.getCode()).collect(Collectors.toList());
246        list.removeAll(reduce);
247
248        while (reduce.size() > size) {
249            int index = new Random().nextInt(reduce.size()-1);
250            reduce.remove(index);
251        }
252        list.addAll(reduce);
253        return reduce.size();
254    }
255
256    public void windowOpened(WindowEvent e) {
257        setCursor(new Cursor(Cursor.WAIT_CURSOR));
258        DataManager dataManager = DataManager.getInstance();
259        try {
260            asks.addAll(dataManager.loadAsk().stream().filter(
261                    a -> a.getLevel().getGroup() <= report.getGroup() && a.getAbility().isVoiceRecognition() == false)
262                    .collect(Collectors.toList()));
263
264            textInterpretation.addAll(asks.stream().filter(a -> a.getAbility().getText().isEmpty() == false)
265                    .collect(Collectors.toList()));
266            listeningComprehension.addAll(textInterpretation.stream()
267                    .filter(a -> a.getAbility().isVoiceSynthesizer() == true || a.getAbility().isVoiceCapture() == true)
268                    .collect(Collectors.toList()));
269            asks.removeAll(textInterpretation);
270            textInterpretation.removeAll(listeningComprehension);
271
272            reduce();
273
274            Comparator<Ask> comparator = (a1, a2) -> a1.getCodeLevel() - a2.getCodeLevel();
275            asks.sort(comparator);
276            textInterpretation.sort(comparator);
277            listeningComprehension.sort(comparator);
```

Fonte: elaborado pelo autor.

Na linha 260 é feita a chamado do método `loadAsk` da classe `DataManager`. A classe `CompletTestView` não trabalha com perguntas de teste de pronúncia. Assim, na linha 261 é

tratado o critério de seleção das perguntas por código *lambda*, onde o atributo lógico `voiceRecognition` da classe `Ability` deve ser igual a `false`. Na mesma linha também é comparado o valor do atributo `group` (grupo) da classe `Level`. Um teste do grupo `básico` não deve agregar perguntas do grupo `intermediário`, mas um teste de grupo `intermediário` deve agregar as perguntas do grupo `básico` e o teste do grupo `avançado` deve agregar as perguntas de grupo `básico`, `intermediário` e `avançado` juntos.

Nas linhas entre 264 e 270 é feita a separação das perguntas de interpretação de texto e compreensão auditiva, colocadas em listas separadas. Na linha 272 é chamado o método `reduce` para fazer a redução do número de perguntas removendo-as aleatoriamente. E por fim na linha 274 é criado um `Comparator` para fazer a ordenação das perguntas por ordem crescente de nível. O teste de proficiência é dividido em etapas, primeiro por questões de gramática, depois interpretação de texto e por último, compreensão auditiva e o critério de ordenação é por ordem crescente de nível. Observa-se que dentro do método `reduce` é especificado um número limite de no máximo 20 perguntas por nível na linha 239 sendo que não pode haver mais de 5 de interpretação de texto e outras 5 de compreensão auditiva.

A classe `RecognitionTesteView` monta o teste formado por questões de pronúncia. Na Figura 17 no método `windowOpened` é feita a seleção de perguntas onde o atributo `voiceRecognition` da classe `Ability` deve ser igual a `true`. E em seguida é feita uma redução para no máximo 10 perguntas, sem verificar o grupo a que elas pertencem. O teste de pronúncia tambem é detalhado nas páginas 36 e 50.

Figura 17 – Trecho de código montando teste de pronúncia

```
123     public void windowOpened(WindowEvent e) {
124         setCursor(new Cursor(Cursor.WAIT_CURSOR));
125         DataManager dataManager = DataManager.getInstance();
126         try {
127             asks.addAll(dataManager.loadAsk().stream()
128                     .filter(a -> a.getLevel().getGroup() <= 2 && a.getAbility().isVoiceRecognition())
129                     .collect(Collectors.toList()));
130 
131             while (asks.size() > 10) {
132                 int index = new Random().nextInt(asks.size());
133                 asks.remove(index);
134             }
```

Fonte: elaborado pelo autor.

3.2.2.3 Diagrama do relatório do teste de proficiência

Para o protótipo gerar o relatório do teste de proficiência e calcular o resultado do teste ele utiliza as classes expostas na Figura 18. A seguir é descrito o que cada classe da Figura 18 faz:

a) classe `Answer`: classe responsável pela contagem das respostas. Cada atributo dessa classe corresponde aos três estados que a resposta de uma pergunta pode

assumir durante o teste: `notAnswer` (não respondido), `correct` (correto) e `incorrect` (incorreto).

b) classe `Answers`: classe responsável por associar as respostas do teste com o nível. Assim obtém-se quantas respostas corretas foram dadas em determinado nível. Cada atributo do tipo `Answer` dessa classe representa uma habilidade avaliada menos a pronúncia que é tratada separadamente;

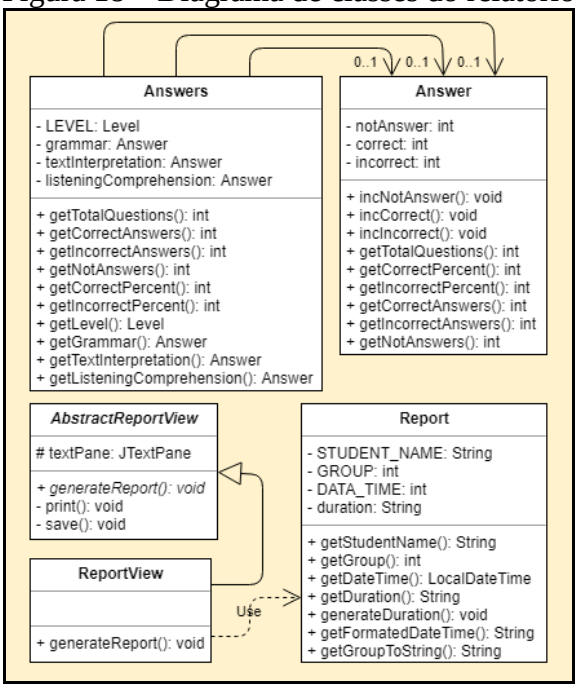
c) classe `Report`: classe responsável por armazenar os parâmetros iniciais do teste de proficiência como o nome do aluno, o grupo das perguntas, a data de realização do teste e no final calcula a duração do teste;

d) classe `AbstractReportView`: classe responsável por gerar uma tela de relatório;

e) classe `ReportView`: classe herdeira de `AbstractReportView`. Cada classe de teste `CompletTextView` e `RecognitionTestView` precisa de uma tela de relatório diferente. Dentro dessas classes existe uma classe interna herdeira da classe `AbstractReportView` que é representada no diagrama da Figura 18 como uma única classe chamada `ReportView`.

As classes `CompletTestView` e `RecognitionTestView` recebem pelo método construtor, a classe Report. É através dessa classe pelo atributo `group` que as perguntas são selecionadas no método `windowsOpened` da classe `CompletTestView`. Os demais atributos da classe `Report` são usados na tela de relatório.

Figura 18 – Diagrama de classes do relatório



Fonte: elaborado pelo autor.

A Figura 19 mostra o trecho de código da classe `CompletTestView` onde é realizada a contagem de respostas corretas, incorretas e não respondidas. Esse método é executado toda vez que o teste avança para a próxima pergunta. Entre as linhas 172 e 181 é implementado um algoritmo de busca para encontrar um objeto `Anwers` que tenha a mesma classe `Level` (nível) da pergunta passada por parâmetro pelo objeto `Ask`. Isso é necessário porquê a resposta precisa estar associada ao nível, assim descobre-se quantas respostas erradas ou corretas foram dadas para cada nível da pergunta.

Figura 19 – Trecho de código contagem de respostas

```
170    @Override
171    protected void saveResponse(Ask ask, int response) {
172        Answers answer = answers.getLast();
173        if (answer.getLevel().compareTo(ask.getLevel()) != 0) {
174            Optional<Answers> optional = answers.stream().filter(r -> r.getLevel().compareTo(ask.getLevel()) == 0)
175                    .findFirst();
176
177            answer = optional.orElse(new Answers(ask.getLevel()));
178
179            if (!optional.isPresent())
180                answers.addLast(answer);
181        }
182
183        Ability ability = ask.getAbility();
184        if (ability.getText().isEmpty()) {// gramática
185            if (response == -1) // não respondida
186                answer.getGrammar().incNotAnswer();
187            else if (response == ask.getCorrect())// respondida correta
188                answer.getGrammar().incCorrect();
189            else {// respondida errada
190                answer.getGrammar().incIncorrect();
191                incorrectAnswers.put(ask, response);
192            }
193        } else if (ability.isVoiceSynthesizer() || ability.isVoiceCapture()) {// compreensão auditiva
194            if (response == -1) // não respondida
195                answer.getListeningComprehension().incNotAnswer();
196            else if (response == ask.getCorrect())// respondida correta
197                answer.getListeningComprehension().incCorrect();
198            else {// respondida errada
199                answer.getListeningComprehension().incIncorrect();
200                incorrectAnswers.put(ask, response);
201            }
202        } else if (ability.isVoiceRecognition()) {// reconhecimento de voz
203            ;
204        } else {// interpretação de texto
205            if (response == -1) // não respondida
206                answer.getTextInterpretation().incNotAnswer();
207            else if (response == ask.getCorrect())// respondida correta
208                answer.getTextInterpretation().incCorrect();
209            else {// respondida errada
210                answer.getTextInterpretation().incIncorrect();
211                incorrectAnswers.put(ask, response);
212            }
213        }
214    }
```

Fonte: elaborado pelo autor.

Entre as linhas 184 e 213 no bloco de ifs e elses a pergunta é classificada por habilidade, objeto `Ability`. E dentro de outro bloco de ifs e elses é discriminado o valor da resposta, recebido por passagem de parâmetro pelo atributo `response`. Se o valor de `response` for igual a -1 a pergunta não foi respondida. Nas linhas 191, 200 e 211 as perguntas respondidas erradas são alojadas numa fila.

A Figura 20 mostra um trecho de código onde é gerado o relatório em HTML. A linguagem HTML é escrita em formato de `String` e colocado dentro de um objeto `StringBuilder`. Na linha 430 o conteudo é transferido para o objeto `JTextPane` que

interpreta o código HTML na interface gráfica. Em várias linhas desse trecho de código são feitas chamadas a vários métodos gets das classes `Answers` e `Answer` que possuem os contadores das respostas. Entre as linhas 409 e 426 é montada a lista das perguntas erradas anteriormente salvas numa fila.

Figura 20 – Trecho de código geração de relatório

```
362         rel.append(
363                 "<tr style= \"border-top: 1px solid black; \"><table style= \"width: 100%; border: 0px;\">\n");
364         rel.append("<tr>\n");
365         rel.append("<th></th>");
366         rel.append("<th>Gramática</th>");
367         rel.append("<th>Interpretação de texto</th>");
368         rel.append("<th>Compreensão auditiva</th>");
369         rel.append("\n</tr>\n");
370 
371         rel.append("<tr style= \"text-align: center;\">\n");
372         rel.append("<th style= \"text-align: left;\">Total:</th>");
373         rel.append("<td>" + answers.getGrammar().getTotalQuestions() + "</td>");
374         rel.append("<td>" + answers.getTextInterpretation().getTotalQuestions() + "</td>");
375         rel.append("<td>" + answers.getListeningComprehension().getCorrectAnswers() + "</td>");
376         rel.append("\n</tr>\n");
377 
378         rel.append("<tr style= \"text-align: center;\">\n");
379         rel.append("<th style= \"text-align: left;\">Acertos:</th>");
380         rel.append("<td>" + answers.getGrammar().getCorrectAnswers() + " - "
381                 + answers.getGrammar().getCorrectPercent() + "%</td>");
382         rel.append("<td>" + answers.getTextInterpretation().getCorrectAnswers() + " - "
383                 + answers.getTextInterpretation().getCorrectPercent() + "%</td>");
384         rel.append("<td>" + answers.getListeningComprehension().getCorrectAnswers() + " - "
385                 + answers.getListeningComprehension().getCorrectPercent() + "%</td>");
386         rel.append("\n</tr>\n");
387 
388         rel.append("<tr style= \"text-align: center;\">\n");
389         rel.append("<th style= \"text-align: left;\">Erros:</th>");
390         rel.append("<td>" + answers.getGrammar().getIncorrectAnswers() + " - "
391                 + answers.getGrammar().getIncorrectPercent() + "%</td>");
392         rel.append("<td>" + answers.getTextInterpretation().getIncorrectAnswers() + " - "
393                 + answers.getTextInterpretation().getIncorrectPercent() + "%</td>");
394         rel.append("<td>" + answers.getListeningComprehension().getIncorrectAnswers() + " - "
395                 + answers.getListeningComprehension().getIncorrectPercent() + "%</td>");
396         rel.append("\n</tr>\n");
397 
398         rel.append("<tr style= \"text-align: center;\">\n");
399         rel.append("<th style= \"text-align: left;\">Não respondido:</th>");
400         rel.append("<td>" + answers.getGrammar().getNotAnswers() + "</td>");
401         rel.append("<td>" + answers.getTextInterpretation().getNotAnswers() + "</td>");
402         rel.append("<td>" + answers.getListeningComprehension().getNotAnswers() + "</td>");
403         rel.append("\n</tr>\n</table>\n");
404 
405         rel.append("\n</tr>");
406         rel.append("\n</table>\n</p>");
407     });
408 
409     if (!incorrectAnswers.isEmpty()) {
410         rel.append("<p>\n<table id= \"outerTable\">\n");
411         rel.append("<tr style= \"border-bottom: 1px solid black\">\n");
412         rel.append("<th>Respostas erradas</th>\n</tr>\n");
413         incorrectAnswers.forEach((a, i) -> {
414             rel.append("<tr style= \"border-bottom: 1px solid black\">\n");
415             rel.append("<table id= \"innerTable\">\n");
416             rel.append("<tr>\n<td colspan=\"2\">" + a.getQuestion() + "</td>\n</tr>\n");
417 
418             rel.append("<tr>\n");
419             rel.append("<td> &radic; " + a.getResponses()[a.getCorrect()] + "</td>\n");
420             rel.append("<td> X " + a.getResponses()[i] + "</td>\n");
421             rel.append("</tr>\n");
422 
423             rel.append("\n</tr>\n</table>\n");
424         });
425         rel.append("\n</table>\n</p>");
426     }
```

Fonte: elaborado pelo autor.

3.3 IMPLEMENTAÇÃO

Nesta seção são mostradas as técnicas e ferramentas utilizadas no desenvolvimento do Protótipo, como também a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O protótipo foi desenvolvido no ambiente de desenvolvimento Eclipse Oxygen, na linguagem de programação Java 8. A interface gráfica do protótipo foi implementada com os recursos da biblioteca Swing, padrão do próprio Java Development Kit (JDK). Para o armazenamento de dados foi usado a estrutura de arquivos em formato eXtensible Markup Language (XML) e a biblioteca usada para fazer a conversão dos objetos Java para XML foi a API Java Architecture for XML Binding (JAXB), também presente no JDK. Para o processamento da linguagem natural foram utilizadas, o Mary Text To Speech (TTS) na versão 5.2 e Sphinx versão 4-5prealpha. Além destas tecnologias também foram utilizados no protótipo outros recursos avançados presentes no JDK como Reflection (Reflexão), e Generics (Genérico) para se obter o máximo do reaproveitamento do código e eliminar a repetição, mas esses elementos não são abordados neste trabalho por não terem relação com os objetivos específicos do protótipo.

3.3.2 O processamento da Linguagem Natural

Um dos objetivos específicos do protótipo é avaliar a compreensão auditiva no idioma alemão usando síntese de voz. O protótipo, no entanto, oferece mais formas de avaliar a compreensão auditiva além da síntese de voz. A classe `AudioPlayer` é capaz de reproduzir todas as extensões de arquivos de áudio compatível com Java como .wav, .aiff e .au. A janela de cadastro de habilidades exposta na seção 3.3.4 na Figura 30 possui os recursos necessários para acessar esses arquivos no computador, reproduzi-los e guardá-los para serem usados durante o teste.

A Figura 21 mostra um trecho de código da etapa de criação de voz sintetizada pela tela de cadastro de habilidade. A ferramenta MaryTTS na versão 5.2 foi usada no protótipo. Na linha 216 é criada a instância da classe `MaryInterface` e na linha 228 é chamado o método `generateAudio` com o texto passado por parâmetro. O método retorna um objeto do tipo `AudioInputStream` para ser reproduzido pela classe `AudioPlayer` que acessa o sistema de som do computador.

Durante o teste de proficiência o código para reproduzir o áudio é igual ao exposto nas linhas 229 e 230 da Figura 21. Na linha 229 um arquivo de áudio é aberto pela classe

`AudioPlayer` e em seguida o método `start` é chamado e põe o áudio para ser reproduzido. O atributo `audio` da classe `Ability` contém o endereço do arquivo de som no computador.

Figura 21 – Método que gera a voz sintetizada

```
210    private void createSynthesizer() {
211        try {
212            if (taText.getText().isEmpty())
213                throw new IllegalArgumentException("É necessario Texto para a Síntese de Voz!");
214
215            tfAudio.setText(null);
216            MaryInterface marytts = new LocalMaryInterface();
217            marytts.setLocale(Locale.GERMAN);
218            marytts.setInputType("TEXT");// opcional
219            marytts.setOutputType("AUDIO");// opcional
220            marytts.setStyle("happy");// opcional
221
222            Object[] voices = marytts.getAvailableVoices().toArray();
223            Object value = JOptionPane.showInternalInputDialog(this, "Selecione a Voz:", "Mary Text To Speech",
224                    JOptionPane.QUESTION_MESSAGE, null, voices, voices[0]);
225            if (value != null) {
226                marytts.setVoice(value.toString());
227
228                audioPlayer.createAudioFile(marytts.generateAudio(taText.getText()));
229                audioPlayer.openFile(audioPlayer.getPath());
230                audioPlayer.start();
231                tfAudio.setText(audioPlayer.getPath());
232
233                alterStateButtons(btnRecord.isEnabled(), true, true, true);
234            }
235        } catch (IllegalArgumentException | MaryConfigurationException | SynthesisException | IOException
236                | LineUnavailableException | UnsupportedAudioFileException e) {
237            showMessage(e.getMessage(), e.getClass().getSimpleName());
238        }
239    }
```

Fonte: elaborado pelo autor.

Outro objetivo específico do protótipo é efetuar o teste de pronúncia no idioma alemão. A biblioteca Sphinx oferece três interfaces de entrada de voz (CMUSPHINX):

1- LiveSpeechRecognizer: utiliza a voz capturada voz pelo microfone em tempo real;
2- StreamSpeechRecognizer: utiliza um objeto InputStream com dados de voz podendo ser transmitido por arquivo ou rede.
3- SpeechAligner: alinhamento de texto com o arquivo de áudio.

O primeiro método acusou algumas exceções nos testes realizado e o tempo de resposta da ferramenta da inicialização é demorado. O segundo trouxe melhores resultados porquê permite a gravação do áudio capturado com frequência de som e formato adequado para a biblioteca. A classe `AudioRecord` é capaz de captar o áudio do microfone e gravar em arquivo com formato e frequência especificados por passagem de parâmetro do método `openRecordDevice`. Esse método é mostrado no APÊNDICE C – Captura de áudio pelo microfone

A Figura 22 mostra um trecho de código feito para chamar a biblioteca Sphinx. Na linha 135 é implementado o construtor da Classe `ParallelRecognizer` que recebe um objeto `InputStream`. O objeto `Configuration` contém as informações do modelo acústico, dicionário e gramática que a biblioteca deve usar para fazer a interpretação dos dados de voz. O objeto `InputStream` foi carregado com o arquivo de áudio gravado pela classe

`AudioRecord`. Na linha 155, dentro de um laço, são obtidos em formato de texto (`String`) os resultados da interpretação da fala pelo método `getHypothesis`.

Figura 22 – Método para reconhecimento de voz

```
129    public final class ParallelRecognition extends Thread implements Runnable {
130        private StreamSpeechRecognizer recognizer;
131        private InputStream stream;
132        private String result;
133
134        /* método construtor */
135        private ParallelRecognition(InputStream stream) throws IOException {
136            final String ACOUSTIC_MODEL = "resource:/germanAcoustic/";
137            final String DICTIONARY_PATH = "resource:/germanLm/cmusphinx-voxforge-de.dic";
138            final String LANGUAGE_MODEL = "resource:/germanLm/cmusphinx-voxforge-de.lm.bin";
139
140            Configuration configuration = new Configuration();
141            configuration.setAcousticModelPath(ACOUSTIC_MODEL);
142            configuration.setDictionaryPath(DICTIONARY_PATH);
143            configuration.setLanguageModelPath(LANGUAGE_MODEL);
144
145            this.recognizer = new StreamSpeechRecognizer(configuration);
146            this.stream = stream;
147        }
148
149        @Override
150        public void run() {
151            recognizer.startRecognition(stream);
152            StringBuilder hypothesis = new StringBuilder();
153            SpeechResult speechResult;
154            while ((speechResult = recognizer.getResult()) != null) {
155                hypothesis.append(speechResult.getHypothesis());
156                hypothesis.append(" ");
157            }
158            recognizer.stopRecognition();
159            result = hypothesis.toString();
160        }
```

Fonte: elaborado pelo autor.

Era esperado que com a transformação da voz em texto fosse possível avaliar a pronúncia. Poderia ser feito um comparativo entre o texto obtido pela biblioteca Sphinx com o texto solicitado na pergunta do teste, mas infelizmente não trouxe resultados. A ferramenta Sphinx falha no reconhecimento das palavras trazendo resultados na maioria das vezes distantes demais e inesperados. O modelo acústico usado muito provavelmente não estava treinado.

3.3.3 A Estrutura dos dados

Para o armazenamento dos dados foi utilizado a estrutura de arquivos em formato XML. A decisão de adotar o formato XML foi tomada em virtude de vários aspectos. O primeiro deles é o formato organizado e estruturado do arquivo que parece ideal para se armazenar um objeto Java. Assim como o número de dados relevantes para o protótipo realizar o teste de proficiência, não são necessários um número muito grande de perguntas que de outra forma tornaria o teste longo demais. Outro motivo é o JDK possuir recursos para efetuar a transformação das classes Java em formato XML. Assim sendo, o Formato hierárquico do arquivo XML mais do que se adequa ao objetivo do protótipo que é efetuar o

teste de proficiência em língua alemã. Se não fosse usado o formato de arquivo XML seria necessário criar uma outra estrutura para a leitura e gravação dos objetos em arquivos o que poderia causar certos transtornos no acréscimo de algum atributo ou a remoção de um deles numa classe.

A pasta de dados do protótipo fica no mesmo local onde o protótipo é executado. Ela será criada junto com os arquivos .xml em seu interior quando a janela de cadastro de perguntas ou uma das janelas de teste for aberta pela primeira vez. Se abrir apenas a janela de cadastro de nível ou de habilidades somente os arquivos .xml dessas classes serão criados. A razão disso é que a classe `Ask` (pergunta) precisa de uma classe `Level` (nível) e `Ability` (habilidade) para existir. Nesta pasta chamada `data` também são salvos os arquivos de relatório dos testes de proficiência em formato .html e os arquivos de áudio produzidos por captura do microfone ou por síntese de voz[2].

A API Java Architecture for XML Binding (JAXB) fornece uma maneira rápida e conveniente de vincular esquemas XML e representações Java, tornando fácil a tarefa dos desenvolvedores de incorporarem dados XML em aplicações Java (Oracle, 2017). As regras para a transformação das classes Java em formato XML são especificadas no código fonte das classes usando-se `annotations` (anotação). Como mostrado na Figura 23, sobre cada atributo e inclusive sobre o cabeçalho da classe estão declarados um elemento de anotação.

Figura 23 – Anotações em Java

```
17  @XmlAccessorType(XmlAccessType.FIELD)
18  public class Level implements Comparable<Level> {
19      @XmlAttribute(name = "code", required = true)
20      private int code;
21      @XmlElement(name = "name", required = true)
22      private String name;
23      @XmlElement(name = "group", required = true)
24      private int group;
25      @XmlElement(name = "description", required = true)
26      private String description;
27      @XmlElement(name = "relations", required = true)
28      private int relations;
29
30      /* contrutor padrão. */
31      public Level(Integer code) {
32          this.code = code;
33          name = null;
34          group = 0;
35          description = null;
36          relations = 0;
37      }
```

Fonte: elaborado pelo autor.

[2] Esses arquivos são criados por padrão na pasta do protótipo. Se forem movidos para outro lugar e o protótipo precisar deles durante um teste, é preciso atualizar a localização deles pela tela de cadastro de habilidades.

A transformação de objetos Java em formato de arquivo XML é chamado marshalling[3]. A Figura 24 mostra o trecho de código do método `marshallToFile` implementado na classe `DataManager` do protótipo. Na linha 145 é criado o objeto `Marshaller` que controla o processo de serialização. O objeto Marshaller também possui propriedades que podem ser definidas pelo método `setProperty` para especificar a codificação de saída dos dados XML como na linha 146. Na linha 147 o método `marshal` do objeto `Marshaller` é chamado recebendo por parâmetro o objeto raiz e o objeto de fluxo de saída onde será gravado o arquivo XML. (ORT e BHAKTI, 2013).

Figura 24 – método marshall

```
129    private final void marshallToFile(Object object, String file) throws JAXBException, IOException {
130        Path path = Paths.get(DEFAULT_DIRECTORY);
131
132        if (Files.notExists(path, LinkOption.NOFOLLOW_LINKS))
133            Files.createDirectory(path);
134        path = path.resolve(file);
135        if (Files.notExists(path, LinkOption.NOFOLLOW_LINKS))
136            Files.createFile(path);
137        else if (Files.exists(path, LinkOption.NOFOLLOW_LINKS))
138            validateFile(path);
139
140        // FileWriter foi a melhor opção para gravação com Jaxb
141        FileWriter writer = null;
142        try {
143            writer = new FileWriter(path.toFile());
144            JAXBContext context = JAXBContext.newInstance(object.getClass());
145            Marshaller marshaller = context.createMarshaller();
146            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
147            marshaller.marshal(object, writer);
148            writer.flush();
149        } finally {
150            if (writer != null)
151                writer.close();
152        }
153    }
```

Fonte elaborado pelo autor.

O objeto `JAXBContext` é um elemento presente na Figura 24 e Figura 25. Esse objeto fornece o ponto de entrada para a API JAXB. Ele é criado antes dos objetos `Marshaller` e `Unmarshaller` apresentado na Figura 25. Na criação do objeto JAXBContext é preciso especificar o objeto de contexto. Ele fornece a capacidade de serializar e desserializar as ligações dos elementos XML. (ORT e BHAKTI, 2013). No protótipo existem três objetos de contexto para serem usados nos métodos `marshallToFile` e `unmarshallFromFile`, sendo eles as classes internas `Asks`, `Abilities` e `Levels` da classe `DataManager` já detalhadas na seção 3.2.2.1.

A Figura 25 mostra o método `unmarshallFromFile` que utiliza o objeto `Unmarshaller` para fazer a desserialização do documento XML. Na linha 173 é chamado o método `unmarshal` dao objeto `Unmarshaller` que retorna um elemento da classe `Object`.

[3] Marshalling é um termo em inglês sem correspondência no português, contudo é um processo muito semelhante a serialização.

Figura 25 – método unmarshall

```
159    private final Object unmarshallFromFile(Object object, String file) throws JAXBException, IOException {
160        final Path path = Paths.get(DEFAULT_DIRECTORY, file);
161
162        if (Files.notExists(path, LinkOption.NOFOLLOW_LINKS))
163            marshallToFile(object, file);
164        validateFile(path);
165
166        // FileReader foi a melhor opção para leitura com Jaxb
167        Object obj = null;
168        FileReader reader = null;
169        try {
170            reader = new FileReader(path.toFile());
171            JAXBContext context = JAXBContext.newInstance(object.getClass());
172            Unmarshaller unmarshaller = context.createUnmarshaller();
173            obj = unmarshaller.unmarshal(reader);
174        } finally {
175            if (reader != null)
176                reader.close();
177        }
178        return obj;
179    }
```

Fonte: elaborado pelo autor.

Na seção 3.2.2.1 foi exposto que a classe `Ask` possui dois atributos `int` chamados `codeLevel` e `codeAbility` junto do relacionamento com as classes `Level` e `Ability`. A razão da existência dos atributos tipo `int` é eliminar a redundância ou repetição de dados que ocorre com a relação que a classe `Ask` possui com as classes `Level` e `Ability`. A Figura 26 mostra em tom de vermelho a diferença de um arquivo com redundância e outro sem.

Figura 26 – Estrutura XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<asks>
    <ask code="1">
        <level code="1">
            <name>A1</name>
            <group>0</group>
            <description>Pode entender e utilizar expressões
            familiares do dia a dia... </description>
            <relations>42</relations>
        </level>
        <ability code="1">
            <name>default</name>
            <time>30</time>
            <text></text>
            <image>F:\workspace\Project\data\Fahne.png</image>
            <audio></audio>
            <
            <
            <
            <
        </abilit
    <question>_
        <respo
            <
            <
            <
            <
        </respo
        <corre
    </ask>
<asks>
```

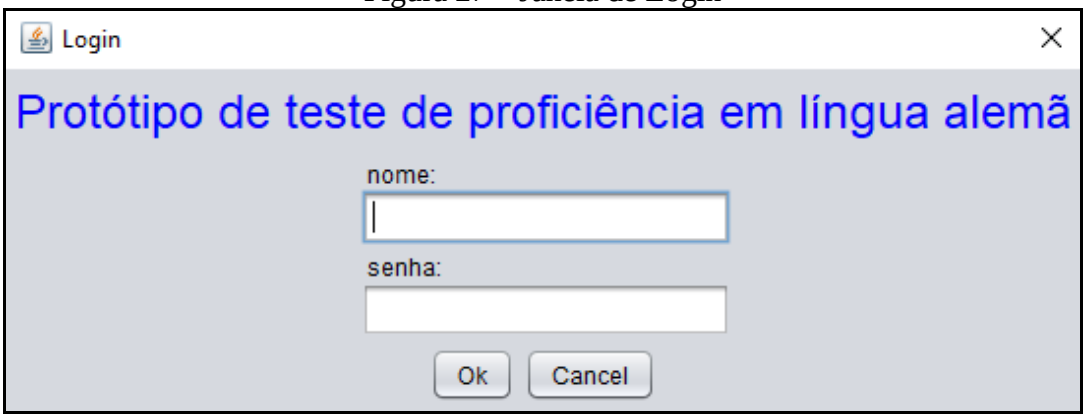
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<asks>
    <ask code="1">
        <codeLevel>1</codeLevel>
        <codeAbility>1</codeAbility>
        <question>___ heißen Sie?</question>
            <responses>
                <response>Was</response>
                <response>Wo</response>
                <response>Wie</response>
                <response>Wer</response>
            </responses>
        <correct>2</correct>
    </ask>
<asks>
```

Fonte: elaborado pelo autor.

3.3.4 Operacionalidade da implementação

Ao executar o protótipo, primeiro é necessário realizar o *login* conforme mostra a Figura 27. Usuário e senha `professor` dá acesso total ao protótipo, usuário e senha `aluno` dá acesso apenas às operações de teste de proficiência. Qualquer outro valor é recusado.

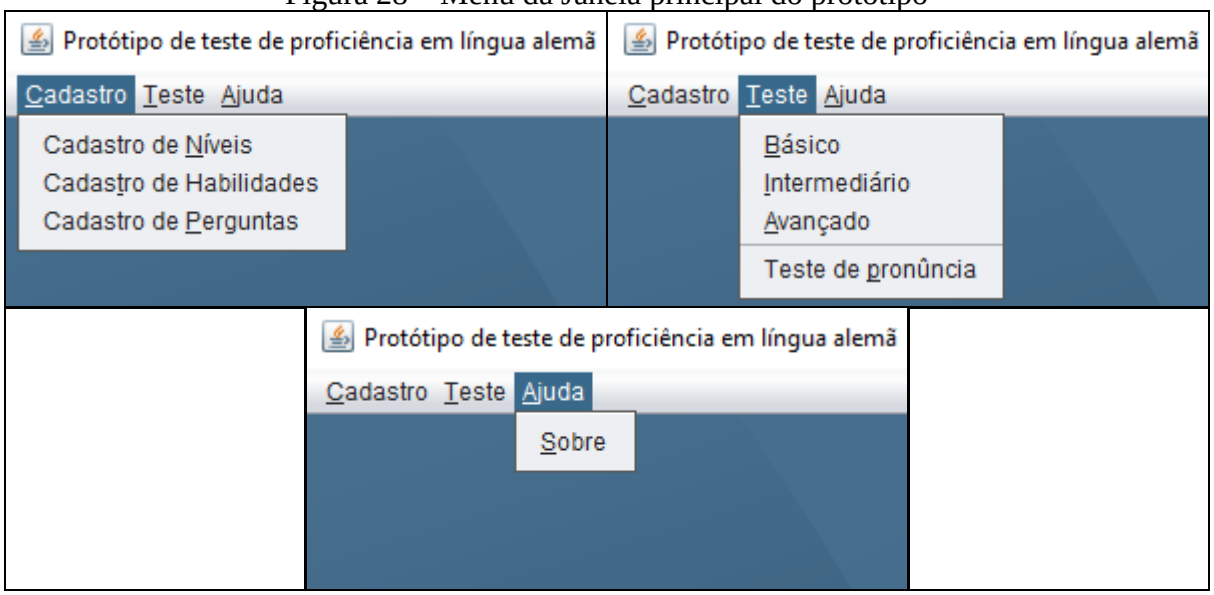
Figura 27 – Janela de Login



Fonte: elaborado pelo autor.

Após o *login* como `professor` ou `aluno`, a Janela principal do protótipo se abre. No alto dela é visível o `menu principal` que dá acesso a todos os recursos que o protótipo oferece, conforme a Figura 28. Existem três opções de cadastro que só serão visíveis ao `professor`. Quatro opções de teste de proficiência e um `item` de ajuda com informações do protótipo são visíveis a todos.

Figura 28 – Menu da Janela principal do protótipo

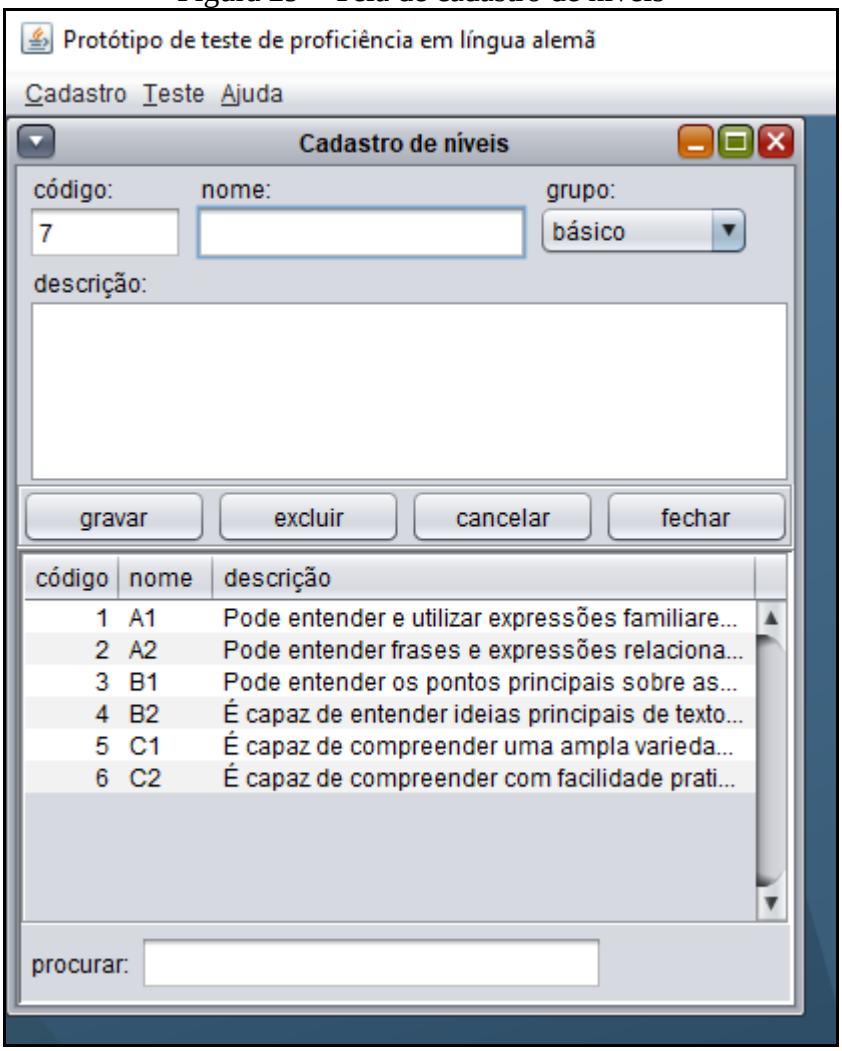


Fonte: elaborado pelo autor.

3.3.4.1 Janelas de cadastro

As janelas de cadastro do protótipo são todas internas e por isso são exibidas dentro da janela principal, como mostra a Figura 29.

Figura 29 – Tela de cadastro de níveis



Fonte: elaborado pelo autor.

Todas as janelas de cadastro possuem uma estrutura similar à da Figura 29. O campo `código` é comum em todas elas e seu valor é gerado automaticamente, não podendo ser alterado. Os quatro botões: `gravar`, `excluir`, `cancelar` e `fechar` também estão presentes em todas as janelas de cadastro junto da `tabela` que mostra dados já cadastrados e um campo de pesquisa para a `tabela`.

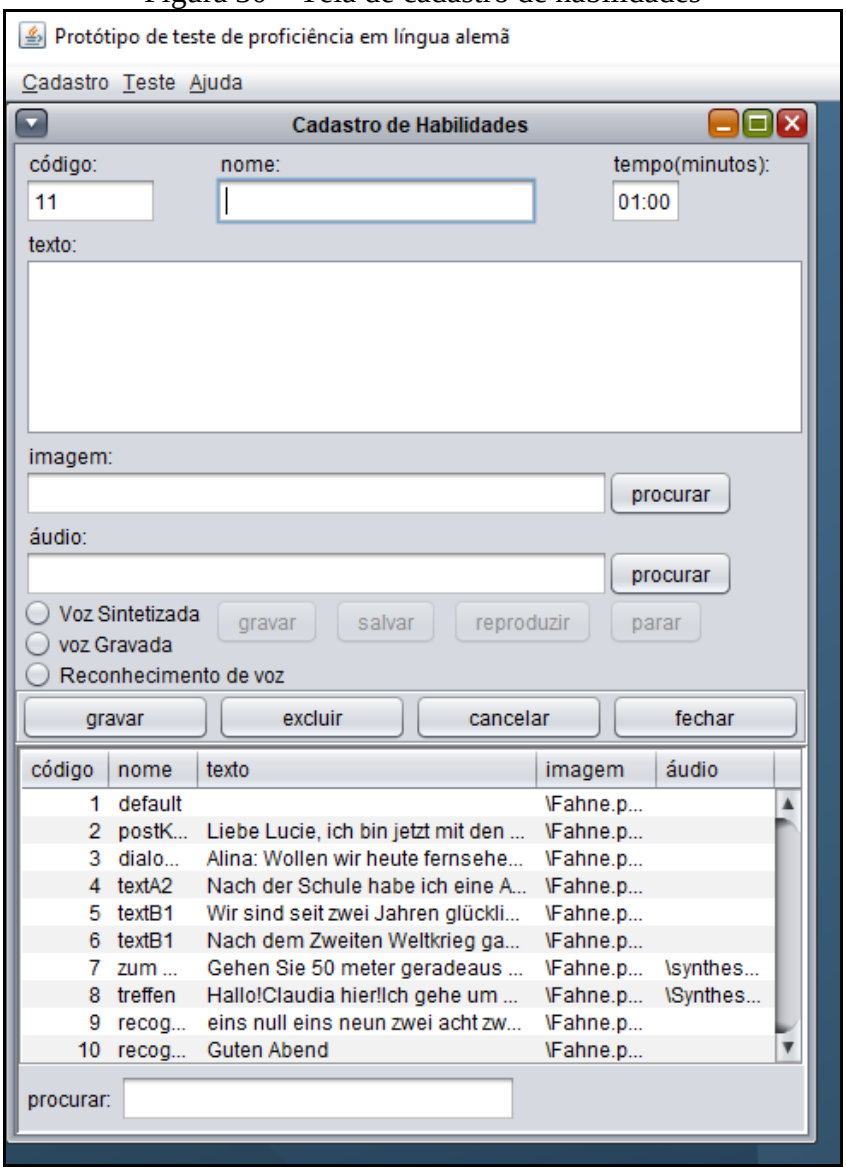
Os campos `nome`, `grupo` e `descrição` na Figura 29 são inerentes a classe nível do cadastro e todos devem ser preenchidos pelo `professor` antes de se efetuar a gravação dos dados. Exceto o campo `código` que como foi descrito, sempre estará atrelado a um valor. A ação da gravação dos dados é acionada pelo clique do botão `gravar`, com a gravação o nível também será inserido na última linha da `tabela`[4].

[4] A tabela foi o elemento do protótipo implementado com recursos como generics e reflection. Boa parte do código para inserir e remover elementos e a construção das colunas foram feitos usando esses recursos.

Quando se seleciona uma das linhas da `tabela` os campos da janela de cadastro são preenchidos com os dados do nível selecionado. Esses campos podem ter seus valores alterados, mas só substituirão os valores antigos com o clique do botão `gravar`. A ação de seleção de uma linha na `tabela` também é requerida para se efetuar a exclusão de dados, ação executada pelo clique do botão `excluir`, mas para que a exclusão seja bem-sucedida, é necessário que o nível não esteja relacionado com nenhuma pergunta cadastrada na tela de cadastro de perguntas.

O botão `cancelar` limpa todos os campos preenchidos, menos o campo `código` que é preenchido por um valor novo e se a `tabela` tiver uma linha selecionada ela será desmarcada também. O botão `fechar` apenas fecha a janela interna. O campo `procurar` gera um filtro na `tabela` mostrando apenas níveis com descrição relevantes com o termo da pesquisa.

Figura 30 – Tela de cadastro de habilidades



Fonte: elaborado pelo autor.

A Figura 30 mostra a tela de cadastro de habilidades. Associada a uma pergunta ela especifica a área de habilidade que deve ser avaliada pela pergunta. Os campos `nome`, `tempo` e `imagem` precisam estar obrigatoriamente preenchidos para se efetuar o cadastro de uma habilidade. O campo `nome` não possui nenhuma funcionalidade específica, mas pode ser usado como um título sugestivo ao tema da habilidade. O campo `tempo` especifica a duração em minutos que o participante do teste tem para responder à pergunta antes de avançar para a próxima pergunta. Se o tempo se esgotar o teste avança para a próxima pergunta desconsiderando qualquer resposta selecionada. O campo `tempo` está associado a uma habilidade para que perguntas associadas a uma mesma habilidade tenham o mesmo tempo em comum. O campo `tempo` não pode ter valores menor de 10 segundos e nem maior de 10 minutos.

O campo `texto` não é de preenchimento obrigatório e serve para armazenar o texto que deve ser interpretado durante o teste de proficiência. O campo `imagem` associa uma imagem que será exibida ao lado do texto na janela de teste, isso é mostrado na Figura 35 e Figura 38. O campo `áudio` associa um arquivo de áudio que será reproduzido durante o teste. Para selecionar uma `imagem` ou `áudio` é necessário clicar dos botões `procurar` posicionado ao lado desses campos.

Quando se seleciona um arquivo de áudio nesta tela, os botões `reproduzir` e `parar` serão habilitados, um serve para reproduzir o áudio e outro para interromper. O campo `texto` precisa estar obrigatoriamente preenchido para se efetuar a gravação da habilidade com o áudio. Mas neste caso o campo `texto` será usado para descrever o conteúdo do `áudio` e não para a interpretação de texto, uma vez que seu conteúdo será omitido pela janela de teste de proficiência para reproduzir o `áudio`.

Os botões de seleção `voz sintetizada`, `voz gravada` e `reconhecimento de voz` só podem ser selecionados individualmente, a ação de selecionar um deles desfaz a seleção de outro. A seleção dos botões `voz sintetizada` ou `voz gravada` habilitam o botão `gravar`. Se o botão `gravar` é clicado com a opção `voz sintetizada` ele vai executar o código já exposto na Figura 21, página 36. O conteúdo presente no campo texto dessa Figura 30 será usado para transformar texto em voz. Se o botão `gravar` for clicado com a opção `voz gravada`, o botão `parar` será habilitado e a voz do microfone será gravada até que o clique no botão `parar` interrompa a gravação.

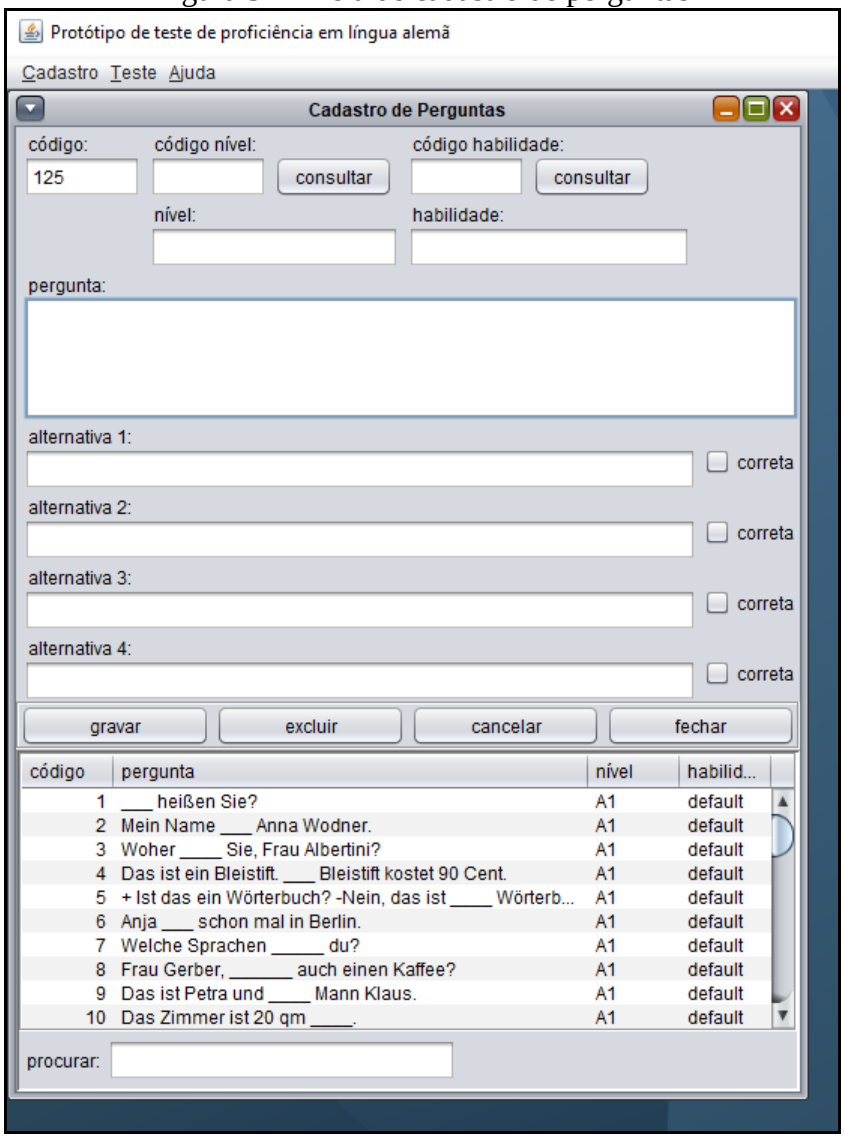
Ambos os botões de `voz sintetizada` e `voz gravado` produzem um arquivo de áudio que ficará salvo na pasta `data` do protótipo com o nome `record.wav`. O Botão `salvar`

estará habilitado nesta altura e com o clique dele será possível mover e mudar o nome do arquivo de áudio produzido. Mudar o nome do arquivo gravado é muito importante senão ele será sobrescrito na próxima gravação. Os botões `reproduzir` e `parar` sempre serão habilitados quando existir um áudio para ser reproduzido.

O botão `reconhecimento de voz` desabilita a o botão `procurar` ao lado do campo `áudio`, pois não precisa de arquivo de áudio. Mas por outro lado, o campo `texto` precisa ser preenchido para indicar no teste o que é requerido que o participando diga diante do microfone.

O cadastro de perguntas é mostrado na Figura 31. Para o cadastro de uma pergunta são necessários um `nível`, uma `habilidade`, uma `pergunta` e quatro alternativas como `resposta` sendo uma delas a `correta`.

Figura 31 – Tela de cadastro de perguntas

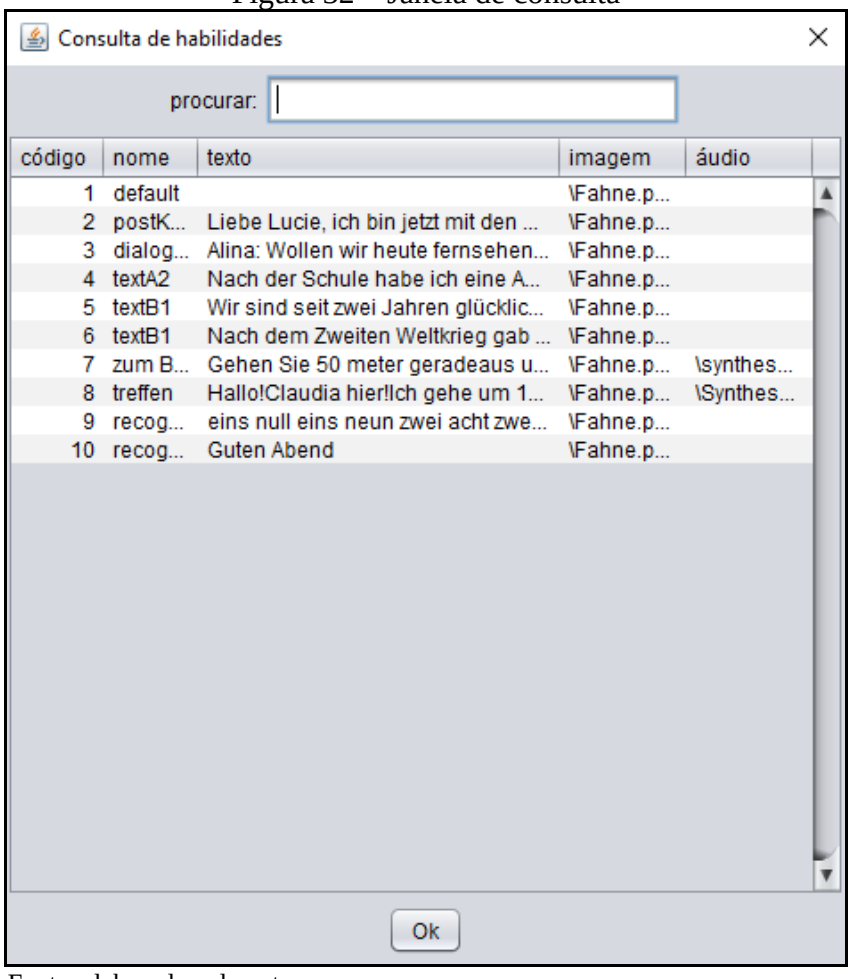


Fonte: elaborado pelo autor.

O `nível` e a `habilidade` são associados pelo clique dos botões `consultar`. O campo `pergunta` deve ser preenchido por uma pergunta, que de preferência deve ter relação com o conteúdo da `habilidade` associada como um texto para ser interpretado ou um áudio para ser compreendido. Os quatro campos para as possíveis `respostas` oferecem ao lado quatro outros campos para indicar a resposta `correta` e apenas uma pode ser selecionada.

Figura 32 mostra a janela de consulta aberta pela ação do botão `consultar` da tela de cadastro de perguntas. Essa janela de consulta só serve para selecionar o nível ou a habilidade para colocar no cadastro de perguntas. Basta clicar na `tabela` e no botão `ok` para confirmar a seleção. Na tela do cadastro de perguntas a habilidade e o nível selecionados ficarão associado a pergunta criada quando o botão `gravar` for pressionado.

Figura 32 – Janela de consulta



Fonte: elaborado pelo autor.

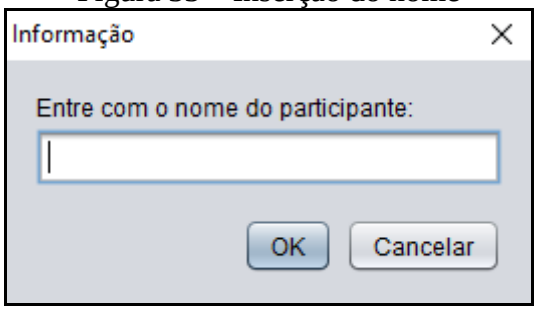
3.3.4.2 Janelas do teste de proficiência

A seguir são apresentados alguns detalhes sobre os quatro tipos de teste do protótipo. Na Figura 28 da seção anterior foram mostrados os quatro `itens do menu` principal usados para chamar as Janelas de teste:

a) `Básico`: monta um teste selecionando perguntas do grupo `básico` que equivale aos níveis A1 e A2 do CEFR;
b) `Intermediário`: monta um teste com perguntas do grupo `intermediário` que equivale aos níveis B1 e B2 do CEFR;
c) `Avançado`: monta um teste com perguntas do grupo `avançado` que equivale aos níveis C1 e C2 do CEFR;
d) `Teste de pronúncia`: monta um teste com perguntas de todos os grupos e nível, mas somente com habilidades relevantes a pronúncia.

No clique de um dos `itens de menu` o grupo a que o teste deve ser montado é armazenado na classe `Report` já exposta na seção 3.2.2.3. Também no clique de um desses `itens` é solicitado o nome de quem vai realizar o teste conforme a tela da Figura 33. Essa tela não aceita valores nulos e se pressionar no botão `cancelar` o teste é cancelado. O nome inserido nessa tela também ficará armazenado na classe `Report` junto da data e hora de início do teste para serem usados na tela de relatório.

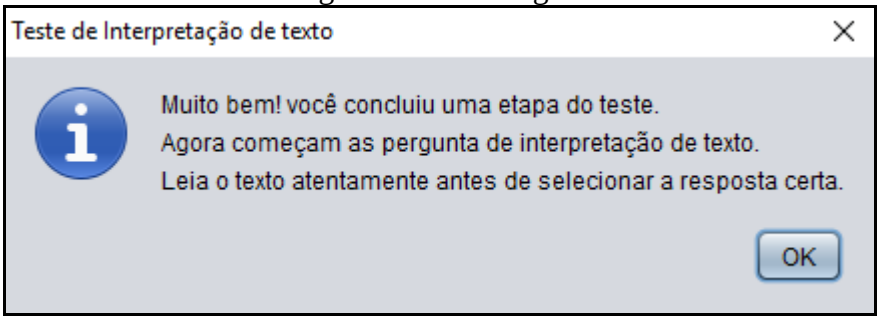
Figura 33 – Inserção do nome



Fonte: elaborado pelo autor.

Na seção 3.2.2.2 foi detalhada a implementação do método `windowOpened` das classes de teste. A seleção e ordenação das perguntas torna o teste divido, primeiro por perguntas de gramática, depois interpretação de texto e compreensão auditiva. A Figura 34 ilustra uma das mensagens que são mostradas entre as etapas que dividem o teste. Ela informa que tipos de perguntas estão por vir para o aluno se preparar antes de responder as novas perguntas.

Figura 34 – Mensagem



Fonte: elaborado pelo autor.

A Figura 35 mostra a tela do teste de alemão. O conteúdo da janela é montado com base nos valores dos atributos da classe `Ask` (pergunta) e `Ability` (habilidade). No alto da janela no canto esquerdo é mostrado o contador que indica o `tempo` que resta para o participante responder cada pergunta. Esse contador começa exatamente com o tempo salvo na classe `Ability` no cadastro de habilidades. Se o contador chegar a zero a resposta é desconsiderada e o teste avança para a próxima pergunta sem opção de retorno. No canto direito é mostrado quantas perguntas restam para concluir o teste. No centro da tela mais à esquerda é possível ver uma `imagem` pertencente a classe `Ability`. Se a classe `Ability` tiver um `texto` ele será mostrado ao lado direito da imagem. Se a classe `Ability` tiver incluso um áudio ele começará a ser reproduzido e continuará até que se avance para a próxima pergunta.

Figura 35 – Teste de proficiência

Fonte: elaborado pelo autor.

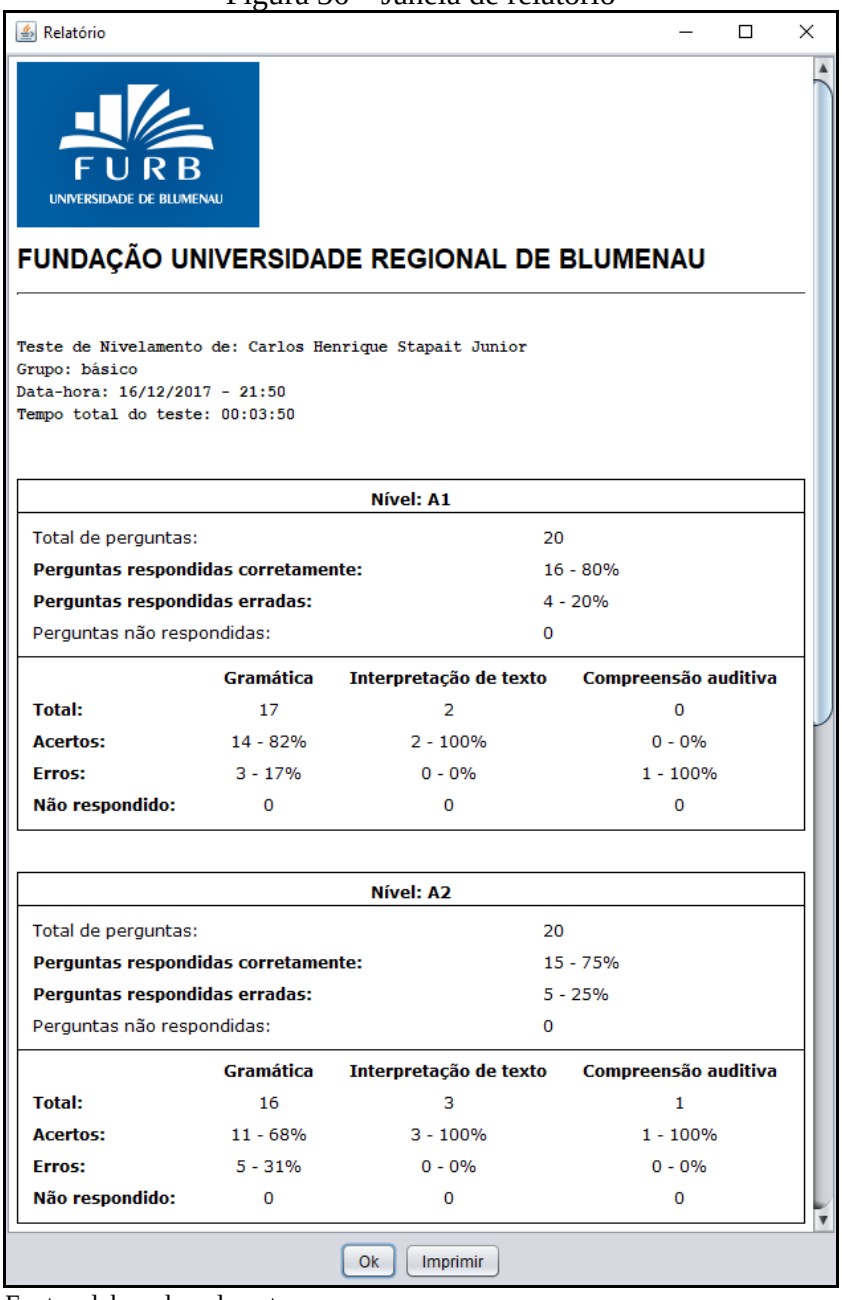
Ainda na Figura 35 na parte de baixo da `imagem` é mostrada a `pergunta` e logo em seguida quatro alternativas como possíveis `respostas` para esta pergunta. Somente uma alternativa pode ser selecionada. O botão `Avançar` confirma a resposta selecionada e avança para a próxima pergunta. Essa ação é necessária para se confirmar a resposta, esperar o contador zerar para avançar não salva a resposta.

O botão `Concluir` tem a mesma ação de fechar a janela do teste. Se for pressionado antes de chegar à última pergunta, então será mostrada uma mensagem para a confirmação do encerramento do teste. A resposta da pergunta atual será salva, mas todas as outras que não foram respondidas influenciarão no resultado do teste. O modo como é feita a contagem das respostas foi exposto na página 33 com trecho de código. Aquele trecho de código é

executado sempre que o teste avança para a próxima pergunta. Se o teste for encerrado antes de passar por todas as perguntas o trecho de código também é chamado para fazer a contagem de perguntas não respondidas.

A Janela de relatório é mostrada na Figura 36 e Figura 37. A Figura 36 mostra a parte da tela onde é mostrado o nome de quem realizou o teste, o grupo do nível das perguntas em que o teste foi montado. A data de realização e a hora de início do teste e o tempo total do teste. Nas tabelas do relatório são mostradas as estatísticas gerais do teste, o total de perguntas pertencente a cada nível, o total de perguntas acertadas e erradas. Quantidade de questões de gramática, interpretação de texto e compreensão auditivas que foram respondidas ou não.

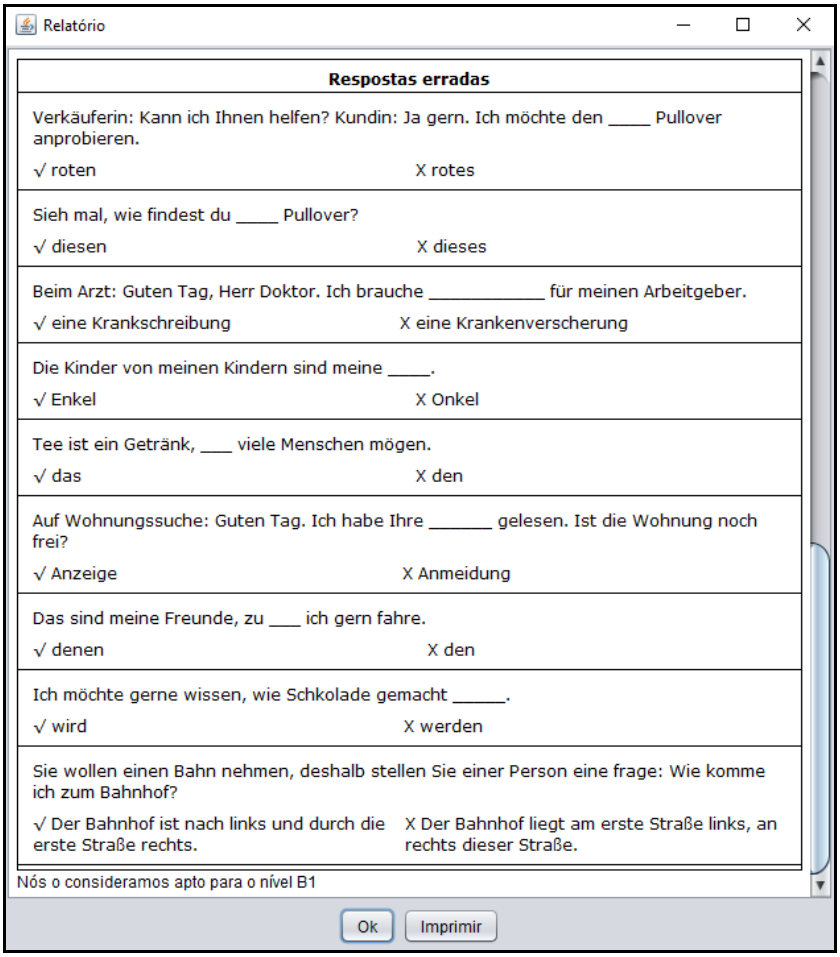
Figura 36 – Janela de relatório



Fonte: elaborado pelo autor.

A Figura 37 mostra o final da Janela de relatório com a lista das perguntas respondidas erradas junto da sua resposta correta. Abaixo das listas de perguntas é mostrado o resultado do nivelamento. O botão imprimir na parte de baixo da janela oferece a opção de emitir o relatório para ser impresso, a outra fecha a janela. O relatório é salvo automaticamente na pasta `data` do protótipo em formato HTML podendo ser aberto no navegador. Ao final é descrito o nível que o teste considera apto para o aluno estudar. Como o protótipo calcula esse resultado é detalhado no APÊNDICE B – Nivelamento

Figura 37 – Perguntas respondidas erradas na janela de relatório



Fonte: elaborado pelo autor.

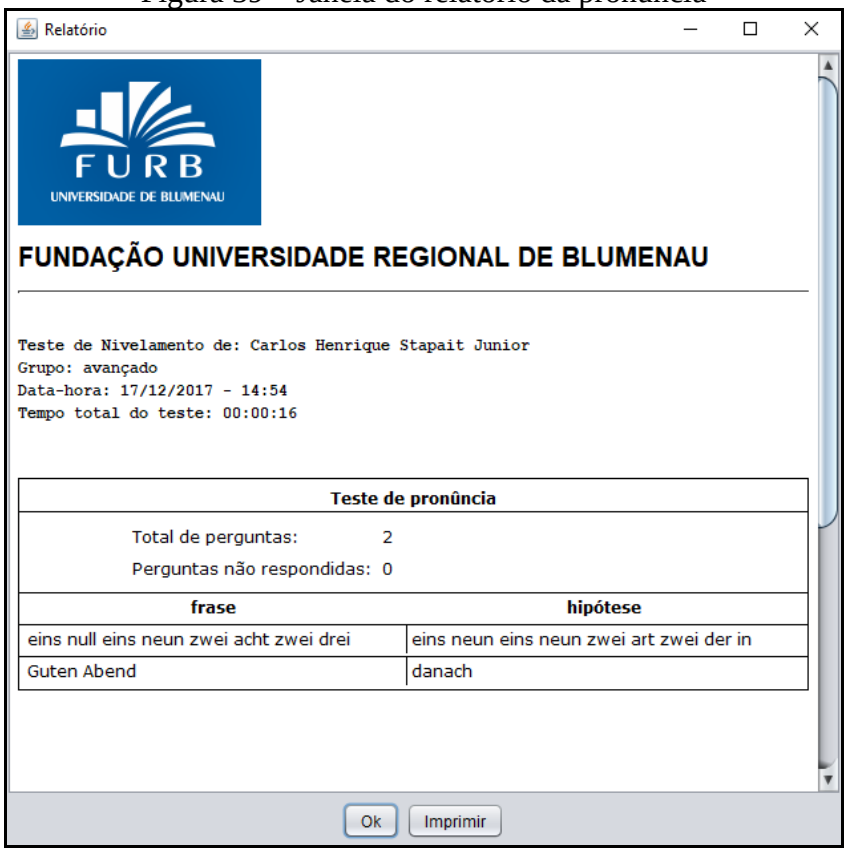
A Janela do teste de pronúncia é mostrada na Figura 38. Ela é idêntica à janela da Figura 35, exceto que a resposta para a pergunta é dada pelo microfone, e neste tipo de teste mesmo que o contador chegue a zero a resposta será avaliada. Para cada pergunta mostrada é criado um arquivo de áudio pela classe `AudioRecord` na pasta `data` do protótipo. Quando o botão avançar é clicado, a gravação é interrompida. Nisso é criada uma *thread*, implementada pela classe `ParallelRecognition` para fazer a interpretação de voz no arquivo gravado. Essa implementação da classe `ParallelRecognition` foi detalhada na página 37.

Figura 38 – Teste de pronúncia

Fonte: elaborado pelo autor.

A Janela de relatório do teste de pronúncia é mostrada na Figura 39. Ela também é idêntica à janela de relatório mostrada anteriormente, mas não mostra nenhuma estatística e sim o que era esperado como resposta e o que foi interpretado pela captura do microfone.

Figura 39 – Janela do relatório da pronúncia



Fonte: elaborado pelo autor.

A Figura 39 confirma o que foi exposto na seção 3.3.2, o reconhecimento de voz não alcançou os resultados esperados. Mas isso não justifica dizer que o teste de pronúncia no protótipo não tenha gerado resultados. Os arquivos de áudio gravados durante o teste ficam guardados e podem ser encontrados na pasta `data` do protótipo. Eles podem ser reproduzidos por qualquer reprodutor de arquivos .wav ou pela tela de cadastro de habilidades, mas ela não é própria para isso.

3.4 ANÁLISE DOS RESULTADOS

Esta seção trata dos resultados obtidos. Primeiro é descrito a apresentação do protótipo ao professor de alemão do FURB Idiomas. Depois é apresentado o feedback do experimento realizado com uma turma de alemão. Por fim é feita a comparação com os trabalhos correlatos.

3.4.1 Experimento de Validação

O protótipo, em uma primeira versão, foi apresentado ao professor Nestor Alberto Freese, do FURB Idiomas. Ele constatou que a funcionalidade do protótipo se aplica para avaliar os conhecimentos de quem estivesse na fase final de um módulo de um curso de alemão ou que tivesse contato com a língua. Ele registrou que o protótipo ficou simples e fácil de usar. O cadastro de habilidades possuía o diferencial de associar texto, imagem e áudio às perguntas, elementos importantes num curso de línguas.

Para a aplicação do experimento foram cadastradas no protótipo perguntas obtidas por download em formato .pdf (Portable Document Format) no site do Studio D (CORNELSEN, 2017). A turma do professor Nestor de nível B1 conseguiu responder corretamente a maioria das perguntas em menos de trinta minutos. O professor e os alunos comentaram que as perguntas de interpretação de texto e compreensão auditiva apareciam repentinamente no meio do teste, que era formado principalmente por questões de gramática. Foi sugerido então, dividir o teste em etapas e mostrar uma mensagem após cada etapa ser concluída. Na janela de relatório também foi sugerido mostrar a lista das perguntas respondidas erradas, junto com a resposta certa. O professor Nestor também sugeriu que as questões de interpretação de texto não tivessem tempo de resposta. Estas sugestões foram incorporadas em uma nova versão.

3.4.2 Comparativo com Trabalhos Correlatos e Considerações Finais

O Quadro 1 apresenta um comparativo entre o protótipo e os trabalhos correlatos, com as principais características e semelhanças. Verifica-se que o protótipo se demonstra mais

abrangente pois inclui o teste de pronúncia. Outro diferencial do protótipo foi a inclusão da síntese de voz junto da reprodução de áudios gravados. A gramática e o vocabulário são testados com questões de múltipla escolha. A avaliação da escrita não é abordada nem pelo protótipo, nem pelos trabalhos correlatos.

Quadro 1 – Comparativo dos Trabalhos Correlatos

teste de nivelamento / habilidades verificadas	Protótipo (2017)	Deutsche Welle (2015)	Goethe Institut (2016)
compreensão auditiva	+	+	+
Pronúncia	+/-	-	-
Leitura	+	+	+
Escrita	-	-	-
gramática e vocabulário	+	+/-	+

Fonte: elaborado pelo autor.

Durante o desenvolvimento deste trabalho foram encontradas dificuldades relacionadas ao suporte no idioma alemão de várias bibliotecas de processamento da linguagem natural que fossem compatíveis com Java. O MaryTTS e Sphinx foram as poucas alternativas encontradas, por que eles trabalham em cima de modelos estatísticos que descrevem de forma independente cada língua. O sintetizador de voz MaryTTS funcionou muito bem nos testes realizados com o protótipo, além de gratuito e de código aberto é feito totalmente em Java. O Sphinx é uma ferramenta que demonstra potencial, mas o modelo acústico da língua alemã fornecido no site da CMUSphinx, não apresenta ter qualidade. Identificou-se também uma limitação no formato do áudio que a ferramenta Sphinx trabalha. Conclui-se assim que o Sphinx pode não funcionar corretamente com algumas interfaces de captura de voz.

4 CONCLUSÕES

O objetivo do trabalho consistiu em construir um protótipo para avaliar o conhecimento no idioma alemão. O primeiro objetivo especifico de avaliar a gramática foi atingido permitindo o cadastro de perguntas de múltipla escolha. O segundo objetivo específico de avaliar a habilidade de leitura foi atingido permitindo a adição de texto junto às perguntas. O terceiro objetivo específico de avaliar a compreensão auditiva com síntese de voz também foi atingido com a utilização da biblioteca MaryTTS. O quarto objetivo específico era avaliar a capacidade de pronúncia e não foi alcançado com a biblioteca CMUSphinx. Neste caso a biblioteca Sphinx integrada ao protótipo transforma a voz capturada pelo microfone em texto. A biblioteca não possui recursos para medir o grau de confiança nas palavras encontradas e apresentou restrições quanto a frequência de som que ela trabalha. Tentativas de converter o arquivo de áudio para a frequência especificada pela ferramenta também não deram resultados. O modelo acústico para a língua alemã obtida no site da biblioteca CMUSphinx parece não estar devidamente treinado trazendo resultados distantes do que era esperado.

O protótipo atingiu os resultados esperados na avaliação dos conhecimentos em gramática, leitura e compreensão auditiva. Entretanto, são necessários estudos mais profundos a fim de avaliar a pronúncia sem a intervenção humana.

O *feedback* dos alunos e do professor Nestor Alberto Freese foi muito importante para o melhoramento do protótipo e todas as sugestões foram agregadas. O teste de proficiência faz uma ordenação na lista de perguntas e mostra mensagens a cada etapa do teste. As perguntas de interpretação de texto tiveram seu tempo estendido. E por fim, na Janela de relatório é exposta uma lista das perguntas respondidas erradas, mostrando a alternativa correta e qual o aluno selecionou erroneamente.

4.1 EXTENSÕES

As sugestões de extensões para trabalhos futuros estão listadas abaixo:

a) permitir a associação de vídeos no cadastro de temas;
b) permitir a reprodução de vídeos durante o teste;
c) permitir a reprodução de mais extensões de arquivos de áudio;
d) possibilitar testes de proficiência com somente questões de interpretação de texto, de gramática, compreensão auditiva ou pronúncia;
e) possibilitar a distinção de níveis por outro valor e não mais pelo atributo `code`;

f) acrescentar no cadastro de perguntas, um campo que justifique a resposta correta, e mostre essa explicação na tela de relatório ao lado das perguntas erradas.

REFERÊNCIAS

CMUSPHINX. **CMUSphinx**: open source speech recognition toolkit. [S.l.], 2017. Disponível em: <https://cmusphinx.github.io>. Acesso em: 21 set. 2017.

CORNELSEN. **Einstufungstests A1, A2 und B1**. [S.l], 2017. Disponível em: <https://www.cornelsen.de/studio_d/1.c.2583414.de>. Acesso em: 15 jul. 2017.

COSTA, Ericson S. et al. **Um sintetizador de voz baseado em HMMs livre**: dando novas vozes para aplicações livres no português do Brasil. In: WORKSHOP DE SOFTWARE LIVRE, 13., 2012, Porto Alegre. **Proceedings...** Porto Alegre: Associação Software Livre.Org., 2012. Não paginado. Disponível em: <http://wsl.softwarelivre.org/2012/0011/38.pdf>. Acesso em: 10 nov. 2016

COUNCIL OF EUROPA. **Common European Framework of Reference for Languages**: learning, teaching, assessment (CEFR). [S.l.], [2014?]. Disponível em: <http://www.coe.int/t/dg4/linguistic/Cadre1_en.asp>. Acesso em: 15 set. 2016.

COUNCIL OF EUROPA. **Common European Framework of Reference for Languages**: learning, teaching, assessment. [S.l.], [2014?]. Disponível em: <http://www.coe.int/t/dg4/linguistic/Source/Framework_EN.pdf>. Acesso em: 09 nov. 2016.

DEUTSCHE WELLE. **Einstufungstest**. [S.l.], 2015. Disponível em: <http://einstufungstest.dw.de/>. Acesso em: 15 set. 2016.

DFKI GMBH. **The Mary Text-To-Speech system (Mary TTS)**. [Saarbrücken], 2016. Disponível em: <http://mary.dfki.de/>. Acesso em: 19 set. 2016.

FURB IDIOMAS. **Nivelamento**. [Blumenau], [2016?]. Disponível em: <http://www.furb.br/web/1540/cursos/idiomas/nivelamento>. Acesso em: 30 ago. 2016.

GOETHE INSTITUT. **Testen sie ihr Deutsch**. [S.l.], 2016. Disponível em: <https://www.goethe.de/pt/spr/kup/tsd.html>. Acesso em: 8 set. 2016.

MCLELLAN, Charles. **How we learned to talk to computers, and how they learned to answer back.** [S.l.], 2016. Disponível em: <http://www.zdnet.com/article/how-we-learned-to-talk-to-computers-and-how-they-learned-to-answer-back/>. Acesso em: 10 nov. 2016.

OLIVEIRA, Rafael et al. Recursos para desenvolvimento de aplicativos com suporte a reconhecimento de voz para desktop e sistemas embarcados. In: WORKSHOP DE SOFTWARE LIVRE, 12., 2011, Porto Alegre. **Proceedings...** Porto Alegre: Associação Software Livre.Org, 2011. Não paginado. Disponível em: <wsl.softwarelivre.org/2011/0016/85180_1.pdf>. Acesso em: 20 set. 2016.

ORT, Ed; BHAKTI, Mehta. **Java Architecture for XML Binding (JAXB)**. [S.l.], 2017. Disponível em: <http://www.oracle.com/technetwork/articles/javase/index-140168.html>. Acesso em: 10 set. 2017.

ORACLE. **Java Documentation**: The Java™ Tutorials. [S.l.], 2013. Disponível em: <http://docs.oracle.com/javase/tutorial/index.html>. Acesso em: 08 set. 2017.

POST, Marlies. **TCC**. [mensagem pessoal]. Mensagem recebida por <chsjwarrior@gmail.com> em 25 ago. 2016.

UNIVERSIDADE DE COIMBRA. **Quadro Europeu Comum de Referência para as Línguas**. [Lisboa], 2014. Disponível em: <http://www.uc.pt/fluc/cl/diplomas/qecr>. Acesso em: 17 set. 2016.

APÊNDICE A – Quadro Europeu Comum de Referência para Línguas

A Figura 40 descreve de forma clara as competências e as capacidades comunicativas que os estudantes devem ter dentro dos seis níveis de proficiência do CEFR

Figura 40 – Descrição dos níveis do CEFR

Utilizador proficiente	C2 É capaz de compreender, sem esforço, praticamente tudo o que ouve ou lê. É capaz de resumir as informações recolhidas em diversas fontes orais e escritas, reconstruindo argumentos e factos de um modo coerente. É capaz de se exprimir espontaneamente, de modo fluente e com exactidão, sendo capaz de distinguir finas variações de significado em situações complexas. C1 É capaz de compreender um vasto número de textos longos e exigentes, reconhecendo os seus significados implícitos. É capaz de se exprimir de forma fluente e espontânea sem precisar de procurar muito as palavras. É capaz de usar a língua de modo flexível e eficaz para fins sociais, académicos e profissionais. Pode exprimir-se sobre temas complexos, de forma clara e bem estruturada, manifestando o domínio de mecanismos de organização, de articulação e de coesão do discurso.
Utilizador independente	B2 É capaz de compreender as ideias principais em textos complexos sobre assuntos concretos e abstractos, incluindo discussões técnicas na sua área de especialidade. É capaz de comunicar com um certo grau de espontaneidade e de à-vontade com falantes nativos, sem que haja tensão de parte a parte. É capaz de exprimir-se de modo claro e pormenorizado sobre uma grande variedade de temas e explicar um ponto de vista sobre um tema da actualidade, expondo as vantagens e os inconvenientes de várias possibilidades. B1 É capaz de compreender as questões principais, quando é usada uma linguagem clara e estandardizada e os assuntos lhe são familiares (temas abordados no trabalho, na escola e nos momentos de lazer, etc.) É capaz de lidar com a maioria das situações encontradas na região onde se fala a língua-alvo. É capaz de produzir um discurso simples e coerente sobre assuntos que lhe são familiares ou de interesse pessoal. Pode descrever experiências e eventos, sonhos, esperanças e ambições, bem como expor brevemente razões e justificações para uma opinião ou um projecto.
Utilizador elementar	A2 É capaz de compreender frases isoladas e expressões frequentes relacionadas com áreas de prioridade imediata (p. ex.: informações pessoais e familiares simples, compras, meio circundante). É capaz de comunicar em tarefas simples e em rotinas que exigem apenas uma troca de informação simples e directa sobre assuntos que lhe são familiares e habituais. Pode descrever de modo simples a sua formação, o meio circundante e, ainda, referir assuntos relacionados com necessidades imediatas. A1 É capaz de compreender e usar expressões familiares e quotidianas, assim como enunciados muito simples, que visam satisfazer necessidades concretas. Pode apresentar-se e apresentar outros e é capaz de fazer perguntas e dar respostas sobre aspectos pessoais como, por exemplo, o local onde vive, as pessoas que conhece e as coisas que tem. Pode comunicar de modo simples, se o interlocutor falar lenta e distintamente e se mostrar cooperante.

Fonte: Universidade de Coimbra (2014).

O CEFR fornece uma base transparente, coerente e compreensiva para a elaboração de programas e diretrizes curriculares para o ensino-aprendizado e avaliação de línguas estrangeiras. Os níveis do CEFR descrevem de forma abrangente o que os estudantes devem ser capazes de fazer e com que nível de qualidade. Segundo Council of Europe (2014), as descrições abrangem vários contextos, temas, tarefas e propósitos comunicativos ligados ao país onde o idioma é falado, um aprendizado baseado na ação.

O CEFR procura ser um ponto de referência e não um instrumento prático de ensino ou avaliação. Todavia, o CEFR apresenta opções entre diferentes tipos de avaliação, bem

como as vantagens e desvantagens de cada uma delas (COUNCIL OF EUROPE, 2014, p. 183). No contexto desse trabalho, tem-se os seguintes tipos de avaliação:

a) avaliação dos resultados: é a avaliação da consecução do que foi ensinado, medindo apenas o que foi determinado;
b) avaliação da proficiência: é uma avaliação do que se pode fazer ou do que se sabe, dando ao estudante a oportunidade de mostrar aquilo que conseguiu atingir;
c) avaliação do desempenho: consiste em o estudante produzir uma amostra de discurso oral ou escrito, sendo que o estudante pode escolher as palavras que quer usar para se expressar;
d) avaliação de conhecimentos: devem ser respondidas perguntas que podem ser retiradas de qualquer tema ou contexto para provar que o estudante tem domínio na língua, sendo que a avaliação pode usar palavras ligadas a um contexto que o estudante ainda não conhece;
e) avaliação objetiva: é uma avaliação onde as questões possuem apenas uma resposta correta dentre algumas alternativas;
f) avaliação subjetiva: as respostas são escritas pelo aluno, portanto uma avaliação subjetiva pode melhor avaliar o domínio de vocabulário e gramática que a avaliação objetiva.

APÊNDICE B – Nivelamento

A figura abaixo mostra o trecho de código onde é feito o cálculo do teste de nivelamento na classe `CompletTestView`. Na linha 482 implementado em código *lambda* é procurado numa lista, a classe `Answers` com o maior número de acertos. Como já foi exposto na seção 3.2.2.3 essa classe associa as respostas (classe `Answer`) a um nível. (`Level`). Na linha 487 é verificado se a porcentagem das respostas corretas é maior ou igual a 80. Se for verdadeiro o aluno pode se aprofundar no idioma em um nível maior, senão é recomendado maiores esforços nos estudos como descrito na linha 494.

Figura 41 – trecho de código

```
473             List<Level> levels = null;
474             try {
475                 levels = DataManager.getInstance().loadLevel();
476                 Collections.sort(levels);
477             } catch (IOException | JAXBException e) {
478                 showErrorMessage(e.getMessage(), e.getClass().getSimpleName());
479             }
480 
481             if (levels != null && !levels.isEmpty()) {
482                 Optional<Answers> optional = answers.stream()
483                         .max((a1, a2) -> a1.getCorrectPercent() + a2.getCorrectPercent());
484 
485                 if (optional.isPresent()) {
486                     Answers answers = optional.get();
487                     if (answers.getCorrectPercent() >= 80) {
488                         for (Level l : levels)
489                             if (l.compareTo(answers.getLevel()) == 1) {
490                                 rel.append("Nós o consideramos apto para o nível " + l.toString());
491                                 break;
492                             }
493                     } else
494                         rel.append("Nós o aconselhamos a reforçar o nível " + answers.getLevel().toString());
495                 }
496             }
```

Fonte: elaborado pelo autor.

APÊNDICE C – Captura de áudio pelo microfone

O trecho de código na figura abaixo mostra como é feito a captura de áudio no protótipo. Na linha 59 é especificado o formato do áudio na classe `AudioFormat` e na linha 65 é obtida a interface `TargetDataLine` da classe `AudioSystem`. Na linha 66 na interface `TargetDataLine` são introduzidos o objeto `AudioFormat` com o tamanho do *buffer* da gravação. E dentro do método `run` na linha 76 a classe `AudioSystem` faz a gravação dos *buffers* em disco.

Figura 42 – trecho de código da gravação do áudio

```
43      public void openRecordDevice(String encoding, float sambleRate, int sampleSize, int channels, boolean signed,
44              boolean bigEndian) throws LineUnavailableException, IllegalArgumentException {
45          AudioFormat.Encoding encode = AudioFormat.Encoding.ULAW;
46          if (encoding.equals(ENCODING_LINEAR)) {
47              if (signed)
48                  encode = AudioFormat.Encoding.PCM_SIGNED;
49              else
50                  encode = AudioFormat.Encoding.PCM_UNSIGNED;
51          } else if (encoding.equals(ENCODING_ALAW))
52              encode = AudioFormat.Encoding.ALAW;
53
54          if (channels < 1)
55              channels = 1;
56          else if (channels > 2)
57              channels = 2;
58
59          AudioFormat audioFormat = new AudioFormat(encode, sambleRate, sampleSize, channels, (sampleSize / 8) * channels,
60                  sambleRate, bigEndian);
61          DataLine.Info info = new DataLine.Info(TargetDataLine.class, audioFormat);
62          if (!AudioSystem.isLineSupported(info))
63              new LineUnavailableException("Microfone não esta disponível.");
64
65          targetDataline = (TargetDataLine) AudioSystem.getLine(info);
66          targetDataline.open(audioFormat, targetDataline.getBufferSize());
67      }
68
69      @Override
70      public void run() {
71          if (targetDataline == null)
72              return;
73
74          targetDataline.start();
75          try {
76              AudioSystem.write(new AudioInputStream(targetDataline), AudioFileFormat.Type.WAVE, path.toFile());
77              if (reproduceOnEnd) {
78                  running = true;
79                  openFile(path.toString());
80                  super.run();
81              }
82          } catch (IOException | LineUnavailableException | UnsupportedAudioFileException e) {
83              JOptionPane.showMessageDialog(null, e.getMessage(), e.getClass().getSimpleName(),
84                      JOptionPane.ERROR_MESSAGE);
85          }
86      }
```

Fonte: elaborado pelo autor.