

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

**TAGARELA: APLICATIVO DE COMUNICAÇÃO
ALTERNATIVA NA PLATAFORMA ANDROID**

DARLAN DIEGO DE MARCO

**BLUMENAU
2014**

2014/1-04

DARLAN DIEGO DE MARCO

**TAGARELA: APLICATIVO DE COMUNICAÇÃO
ALTERNATIVA NA PLATAFORMA ANDROID**

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Dalton Solano dos Reis, Mestre – Orientador

**BLUMENAU
2014**

2014/1-04

TAGARELA: APLICATIVO DE COMUNICAÇÃO

ALTERNATIVA NA PLATAFORMA ANDROID

Por

DARLAN DIEGO DE MARCO

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

Presidente: Prof. Dalton Solano dos Reis, M. Sc. – Orientador, FURB

Membro: Prof. Francisco Adell Péricas, M.Sc. – FURB

Membro: Prof. Aurélio Faustino Hoppe, M. Sc. – FURB

Blumenau, 08 de julho de 2014

Dedico este trabalho à família e a todos os amigos, especialmente aqueles que estiveram ao meu lado e acreditaram em mim em todos estes anos.

AGRADECIMENTOS

Aos meus pais e avós pelo apoio e confiança.

À minha companheira Priscila Küll pela compreensão, apoio e o incentivo.

Ao meu orientador Dalton Solano dos Reis pela dedicação, apoio e paciência a mim concedidos durante o desenvolvimento deste trabalho.

Aos meus amigos Alan e Lucas por terem trilhado o caminho acadêmico ao meu lado e principalmente pela grande amizade que conquistamos.

À toda a equipe do Centro Municipal de Educação Alternativa, por todo o apoio e contribuição durante os testes do aplicativo.

Aos pacientes que utilizaram o aplicativo e suas famílias, pela compreensão e auxílio.

À todos os professores da FURB que colaboraram direta ou indiretamente para a minha formação educacional e profissional.

A bondade é uma linguagem que o surdo
consegue ouvir e o cego consegue ler.

Mark Twain

RESUMO

Este trabalho apresenta o desenvolvimento de um aplicativo de comunicação alternativa para a plataforma Android. O aplicativo tem como principal objetivo reproduzir as funcionalidades da sua versão para iOS, permitindo que portadores de deficiências fonoarticulares aumentem sua capacidade de comunicação por meio do uso. O principal motivo desse aumento na capacidade de comunicação é dado pela utilização de planos de comunicação, que utilizam símbolos formados por recursos multimídia da plataforma Android como áudio e imagem. Durante a evolução deste trabalho foram utilizados vários recursos para o seu desenvolvimento, como a ferramenta de abstração de comandos SQL greenDao, as APIs de manipulação de recursos multimídia do Android e APIs para comunicação com servidores através de requisições HTTP. Como avaliação do trabalho foram realizados testes com pacientes portadores de deficiências de comunicação, testes de usabilidade do aplicativo, comparativo entre os trabalhos correlatos e utilização da memória e desempenho. Os resultados obtidos através da utilização do aplicativo por especialistas, permitem por meio de uma abordagem qualitativa concluir que o Tagarela pode auxiliar fonoaudiólogos e pacientes durante as sessões de terapia.

Palavras-chave: Comunicação alternativa. Android. Tecnologia assistiva.

ABSTRACT

This paper presents the development of an alternative communication application for Android platform. The app has as the main purpose to reproduce the functionalities of its iOS version, enabling people with special phono-articulatory needs to increase their communication capability through it. The main reason for the increasingly communication capability is given by the use of communication plans, which use symbols made up of multimedia resources within the Android platform, such as audio and interface image. During the elapse time of this paper many tools were used as resources in order to create it, such as the SQL abstraction layer greenDao, as well as Android multimedia manipulation APIs and client/server HTTP request communication APIs. As the paper evaluation, tests on patients with communication disabilities were conducted, as well as app usability tests, comparison between related jobs, memory usage and performance. The results obtained through the use of the application by professionals allow us to conclude that the Tagarela can assist audiologists and patients during therapy sessions.

Key-words: Alternative communication. Android. Assistive technology.

LISTA DE ILUSTRAÇÕES

Figura 1 – Símbolos Blissymbolics	20
Figura 2 – Símbolos PIC.....	20
Figura 3 – Símbolos PCS	21
Figura 4 – Prancha de comunicação e usuário do Tagarela.....	22
Figura 5 – Diagrama de estados de um objeto MediaPlayer	24
Figura 6 – Diagrama de estados da classe MediaRecorder.....	26
Figura 7 – Abstração realizada pelo greenDao ao acessar o SQLite	27
Figura 8 – Comparação de operações entre greenDao e ORMLite em número de registros...	28
Figura 9 – Tela inicial do TapToTalk	29
Figura 10 – Tela de edição do JABTalk	30
Figura 11 – Tela inicial do Proloquo2Go	31
Figura 12 – Casos de uso do Tagarela referentes ao <i>login</i> no aplicativo.....	34
Figura 13 – Casos de uso do Tagarela referentes a criação de dados do usuário	34
Figura 14 – Casos de uso do Tagarela referentes a visualização dos dados do usuário	35
Figura 15 – Diagrama de pacotes do Tagarela.....	36
Figura 16 – Diagrama de classes das classes modelo do Tagarela	37
Figura 17 – Classes DAO do Tagarela	38
Figura 18 – Diagrama de classes das classes view.dialogs do Tagarela	39
Figura 19 – Diagrama de classes das classes view.dialogs do Tagarela (continuação) ..	39
Figura 20 – Diagrama de classes das classes do pacote view.activities.....	42
Figura 21 – Principais classes do pacote controler	43
Figura 22 – AsyncTasks do pacote controler	44
Figura 23 – Diagrama de classes das classes do pacote utils.....	45
Figura 23 – Diagrama de classes do pacote interfaces	46
Figura 24 – Classes do pacote adapter.....	46
Figura 25 – Diagrama de sequência do processo de criação de símbolos	48
Figura 26 – Diagrama de sequência do processo de criação de planos	49
Figura 27 – Diagrama de atividades do uso padrão do Tagarela	50
Figura 28 – Modelo de entidade e relacionamento da tabela local.....	51
Quadro 1 – Código fonte do método recordAudio	53

Quadro 2 – Código fonte do método <code>startPlaying()</code>	54
Quadro 3 – Código fonte da classe <code>SyncCreatedSymbolTask</code>	55
Figura 29 – Janela de boas vindas do Tagarela.....	58
Figura 30 – Janelas de <i>login</i> e criação de usuário.....	58
Figura 31 – Janela de seleção de tipo de usuário	58
Figura 32 – Janela principal do Tagarela	59
Figura 33 – Janelas para a criação de símbolo.....	59
Figura 34 – Janela de visualização de símbolos	60
Figura 35 – Janela de seleção de layout.....	61
Figura 36 – Tela de criação de planos	61
Figura 37 – Tela de utilização de planos	62
Figura 38 – Janela de observações.....	63
Figura 39 – Janela de histórico de símbolos	63
Figura 40 – Player do Tagarela apresentando um vídeo associado	64
Figura 41 – Comparação das janelas iniciais do aplicativo	68
Figura 42 – Comparação das janelas de seleção de tipo de usuário	68
Figura 43 – Comparação das janelas de <i>login</i> de usuário	68
Figura 44 – Comparação das janelas de criação de usuário.....	68
Figura 45 – Comparação das telas principais do aplicativo.....	69
Figura 46 – Comparação das janelas de seleção de categorias de símbolo	69
Figura 47 – Comparação das janelas de criação de símbolos do aplicativo	69
Figura 48 – Comparação das janelas de visualização de símbolos.....	69
Figura 49 – Comparação das janelas de seleção de <i>layout</i> para novo plano	70
Figura 51 – Comparação das telas de criação de planos com <i>layout</i> 3 x 3	70
Figura 52 – Comparação das telas de utilização de planos.....	70
Quadro 4 – Comparação do Tagarela em Android com as versões iOS e trabalhos correlatos	71
Figura 53 – Consumo de memória durante a utilização do aplicativo Tagarela.....	72
Figura 54 – Tempo de execução de sincronização de itens do aplicativo	72
Quadro 5 – Caso de uso Criar usuários.....	81
Quadro 6 – Caso de uso Criar símbolos	81
Quadro 7 – Caso de uso Criar Planos	82
Quadro 8 – Caso de uso criar observação.....	82
Quadro 9 – Caso de uso Selecionar o layout do plano.....	82

Quadro 10 – Caso de uso Visualizar lista de planos.....	83
Quadro 11 – Caso de uso Visualizar símbolos.....	83
Quadro 12 – Caso de uso Interagir com símbolos	83
Quadro 13 – Caso de uso Exibir borda em símbolos.....	83
Quadro 14 – Caso de uso Reproduzir som associado ao símbolo	84
Quadro 15 – Caso de uso Registrar histórico de símbolos utilizados ...	84
Quadro 16 – Caso de uso Visualizar observações	84
Quadro 17 – Caso de uso Visualizar histórico de símbolos utilizados.	84
Quadro 18 - Caso de uso Realizar login.....	85
Quadro 19 – Caso de uso Receber os planos do usuário armazenados no servidor	85
Quadro 20 – Caso de uso Receber os símbolos armazenados no servidor.	85
Quadro 21 – Caso de uso Receber dados do usuário armazenados no servidor	86
Quadro 22 – Caso de uso Receber as observações do servidor.....	86
Quadro 23 – Caso de uso Receber o histórico de símbolos.....	86
Quadro 24 – Caso de uso Enviar histórico de símbolo para o servidor.	86
Quadro 25 – Caso de uso Enviar dados do usuário para o servidor.....	87
Quadro 26 – Caso de uso Enviar símbolo para o servidor	87
Quadro 27 – Caso de uso Enviar plano para o servidor.....	87
Quadro 28 – Caso de uso Enviar plano para o servidor.....	87
Quadro 29 – Caso de uso Visualizar planos	87
Quadro 30 – Caso de uso Visualizar vídeos do youtube.....	88
Figura 55 – Formulário preenchido pela fonoaudióloga Luana Viola da Cruz	89
Figura 56 – Formulário preenchido pela fonoaudióloga Luana Viola da Cruz (continuação)	89
Figura 57 – Formulário preenchido pela fonoaudióloga Ana Maria Philipps dos Santos	90
Figura 58 – Formulário preenchido pela fonoaudióloga Ana Maria Philipps dos Santos (continuação).....	90
Figura 59 – Formulário preenchido pela fonoaudióloga Binca Kleis de Carvalho	91
Figura 60 – Formulário preenchido pela fonoaudióloga Binca Kleis de Carvalho (continuação)	91
Figura 61 – Caderno de comunicação.....	92
Figura 62 – Caderno de comunicação (continuação).....	92

Figura 63 – Caderno de comunicação (continuação).....	92
Figura 64 – Criança com autismo utilizando o Tagarela	93

LISTA DE SIGLAS

AAC - *Advanced Audio Coding*

ADA - *American with Disabilities ACT*

CA – Comunicação Alternativa

CAA – Comunicação Alternativa e Aumentativa

API – *Application Programming Interface*

CPU – *Central Processing Unit*

DAO – Data Access Object

GPU – *Graphics Processing Unit*

IDE – *Integrated Development Environment*

JSON - *JavaScript Object Notation*

PCA – Programa de Comunicação Alternativa

PCS – *Picture Communication Symbols*

PIC – *Pictogram Ideogram Communication*

SDK – *Software Development Kit*

RF – Requisitos Funcionais

RNF – Requisitos Não Funcionais

TA – Tecnologia Assistiva

UC – *Use Case*

UML - *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO	15
1.1 OBJETIVOS DO TRABALHO.....	16
1.2 ESTRUTURA DO TRABALHO	16
2 FUNDAMENTAÇÃO TEÓRICA	17
2.1 TECNOLOGIA ASSISTIVA	17
2.1.1 Categorias de tecnologia assistiva	18
2.2 COMUNICAÇÃO ALTERNATIVA E AUMENTATIVA	18
2.2.1 Ferramentas de comunicação alternativa e aumentativa	18
2.2.1.1 Sistemas de Símbolos Gráficos.....	19
2.2.1.2 Técnicas de Seleção	21
2.3 TAGARELA	21
2.4 ANDROID	22
2.4.1 Reprodução e gravação de mídias no Android.....	23
2.5 GREENDAO	27
2.6 TRABALHOS CORRELATOS	29
2.6.1 TapToTalk	29
2.6.2 JABTalk.....	30
2.6.3 Proloquo2Go.....	31
3 DESENVOLVIMENTO DO APLICATIVO.....	32
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO	32
3.2 ESPECIFICAÇÃO.....	33
3.2.1 Casos de uso	33
3.2.2 Estrutura do projeto	35
3.2.2.1 Pacote br.com.furb.tagarela.model	36
3.2.2.2 Pacote br.com.furb.tagarela.view	39
3.2.2.3 Pacote br.com.furb.tagarela.controler	43
3.2.2.4 Pacote br.com.furb.tagarela.utils	44
3.2.2.5 Pacote br.com.furb.tagarela.interface	46
3.2.2.6 Pacote br.com.furb.tagarela.adapter	46
3.2.2.7 Pacote br.com.furb.tagarela.listeners	47
3.2.3 Diagramas de sequência	47

3.2.3.1 Diagrama de sequência Criar símbolos	47
3.2.3.2 Diagrama de sequência Criar planos.....	49
3.2.4 Diagrama de Atividades	50
3.2.5 Diagrama MER.....	51
3.3 IMPLEMENTAÇÃO	52
3.3.1 Técnicas e ferramentas utilizadas	52
3.3.2 Gravação e reprodução de áudio	52
3.3.3 Envio de informações ao servidor	54
3.3.4 <i>Design Patterns</i>	57
3.3.5 Operacionalidade da implementação.....	57
3.3.5.1 Criação de usuários	57
3.3.5.2 Tela principal	59
3.3.5.3 Criação de símbolos	59
3.3.5.4 Visualização dos símbolos disponíveis	60
3.3.5.5 Criação de planos	60
3.3.5.6 Utilização de pranchas	62
3.3.5.7 Visualização e criação de observação	62
3.3.5.8 Histórico de símbolos.....	63
3.3.5.9 Visualização de vídeos do youtube	64
3.4 RESULTADOS E DISCUSSÃO	64
3.4.1 Utilização da aplicação em ambiente real	65
3.4.2 Comparação de interface gráfica	68
3.4.3 Comparação entre o trabalho desenvolvido e os trabalhos correlatos.....	70
3.4.4 Utilização de memória e desempenho	71
4 CONCLUSÕES	73
4.1 EXTENSÕES.....	73
REFERÊNCIAS BIBLIOGRÁFICAS.....	75
APÊNDICE A – Descrição dos casos de uso, cenários e fluxos dos casos de uso	77
APÊNDICE B – Formulários sobre experiência de utilização do Tagarela	89
APÊNDICE C – Caderno de comunicação.....	92
APÊNDICE D – Utilização do Tagarela por usuário com autismo.....	93

1 INTRODUÇÃO

As realidades referentes à pessoa com deficiência ainda são bastante desconhecidas da população em geral. Segundo as nações unidas, 10% da população mundial é composta de pessoas com algum tipo de deficiência (NATIONS, 2013). No Brasil, esse número aumenta para 23,9% da população nacional, em torno de 45 milhões de brasileiros nos dias de hoje, segundo o Instituto Brasileiro de Geografia e Estatística (OLIVEIRA, 2012), sendo que a maior proporção se encontra no Nordeste (26,63%) e a menor no Sul (22,50%).

Com o intuito de auxiliar alguns tipos de deficiências existem as Tecnologias Assistivas (TAs). Segundo Bersch et al. (2007 apud BERSCH, 2007, p. 27), Tecnologia Assistiva (TA) "Deve ser entendida como um auxílio que promoverá a ampliação de uma habilidade funcional deficitária ou possibilitará a realização da função desejada e que se encontra impedida por circunstância de deficiência". A TA é composta de recursos e serviços. Os recursos são as ferramentas utilizadas pela pessoa portadora de deficiência e os serviços são aqueles auxiliam o seu desenvolvimento.

O uso de TA envolve as pessoas com deficiência em situações que as desafia a explorar e conhecer a ferramenta. Estas situações auxiliam a construir novos conhecimentos (BERSCH et al., 2007).

Entre os vários tipos de deficiências existem as limitações fonoarticulares, nas quais a comunicação através da fala pode ser alterada ou ausente. Esta dificuldade de comunicação complica os processos de aprendizagem das crianças e estas passam a ser percebidas como deficientes intelectuais. Desta forma, para que uma criança possa acessar o conhecimento e interagir com o meio em que vive é necessário que ela tenha um atendimento especializado que a ajude a desenvolver sua comunicação e mobilidade (BERSCH et al., 2007).

A assistência para deficiências de comunicação no contexto de TA é dada através de uma categoria chamada Comunicação Alternativa e Aumentativa (CAA), que busca auxiliar pessoas com dificuldades de fala ou escrita a ampliar suas habilidades de comunicação.

Ao identificar a necessidade de aplicativos para auxiliar pessoas com deficiência de fala o aluno Alan Filipe Cardozo Fabeni desenvolveu um aplicativo de Comunicação Alternativa (CA) chamado Tagarela. Posteriormente, o Tagarela foi transformado em um projeto da Universidade Regional de Blumenau com o objetivo de desenvolver uma plataforma que disponibilizar uma forma de Comunicação Alternativa aos pacientes e também facilitar a troca de experiência e informações entre as pessoas envolvidas com o paciente (TAGARELA, 2014).

No exposto acima, torna-se evidente a necessidade de ferramentas de CAA e o auxílio que o projeto Tagarela pode trazer a pessoas com deficiência de fala. Motivado por estes fatores, neste trabalho é criado o aplicativo Tagarela para a plataforma Android. O aplicativo desenvolvido explora diversos recursos disponíveis para a plataforma com a intenção de tornar a experiência do usuário imersiva e interativa. Além das funcionalidades desenvolvidas por Fabeni(2012), foram implementados os recursos de sincronização de informações entre dispositivos, o controle de conectividade do dispositivo com a internet, a possibilidade de customização de *layouts* de planos e uma redefinição da interface para o usuário.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é disponibilizar o aplicativo Tagarela na plataforma Android.

Os objetivos específicos do trabalho são:

- a) propiciar um ambiente que permita a customização de símbolos e planos para os usuários;
- b) propiciar a interoperabilidade entre dispositivos através da sincronização de informações por usuário;
- c) propiciar imersão ao usuário do aplicativo através da utilização de recursos multimídia disponibilizados pela plataforma Android;
- d) criar um ambiente que permita ao usuários guardar informações sobre a experiência de utilização do aplicativo e o acompanhamento do progresso na utilização da ferramenta.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos. O capítulo um apresenta uma introdução ao tema abordado, os objetivos e a estrutura deste trabalho.

O capítulo dois contém a fundamentação teórica necessária para permitir um melhor entendimento sobre este trabalho.

O capítulo três apresenta o desenvolvimento do aplicativo, contemplando também os seus requisitos e especificação contendo os casos de uso, diagramas. Neste capítulo são apresentadas as ferramentas utilizadas na implementação, os resultados e discussões.

O capítulo quatro refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo é dividido em seis seções. A seção 2.1 apresenta os conceitos de TA e qual sua importância. A seção seguinte, 2.2 descreve a categoria de TA escolhida para a criação do trabalho. Na seção 2.3 é apresentado o aplicativo Tagarela. A seção 2.4 apresenta a plataforma Android, com detalhes sobre sua arquitetura e componentes. Na seção 2.5 é apresentado o greenDAO ferramenta utilizada para manipular dados no Android. Por fim, a sessão 2.6 apresenta alguns aplicativos correlatos ao aqui desenvolvido.

2.1 TECNOLOGIA ASSISTIVA

Muitos autores escreveram sobre o uso de tecnologias para pessoas com deficiência. Radabaugh (1993) afirma que “Para as pessoas sem deficiência a tecnologia torna as coisas fáceis. Para as pessoas com deficiência, a tecnologia torna as coisas possíveis.” Segundo Cook e Hussey (1995, apud SARTORETTO; BERSCH, 2012) tecnologias assistivas podem ser entendidas como uma grande variedade de equipamentos, serviços, estratégias e práticas criadas e aplicadas para diminuir problemas funcionais sofridos por pessoas com deficiência. Em outras palavras, TA pode ser entendida como um meio de promover, suprir ou prover uma habilidade funcional para pessoas deficientes através de recursos e serviços.

O texto da *American with Disabilities ACT* (ADA, 1990) descreve recursos e serviços em tecnologia assistiva da seguinte forma:

Recurso é todo e qualquer item, equipamento ou parte dele, produto ou sistema fabricado em série ou sob medida, utilizado para aumentar, manter ou melhorar as capacidades funcionais das pessoas com deficiência. Serviços são definidos como aqueles que auxiliam diretamente uma pessoa com deficiência a selecionar, comprar ou usar os recursos acima definidos. (ADA, 1990).

Recursos são ferramentas que vão possibilitar ao deficiente exercer alguma ação ou ajudá-lo a explorar alguma habilidade. Alguns exemplos de recursos podem ser: teclados adaptados, bengalas, brinquedos ou, mesmo no âmbito tecnológico, softwares especializados que auxiliem o deficiente a exercer uma ação. Serviços são profissionais que auxiliam os deficientes a desenvolverem-se fazendo uso das técnicas e recursos providos pela TA.

Bersch et al. (2007, p. 28) transcreve que o objetivo das tecnologias assistivas é oferecer à pessoa com deficiência certa independência, melhorar a qualidade de vida e inclusão social, através do auxílio e ampliação da comunicação, mobilidade, controle do seu ambiente, habilidades de seu aprendizado e trabalho.

2.1.1 Categorias de tecnologia assistiva

Para diferenciar a linha de pesquisa e direcionar a catalogação das TAs, ela é organizada em diversas categorias. Desta forma é possível criar um banco de dados para a identificação de recursos apropriados para o atendimento de uma determinada necessidade (SARTORETTO; BERSCH, 2012). Algumas dessas categorias são:

- a) auxílios para a vida diária e prática;
- b) comunicação alternativa e aumentativa;
- c) recurso de acessibilidade ao computador;
- d) adequação postural;
- e) auxílio de mobilidade;
- f) adaptação de veículos.

2.2 COMUNICAÇÃO ALTERNATIVA E AUMENTATIVA

Pelosi (2011) diz que a CAA é uma categoria de TA voltada para a ampliação das habilidades de comunicação. A CA busca auxiliar pessoas sem fala ou escrita funcional na sua necessidade comunicativa e na sua habilidade de falar e/ou escrever, utilizando outros canais de comunicação diferentes da fala. Através destes, a pessoa com deficiência pode expressar suas opiniões, vontades e necessidades.

De acordo com Bersch et al. (2007), a diferença entre comunicação aumentativa e comunicação alternativa é dada através do grau de necessidade da ferramenta de comunicação. A comunicação é alternativa quando é utilizada uma ferramenta de CAA para se comunicar ao invés da fala, devido à impossibilidade de articular ou produzir sons adequadamente. A comunicação é considerada aumentativa quando o sujeito utiliza de outro meio para complementar ou compensar deficiências que a fala apresenta, mas sem substituí-la completamente.

2.2.1 Ferramentas de comunicação alternativa e aumentativa

Ferramentas de CAA são recursos de TA que auxiliam a comunicação existente ou substituem a fala.

Segundo Bersch et al. (2007), entre as ferramentas mais comuns de CA estão as pranchas de comunicação. Em uma prancha são colocados vários símbolos gráficos. Esses símbolos são definidos de acordo com as necessidades de cada indivíduo e representam alguma ação ou objeto do mundo real. Desta forma o usuário pode se comunicar com outras pessoas indicando de alguma forma o símbolo desejado.

Para formar sentenças complexas, cada símbolo deve ter um significado relevante, dessa forma ao combinar símbolos uma pessoa pode expressar seus desejos e vontades (BERSCH et. al., 2007).

Segundo França (2012), a utilização de CA é realizada através de um Programa de Comunicação Alternativa (PCA). O PCA é composto por quatro estágios que o paciente deve passar para utilizar a CA de forma eficaz. O primeiro estágio é referente a memorização de símbolos, nesta etapa as imagens do símbolo são apresentadas ao paciente sempre quando ele realiza uma ação. Por exemplo, ao entregar um copo com água a um paciente, é mostrado junto do copo uma figura que represente a água. Desta forma, a associação da imagem com a ação é realizada. O segundo estágio é a memorização de sentenças simples que representam a ação que o paciente estiver realizando. Um exemplo seria a apresentação dos símbolos: Eu, Quero e Água, no momento em que o paciente pedir água. O terceiro estágio é permitir ao paciente realizar escolhas simples através de uma única prancha. Por exemplo, a utilização de pranchas específicas para alimentação ou higiene. O quarto estágio é quando o paciente começa a utilizar pranchas para construir um diálogo, de forma que haja a interação comunicativa efetiva entre o paciente e outra pessoa.

Ao criar pranchas de CAA, deve ser escolhido o tipo de símbolos gráficos que essa ferramenta irá utilizar. Esses símbolos podem ser classificados em símbolos pictográficos, símbolos arbitrários e símbolos ideográficos.

Símbolos pictográficos são símbolos que parecem com aquilo que desejam simbolizar. Os símbolos arbitrários são desenhos que não tem relação entre a forma e aquilo que simbolizam. Por fim, os símbolos ideográficos simbolizam ideias que relacionam uma associação gráfica entre o símbolo e o conceito que representa (SARTORETTO; BERSCH, 2012).

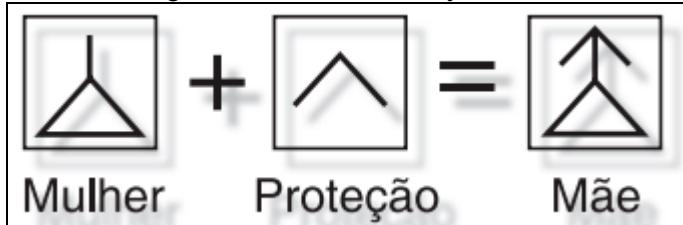
2.2.1.1 Sistemas de Símbolos Gráficos

Segundo Bersch et al. (2007), existem diferentes tipos de sistemas simbólicos reconhecidos internacionalmente e utilizados para a confecção de pranchas de comunicação. Dentre esses os principais são: *Picture Communication Symbols*, *Blissymbolics*, *Pictogram Ideogram Communication* (PIC).

O sistema *Blissymbolics* é um conjunto de símbolos, em sua maioria ideográficos. Foi criado por Charles K. Bliss para facilitar a comunicação internacional de pessoas. Sua primeira aparição foi em 1971.

Estes símbolos são construídos por meio da utilização de símbolos simples e abstratos. Os símbolos são organizados sintaticamente em uma prancha, tendo cada grupo sintático uma correspondência específica. A Figura 1 apresenta alguns símbolos desse sistema.

Figura 1 – Símbolos Blissymbolics

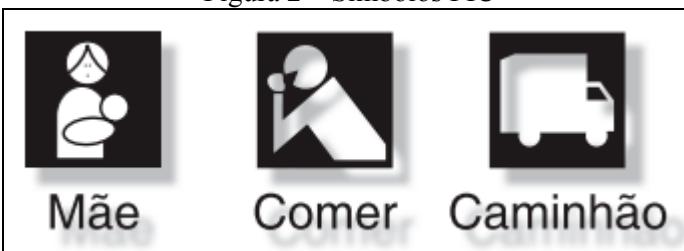


Fonte: Bersch et. al. (2007).

O sistema PIC originou-se no Canadá para diminuir as dificuldades na discriminação de figura-fundo. No Canadá, Estados Unidos, Noruega, Dinamarca, Portugal e Brasil, onde o PIC é usado, acabou substituindo o sistema *Blissymbolics* para usuários que não tenham demonstrado avanços com o sistema.

O PIC é composto por 400/800 símbolos pictográficos em contraste (branco/preto), separados por campos semânticos, onde todos os usuários têm a mesma prancha respeitando sua organização. Devido ao seu tipo de desenho, o PIC é indicado principalmente a indivíduos que apresentem dificuldades visuais, visto que o contraste proporciona maior clareza para usuários de baixa visão. Na Figura 2 são apresentados alguns símbolos PIC.

Figura 2 – Símbolos PIC

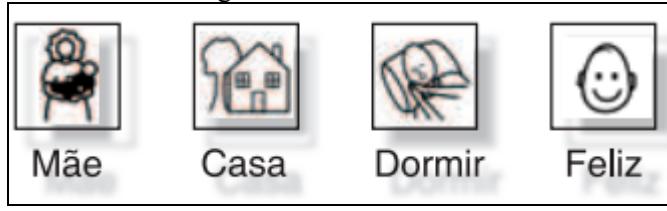


Fonte: Bersch et. al. (2007).

Os símbolos de comunicação pictórica (PCS) foram idealizados em 1980 pela fonoaudióloga Roxanna Mayer Johnson. Este sistema é composto por aproximadamente 8 mil símbolos que podem representar uma grande variedade de vocabulário (BERSCH et al., 2007). Estes símbolos, por sua facilidade de compreensão, são geralmente usados por crianças ou por indivíduos que possuem dificuldade em compreender representações mais abstratas.

Por ser um sistema aberto e graças a sua possibilidade de customização o PCS é um sistema largamente utilizado em pranchas e softwares no Brasil. Pois pode ser adaptado para questões regionais, culturais e pessoais do usuário. A Figura 3 apresenta alguns símbolos do sistema PCS.

Figura 3 – Símbolos PCS



Fonte: Bersch et. al. (2007).

2.2.1.2 Técnicas de Seleção

Segundo Pelosi (2011), é importante saber qual a técnica de seleção mais eficiente para cada usuário de um aplicativo de CA, identificando qual o melhor posicionamento da ferramenta e do usuário. Pelosi (2011) cita que existem três técnicas de seleção, sendo elas:

- seleção direta: é o método mais rápido e pode ser utilizado através do apontar do dedo ou outra parte do corpo, com uma ponteira de cabeça ou uma luz fixada à cabeça por exemplo;
- técnica de varredura: exige que o indivíduo tenha uma resposta voluntária consistente, como piscar os olhos, balançar a cabeça, sorrir ou emitir um som para sinalizar a sua resposta. Em dispositivos com hardware limitado o usuário vai necessitar de um facilitador para apontar os símbolos. Os métodos de varredura podem ser linear, circular, de linhas e colunas ou blocos;
- técnica da codificação: permite a ampliação de significados a partir de um número limitado de símbolos e o aumento de velocidade. É uma técnica bastante eficiente para usuários com dificuldades motoras graves.

De acordo com França (2012), para um software de CA é importante que algumas propriedades de seleção sejam atribuídas dinamicamente e o software seja ajustado individualmente para cada usuário.

2.3 TAGARELA

O Tagarela é um aplicativo de CA desenvolvido originalmente para iOS. De acordo com Fabeni (2012). Este aplicativo tem como objetivo fornecer uma plataforma para que as pessoas envolvidas no processo de comunicação do usuário deficiente interajam entre si e que exista uma evolução na capacidade comunicativa deste paciente. Isso ocorre através de planos de atividades elaborados pelo fonoaudiólogo em conjunto com o tutor do paciente.

Além das funcionalidades básicas de criação e utilização de símbolos, o Tagarela destaca-se pela facilidade de criação de planos através da interação entre os diversos atores envolvidos no processo de comunicação do usuário deficiente e também pela sincronização

dos dados entre esses usuários através da *web*, de forma que as informações relevantes estejam disponíveis a eles em tempo real (FABENI, 2013).

Segundo Fabeni (2012, p. 85), na etapa de validação, o aplicativo foi utilizado em sessões de terapia entre paciente e fonoaudiólogo, obtendo bons resultados e trazendo benefícios reais ao paciente, que a longo prazo, podem significar melhorias significativas à sua evolução na capacidade de se comunicar. A Figura 4 apresenta a prancha de comunicação criada pelo aplicativo e também o Tagarela em uso pelo paciente e seu especialista.

Figura 4 – Prancha de comunicação e usuário do Tagarela



Fonte: Acervo pessoal

2.4 ANDROID

O Android é uma plataforma para dispositivos móveis que inclui aplicações de um sistema operacional, *middleware* e aplicativos chave, como agenda, telefone, câmera, galeria e outros. O Android *Software Development Kit* (SDK) fornece as ferramentas e APIs necessárias para começar a desenvolver aplicações para a plataforma Android usando a linguagem de programação Java (GOOGLE, 2012b). Segundo Google (2012b), o Android tem seu sistema operacional dividido em quatro camadas (*Linux kernel*, *Libraries*, *Application framework* e *Applications*).

O *kernel* Linux utilizado pelo Android, na versão 2.6, é responsável por serviços centrais do sistema, como segurança, gerenciamento de memória, gestão de processos, pilha de rede e modelo de *driver*. O *kernel* também atua como uma camada de abstração entre o hardware e a camada de software (GOOGLE, 2012b).

A camada *Libraries* é implementada em C/C++ e contém um conjunto de bibliotecas utilizadas por diversos componentes do sistema Android. Estas bibliotecas são expostas a desenvolvedores através da *Application framework* (GOOGLE, 2012b).

A *Application framework* é composta por *Application Programmer Interfaces* (APIs) desenvolvidas em Java para abstrair o uso das bibliotecas da camada *Libraries*. Estas APIs

invocam as bibliotecas da segunda camada através das interfaces *Java Native Interface* (JNI) (GOOGLE, 2012b).

Na última camada, *Applications*, estão os aplicativos utilizados diretamente pelos usuários finais, como jogos, calculadora, agenda telefônica e outros. Igualmente aos sistemas operacionais para computadores comuns, são estes softwares que mais interessam ao usuário final, pois ele estará diariamente utilizando os mesmos (GOOGLE, 2012b).

2.4.1 Reprodução e gravação de mídias no Android

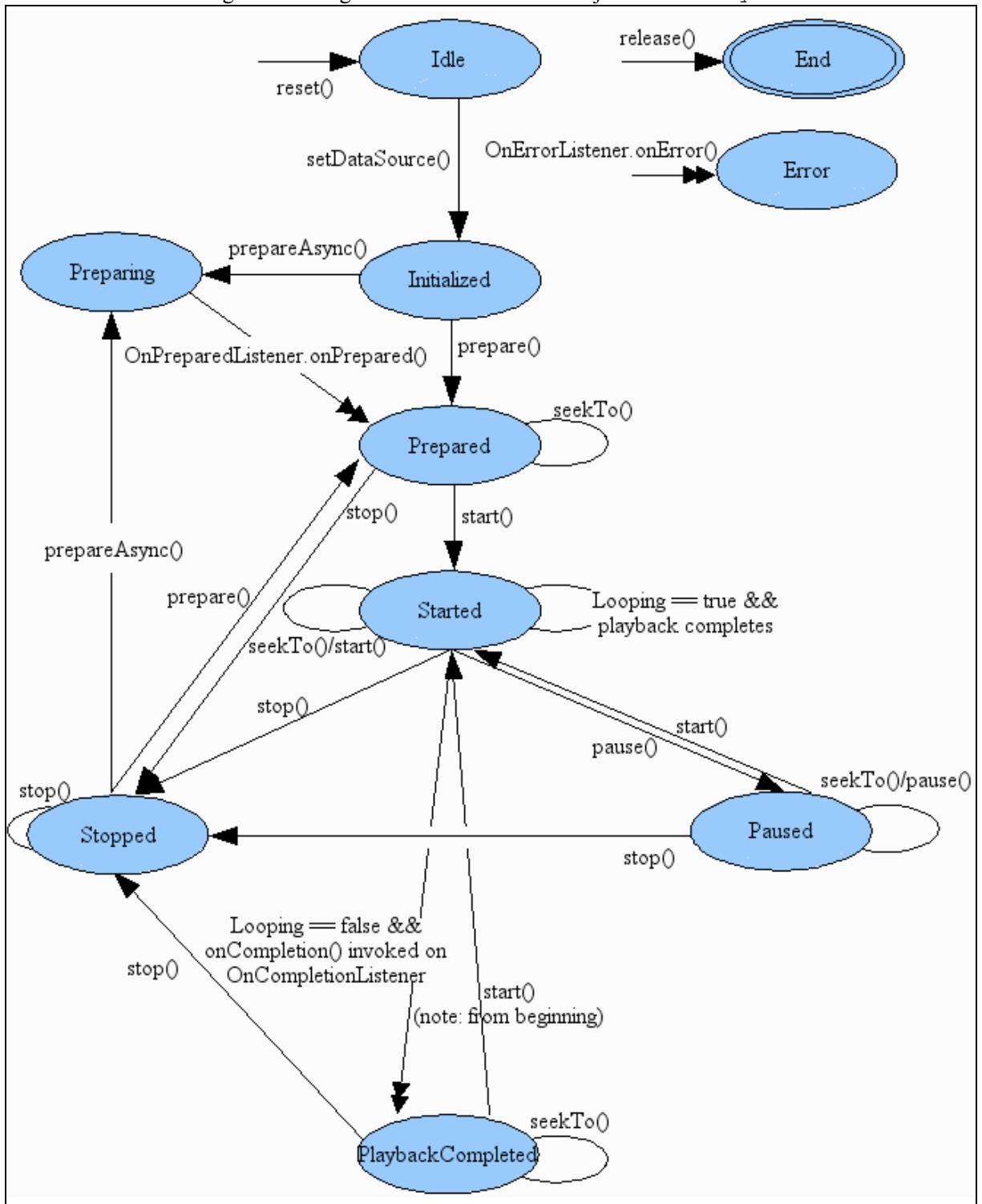
A plataforma Android possui um sistema de codificação/decodificação para uma variedade de multimídias comuns para integração de áudio, vídeo e imagens dentro das aplicações.

O Android permite reproduzir áudio e vídeo de diversos formatos e também vários tipos de fontes de dados, entre eles, reproduzir a partir de arquivos armazenados na pasta `raw` da aplicação, a partir de arquivos autônomos no sistema de arquivos, ou de um *stream* através de uma conexão de rede. Para reproduzir o áudio ou vídeo usa-se a classe `MediaPlayer`. (GOOGLE, 2012b)

O Android utiliza basicamente duas classes para reproduzir mídia de áudio. São elas: `MediaPlayer` e `AudioManager`. A classe `MediaPlayer` é a classe primária para reprodução de áudio e vídeo. A classe `AudioManager` gerencia a reprodução nos dispositivos de saída do aparelho.

O controle de reprodução de áudio, vídeo e *streamming* é realizado como uma máquina de estados. A Figura 5 mostra o ciclo de vida de um objeto `MediaPlayer` e os possíveis estados que o mesmo pode assumir.

Figura 5 – Diagrama de estados de um objeto MediaPlayer



Fonte: Google (2012b).

A partir do diagrama da Figura 5, pode-se ver que um objeto da classe `MediaPlayer` possui diversos estados. Estes estados e como são obtidos estão descritos a seguir:

- a) *idle*: quando um objeto da classe está recém instanciado através da palavra `new` ou após a chamada do método `reset()`;
 - b) *error*: podem haver diversas razões para que uma operação de reprodução falhe,

como: formato de arquivo não suportado, resolução alta, *timeout*, erros de programação, etc. Sob essas circunstâncias, o objeto `MediaPlayer` fica no estado de erro;

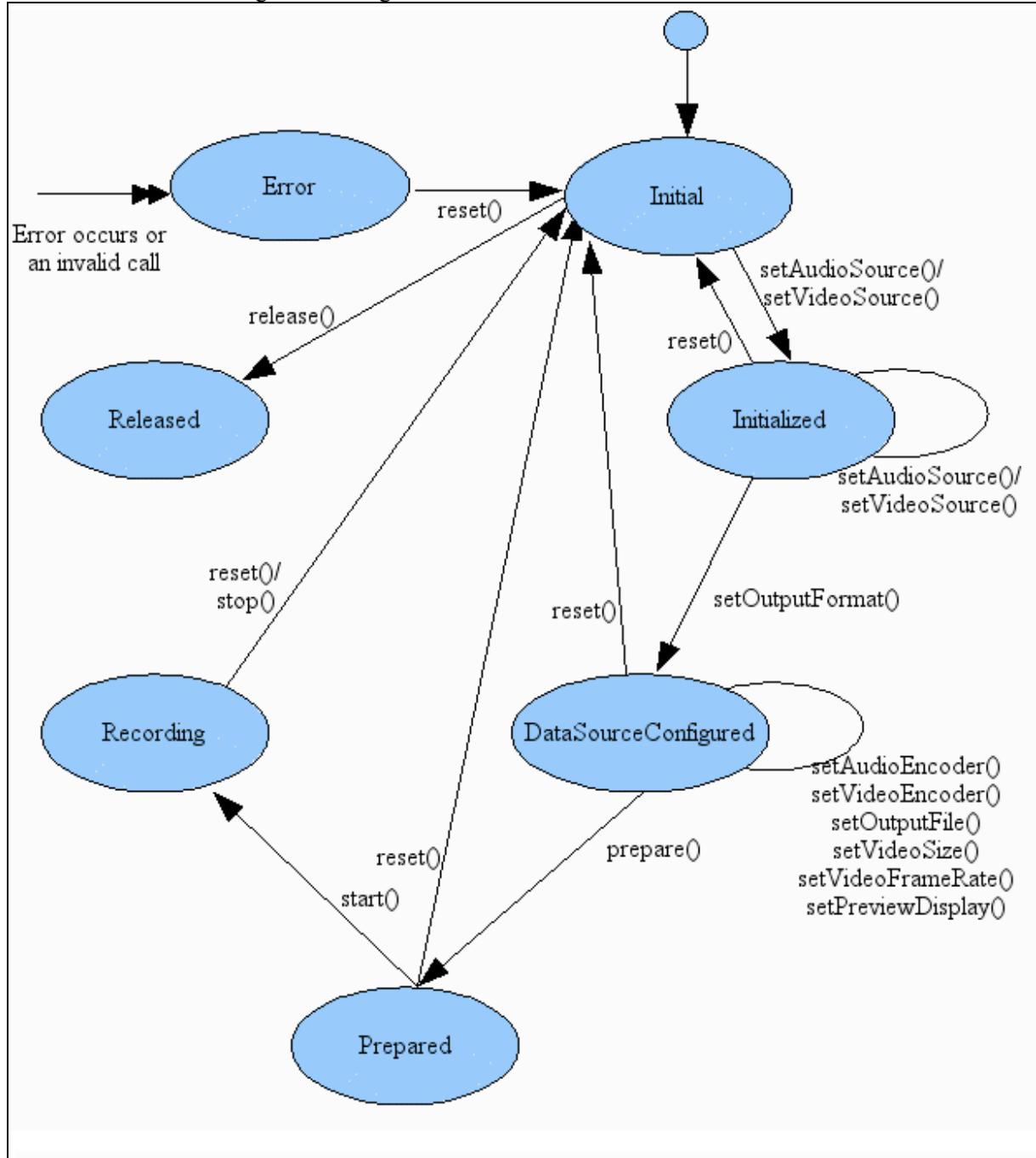
- c) *initialized*: ao utilizar os métodos `setDataSource(FileDescriptor)`, `setDataSource(String)`, `setDataSource(Context, Uri)` ou `setDataSource(FileDescriptor, long, long)`, o objeto é transferido do estado ocioso para o estado inicializado;
- d) *preparing/prepared*: existem duas formas de mudar o estado do objeto para *prepared* uma é síncrona e a outra é assíncrona. A forma síncrona é feita através do método `prepare()` que altera o estado para *prepared* assim que retorna do método. A forma assíncrona é feita através do método `prepareAsync()` que ao entrar no método irá alterar o estado para *preparing* e após a sua finalização altera-o para *prepared*;
- e) *started*: o objeto assume esse estado quando é chamado o método `start()` enquanto o objeto está no status *prepared*;
- f) *paused*: quando é chamado o método `pause()`, após o seu retorno o método entra em estado *paused*;
- g) *stoped*: a reprodução pode ser interrompida nos estados *prepared*, *started*, *paused* e *playbackcompleted*. Essa interrupção é feita através do método `stop()` que deixa o estado do objeto como *stoped*;
- h) *playbackcompleted*: o objeto permite registrar um `OnCompletitionListener` para executar uma operação quando uma reprodução é finalizada. Quando essa operação é executada o objeto é notificado e o estado do objeto é alterado para *playbackcompleted*;
- i) *end*: é o estado final do objeto quando uma reprodução é finalizada, caso a reprodução não esteja em modo de repetição e o objeto não possua um `OnCompletitionListener` associado. O objeto também pode entrar nesse estado quando o método `release()` é chamado.

Além dos estados descritos acima, a classe `MediaPlayer` também possui métodos para controlar diversas propriedades da reprodução, como: volume, repetição, duração e busca.

A plataforma também permite gravar áudio e vídeo usando a classe `MediaRecorder`. É importante ressaltar que ambas as funções, gravar e reproduzir, estão atreladas às limitações

de hardware. A Figura 6 mostra o ciclo de estados que um objeto `MediaRecorder` pode assumir.

Figura 6 – Diagrama de estados da classe `MediaRecorder`



Fonte: Google (2012b).

Os estados apresentados no diagrama e os métodos utilizados para alterar seu estado são descritos a seguir:

- initial*: o objeto entra no estado *initial* assim que é instanciado ou após a chamada dos métodos `reset()` e `stop()`;
- error*: durante a transição de estados e gravação da mídia, caso alguma exceção

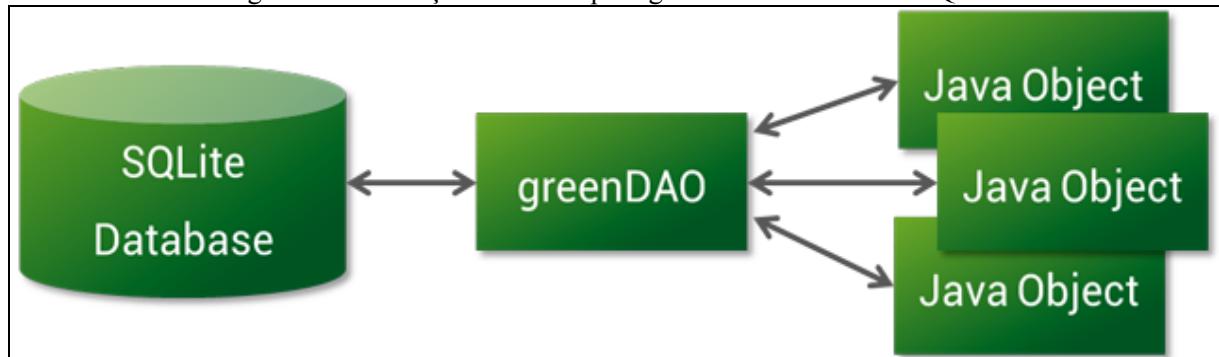
- for lançada o estado do objeto é alterado para *error*;
- c) *initialized*: o estado é alterado para *initialized* quando a fonte de dados para gravação é definida através do método `set AudioSource()` ou do método `set Video Source()`;
 - d) *datasourceconfigured*: a transição para este estado ocorre quando é definido o tipo de formato de saída da mídia através do método `set Output Format()`;
 - e) *prepared*: após a configuração do tipo de mídia e formato, o método `prepare()` altera o estado do objeto para *prepared*;
 - f) *recording*: esse status indica que o objeto está executando a gravação do arquivo, o método para a transição para este estado é o `start()`;
 - g) *released*: é o estado final do objeto, obtido através da chamada para o método `release()`;

2.5 GREENDAO

O greenDAO é uma ferramenta *Object Relational Mapping* (ORM) *opensource* desenvolvida para a plataforma Android que auxilia no desenvolvimento de aplicações com dados gravados no banco de dados padrão do Android, o SQLite (GREENDAO, 2013).

A ferramenta é utilizada para abstrair operações SQL de dados para um paradigma de orientação a objetos, permitindo que as operações básicas de *insert*, *delete*, *update* e *select* sejam realizadas através do mapeamento de objetos Java para tabelas do banco de dados, permitindo o acesso aos dados do banco através desta API (GREENDAO, 2013). A Figura 7, mostra a abstração realizada pelo greenDAO para acesso ao banco de dados através de objetos Java.

Figura 7 – Abstração realizada pelo greenDAO ao acessar o SQLite

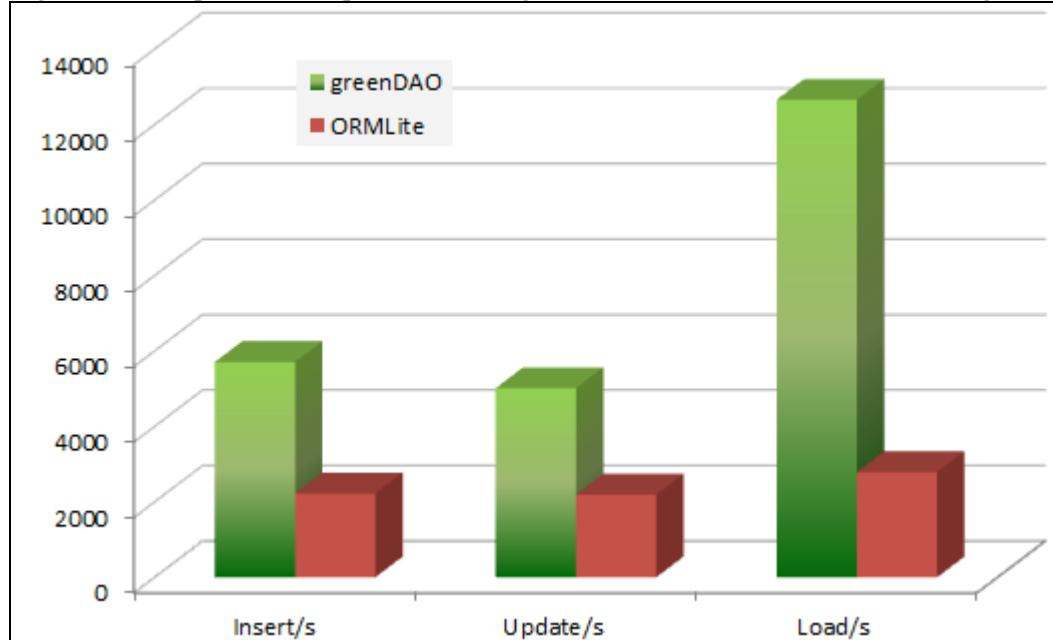


Fonte: greenDAO, 2013.

O ORMLite é uma ferramenta disponível no mercado e que se dispõe a fazer o mesmo trabalho do greenDAO, além de ser uma das mais comuns entre os desenvolvedores. Por isso,

a equipe de desenvolvimento do greenDAO optou por compará-los e verificar o desempenho dos mesmos. Segundo a equipe (GREENDAO, 2013), nos testes realizados, o greenDAO se mostrou cerca de 2 vezes mais rápido para realizar inserções e atualizações e cerca de 4,5 vezes mais rápido para carregar dados do banco de dados. A Figura 8, ilustra os testes realizados das comparações de inserções, atualizações e carregamentos por segundo em cada uma das API's, o código fonte dos testes a suas bibliotecas podem ser encontrados no repositório do greenDAO no *GitHub*.

Figura 8 – Comparaçāo de operações entre greenDao e ORMLite em número de registros



Fonte: greenDao, 2013.

Para utilizar o greenDAO em projetos Android deve-se criar um segundo projeto que tem por objetivo gerar códigos específicos para o seu domínio de projeto. Este projeto deve conter uma classe executável com a modelagem de toda a estrutura do banco de dados. Ao executar este projeto o greenDAO irá criar as classes para comunicação dos objetos modelados com o SQLite.

A partir do momento que o código é gerado, é possível utilizar o greenDAO no projeto. Existem quatro classes essenciais para o funcionamento do greenDAO que são DaoMaster, DaoSession, DAO's e Entities.

A classe DaoMaster contém métodos estáticos para criar as tabelas ou liberá-las. Ele possui duas classes internas, OpenHelper e DevOpenHelper, que são implementações da classe abstrata SQLiteOpenHelper e criam o *schema* do projeto no banco de dados SQLite.

O `DaoSession` fornece métodos de persistência de inserção, exclusão ou carga das entidades e é responsável por manter e controlar todos os objetos `DAO` disponíveis para um esquema específico que pode ser acessado através de métodos *getters*.

Os *Data Access Objects* `DAO`'s são objetos de acesso direto aos dados no banco de dados. Para cada entidade o greenDAO gera um `DAO` específico. Os `DAO`'s tem métodos de persistência mais aprimorados que os métodos do `DaoSession`. As entidades são os objetos persistidos e representam as linhas do banco de dados.

2.6 TRABALHOS CORRELATOS

Hoje existem aplicativos similares ao desenvolvido, disponíveis para os sistemas operacionais móveis iOS e Android. Entre eles estão o TapToTalk (TAPTOTALK, 2014), JABTalk (JABSTONE, 2012) e Proloquo2Go (APPLE, 2012).

2.6.1 TapToTalk

TapToTalk é um aplicativo gratuito que transforma o dispositivo móvel em uma ferramenta de comunicação alternativa acessível (TAPTOTALK, 2014). A imagem apresentada na Figura 9 mostra a tela inicial do aplicativo.

Figura 9 – Tela inicial do TapToTalk



Fonte: TapToTalk (2014).

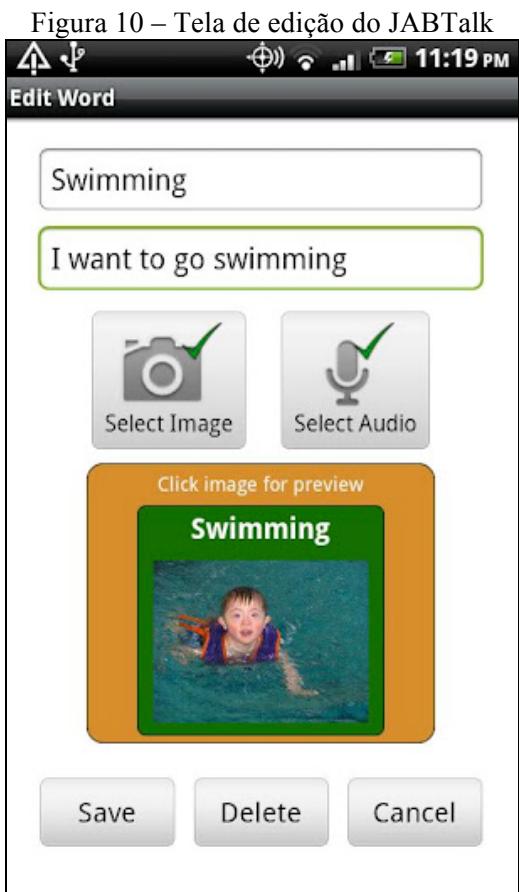
Basicamente, o TapToTalk disponibiliza coleções de imagens relacionadas a algumas das atividades que a pessoa com deficiência pode fazer. Quando a imagem é pressionada o

aplicativo toca o áudio associado a ela e se houverem outras imagens relacionadas a esta ação, outro álbum será apresentado, dando continuidade a comunicação.

Além do álbum de exemplo, também é permitido ao usuário fazer *download* de outros álbuns para o aplicativo.

2.6.2 JABTalk

JABTalk (JABSTONE, 2012) é um aplicativo para Android gratuito e customizável. Apesar de não possuir nenhum álbum de exemplo, o JABTalk permite ao usuário criar álbuns com categorias, subcategorias e palavras. Todos estes álbuns podem ser associados a imagens e sons definidos pelo próprio usuário. A Figura 10 apresenta uma tela de edição da palavra associada a imagem.



Fonte: Jabstone (2012).

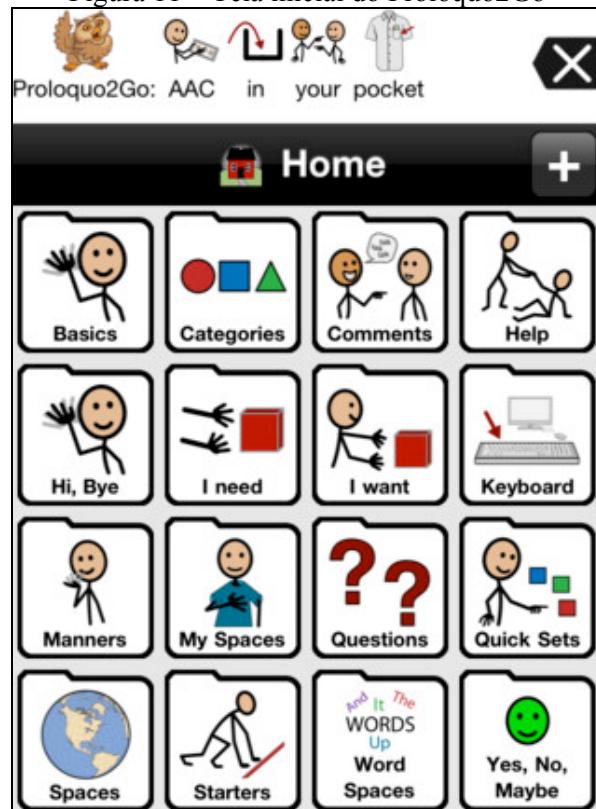
Esta tela é similar também a tela de edição de categorias e subcategorias de imagens. Após o usuário definir a estrutura do aplicativo, é possível navegar entre as categorias e interagir com outras pessoas. De acordo com o site da JABStone (2012), fonoaudiólogos normalmente referem-se ao JABTalk como um aplicativo de CA fácil e eficaz, que através da combinação de imagens, sons e uma interface simples, provê uma solução divertida de usar e fácil de aprender.

2.6.3 Proloquo2Go

Desenvolvido apenas para dispositivos com iOS, o Proloquo2Go é um aplicativo comercial que, segundo a App Store (APPLE, 2012), é uma solução completa de comunicação aumentada e alternativa para pessoas que tem dificuldades de falar. Além das funcionalidades básicas de seleção, fala e customização de álbuns e imagens, o aplicativo também provê uma solução de leitura de texto (disponível apenas em inglês e híndi), tornando opcional a associação de sons a figuras.

O aplicativo possui também um vocabulário inicial com mais de 7.000 variações associadas a uma funcionalidade de previsão de texto. As figuras iniciais do Proloquo2Go são todas em alta resolução. O aplicativo é vendido na App Store pelo valor de cento e oitenta e nove dólares e noventa e nove cents (\$189,99). Na Figura 11 é apresentada a tela inicial do aplicativo.

Figura 11 – Tela inicial do Proloquo2Go



Fonte: Apple (2012).

3 DESENVOLVIMENTO DO APLICATIVO

Neste capítulo são abordadas as etapas de desenvolvimento do aplicativo. A primeira seção apresenta os principais requisitos funcionais e não funcionais do aplicativo desenvolvido. A segunda seção descreve a especificação do aplicativo através de diagramas da *Unified Modeling Language* (UML). A terceira seção apresenta os detalhes da implementação do aplicativo, incluindo os trechos de códigos mais relevantes ao funcionamento do aplicativo e exemplos de uso. Por fim, a quarta seção aborda os resultados deste trabalho.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Em seguida estão listados os requisitos para a criação do aplicativo de CA. Têm-se requisitos para implementação e funcionalidades do aplicativo, que estão categorizados em Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF). Abaixo estão os requisitos desenvolvidos por Fabeni (2012):

- a) permitir a criação de usuários (RF);
- b) permitir ao usuário efetuar o *login* com um usuário já existente (RF);
- c) permitir ao usuário a criação de símbolos com imagem e áudio (RF);
- d) permitir ao usuário a visualização dos símbolos (RF);
- e) permitir ao usuário criar planos (RF);
- f) permitir ao usuário visualizar a lista dos planos criados (RF);
- g) permitir ao usuário visualizar planos (RF);
- h) permitir ao usuário criar históricos de observação (RF);
- i) permitir ao usuário visualizar as observações criadas (RF);
- j) registrar o histórico de utilização dos símbolos em planos (RF);
- k) permitir ao usuário visualizar o histórico de símbolos (RF);
- l) permitir ao usuário interagir com os símbolos através do toque (RF);
- m) tocar o som associado ao símbolo quando o símbolo for selecionado (RF);
- n) exibir uma borda na apresentação dos símbolos de acordo com a cor da categoria selecionada (RF).

A seguir estão os requisitos adicionais propostos para o trabalho que será desenvolvido:

- a) permitir ao usuário escolher o layout do plano (RF);
- b) sincronizar os dados do usuário com o servidor (RF);

- c) permitir iniciar o aplicativo com o dispositivo sem conexão com a internet (RF);
- d) permitir vincular símbolos a vídeos do youtube;
- e) permitir ao usuário visualizar o vídeo associado ao símbolo quando o dispositivo possuir conexão com a internet;
- f) ser implementado utilizando o ambiente de desenvolvimento Eclipse (RNF);
- g) ser implementado utilizando a linguagem de programação Java (RNF);
- h) realizar as trocas de informações entre cliente e servidor enviando e recebendo dados no formato JSON (RNF).

3.2 ESPECIFICAÇÃO

A especificação deste trabalho foi desenvolvida utilizando os diagramas da UML. Os casos de uso são apresentados através de um diagrama de casos de uso, seguidos das descrições dos casos de uso. Em seguida, serão apresentados os diagramas de pacotes exibindo as classes criadas para o aplicativo. Para maior entendimento de alguns fluxos básicos, também serão apresentados diagramas de sequência que representarão casos de uso do aplicativo, após será exibido o diagrama de atividades para apresentar a utilização do aplicativo por completo e por fim um Modelo de Entidade e Relacionamento (MER) da estrutura de dados utilizada pelo Tagarela.

3.2.1 Casos de uso

Nesta seção será apresentado o diagrama de casos de uso do Tagarela e também a descrição dos cenários de cada requisito especificado.

Um único ator irá interagir com todos os casos de uso, esse ator corresponde ao usuário Tutor no projeto Tagarela. Para facilitar a apresentação do diagrama de casos de uso, este foi dividido em três diagramas. A Figura 12 apresenta os casos de uso do usuário quando efetua o *login* no sistema com um usuário já existente, já a Figura 13 apresenta os casos de uso referentes à criação de usuário e de seus dados como símbolos, planos e observações. Por fim, a Figura 14 apresenta os casos de uso da visualização e interação com os itens.

Figura 12 – Casos de uso do Tagarela referentes ao *login* no aplicativo

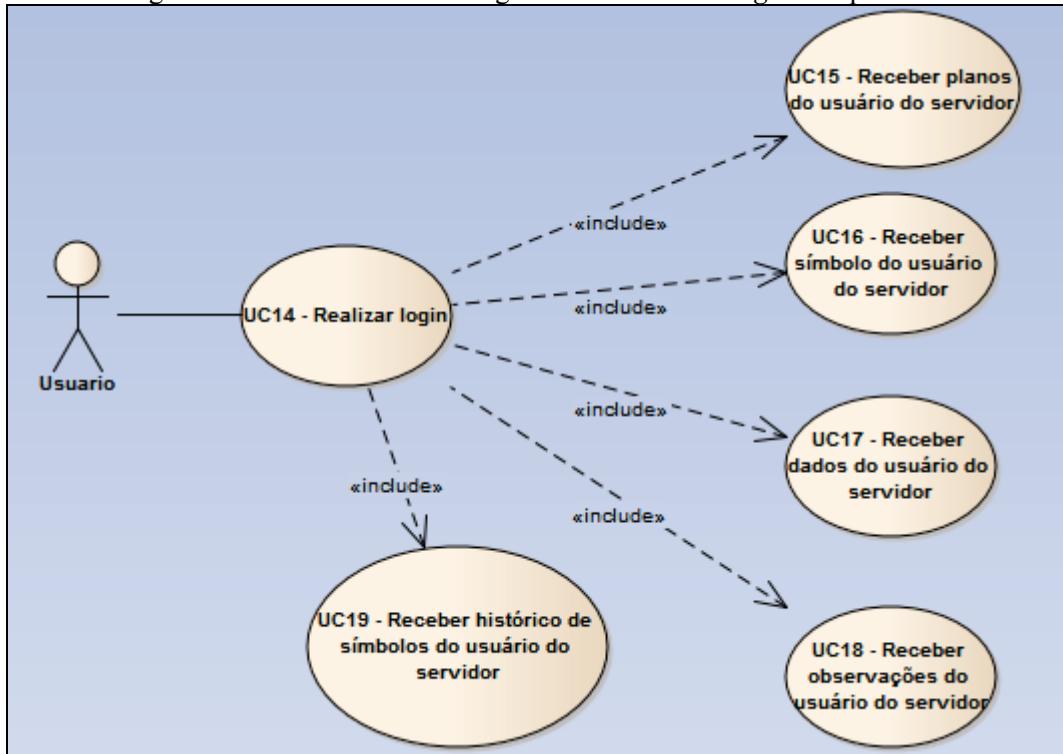


Figura 13 – Casos de uso do Tagarela referentes a criação de dados do usuário

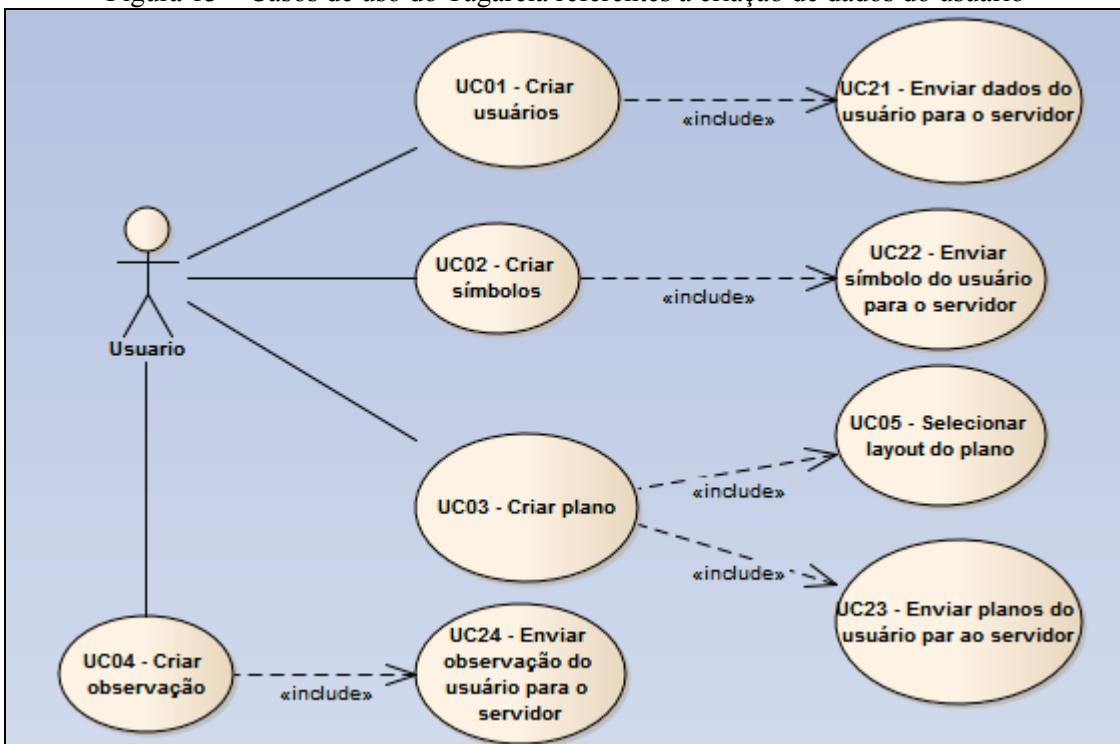
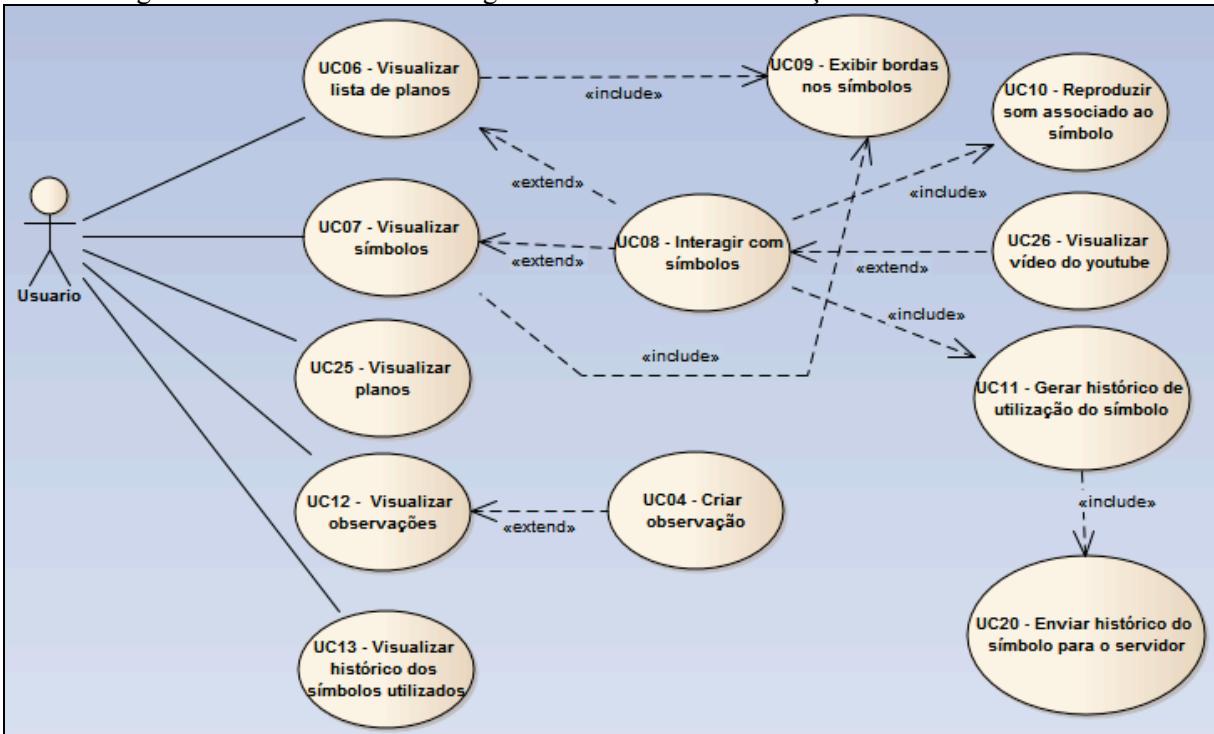


Figura 14 – Casos de uso do Tagarela referentes a visualização dos dados do usuário

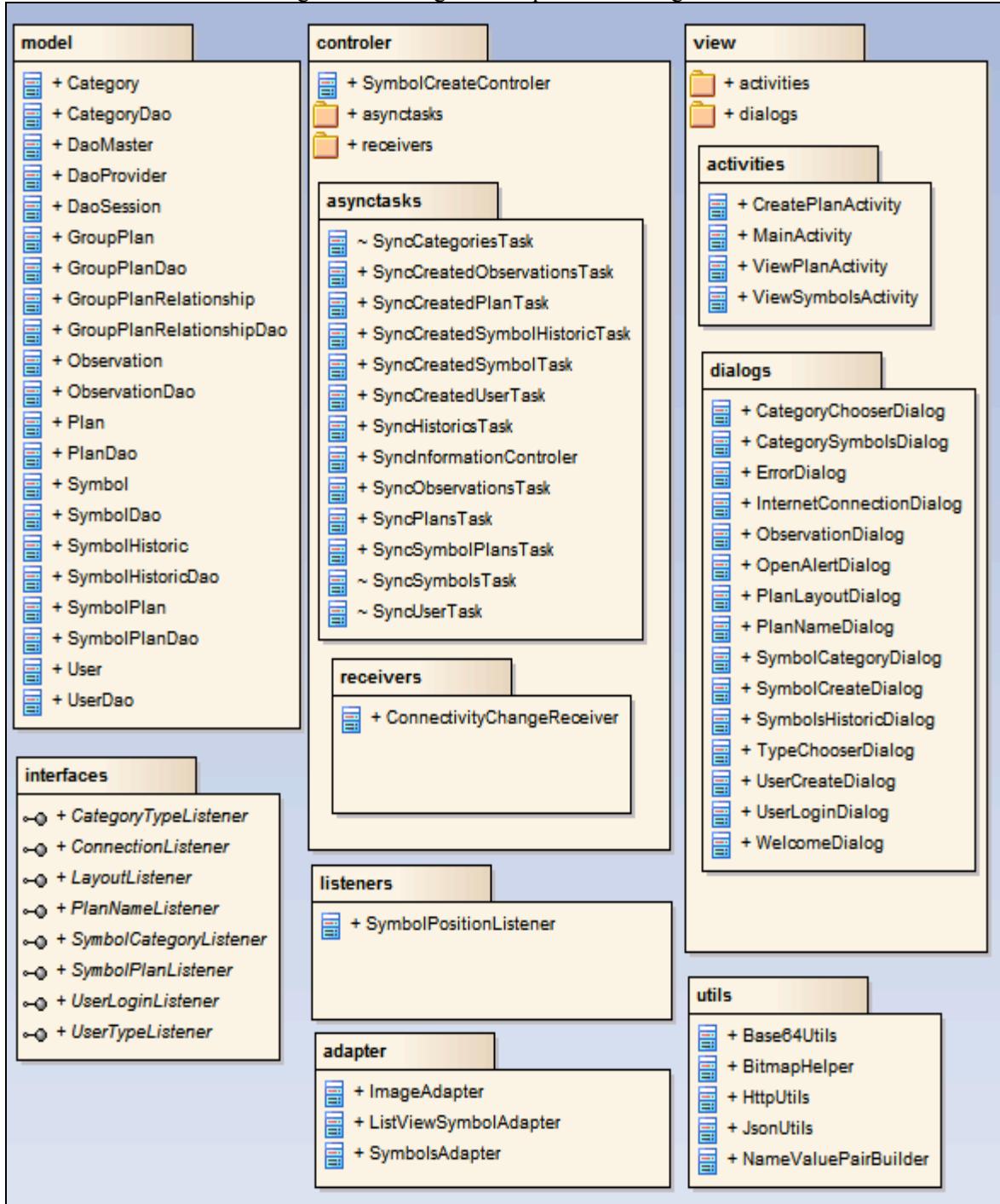


Os diagramas acima foram desenvolvidos com base no padrão UML de especificação utilizando a ferramenta Enterprise Architect. Foi criado ao menos um cenário para cada caso de uso acima e este foi vinculado a no mínimo um requisito funcional, defendendo a sua existência. As descrições dos *Use Cases* (UC), cenários, fluxos e exceções destes casos de uso podem ser vistas no apêndice A.

3.2.2 Estrutura do projeto

Nesta seção são apresentadas as classes e interfaces que compõem o aplicativo, bem como seu relacionamento e estrutura. O projeto está dividido em pacotes que agrupam classes com funcionalidade semelhante. Os pacotes e subpacotes utilizados no Tagarela são: model, controller, asynctasks, receivers, view, activities, dialogs, interfaces, listeners, adapter e util. Para auxiliar o entendimento de como as classes estão organizadas a Figura 15 apresenta os pacotes do projeto e as classes que o compõem.

Figura 15 – Diagrama de pacotes do Tagarela



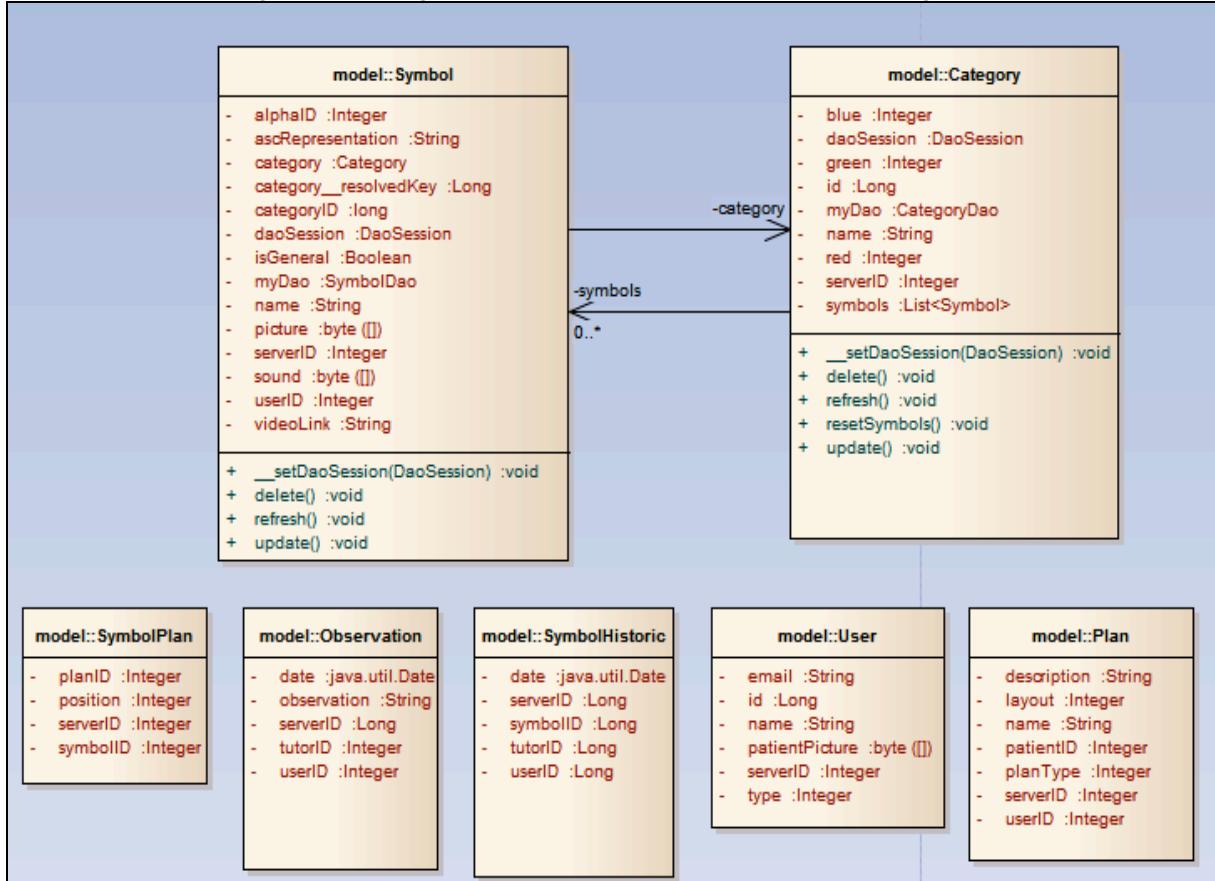
3.2.2.1 Pacote `br.com.furb.tagarela.model`

As classes do pacote `model` são responsáveis por acessar e representar os dados e modelos da aplicação. Neste pacote estão as classes criadas pela API greenDAO compostas por classes modelos, classes DAO, classe DaoProvider, DaoSession e DaoMaster.

Na Figura 16 são apresentadas as classes modelo usadas no Tagarela. Na figura foram removidos os métodos *getters*, *setters* e construtores. Essas classes representam as tabelas do

banco de dados através de objetos no aplicativo, são elas: Category, Symbol, SymbolPlan, Plan, User, SymbolHistoric e Observation.

Figura 16 – Diagrama de classes das classes modelo do Tagarela



A classe `Plan` representa um plano do paciente, esse plano é formado por símbolos que são definidos através da classe `Symbol`, então, como um plano pode conter muitos símbolos e um símbolo pode estar contido em vários planos, foi criada a classe `SymbolPlan` para normalizar dados e evitar o relacionamento *n:n*. As classes `Observation` e `SymbolHistoric` representam os objetos de registro de informações da aplicação, por fim, a classe `User` define o modelo de usuário do aplicativo Tagarela.

Nas classes `Symbol` e `Category` foi criado um relacionamento para que fosse possível listar todos os símbolos de uma categoria sem a necessidade de uma nova consulta no banco de dados.

Com exceção da classe `DaoProvider`, todas as classes são geradas pelo `greenDao` e irão abstrair a utilização de SQL no código fonte. A classe `DaoProvider` foi criada utilizando o padrão de projeto *Singleton* com o intuito de iniciar e prover acesso a todas as classes DAO (*Data Access Object*) geradas pelo `greenDao`. O acesso ao objeto da classe é fornecido através do método `getInstance(Context contexto)`. Quando é feita a chamada do método

`getInstance` pela primeira vez, é instanciado o objeto privado `daoProvider`, que em seu construtor inicializa o banco de dados no dispositivo e atribui valor aos objetos DAO, após isso, retorna a instância do objeto. A Figura 17 apresentam as classes DAO do Tagarela.

Figura 17 – Classes DAO do Tagarela

<pre> SymbolDao AbstractDao + category_SymbolsQuery :Query<Symbol> - daoSession :DaoSession - selectDeep :String + TABLENAME :String = "SYMBOL" {readOnly} + _queryCategory_Symbols(long) :List<Symbol> # attachEntity(Symbol) :void # bindValues(SQLiteStatement, Symbol) :void + createTable(SQLiteDatabase, boolean) :void + dropTable(SQLiteDatabase, boolean) :void + getKey(Symbol) :Void # getSelectDeep() :String # isEntityUpdateable() :boolean + loadAllDeepFromCursor(Cursor) :List<Symbol> # loadCurrentDeep(Cursor, boolean) :Symbol + loadDeep(Long) :Symbol # loadDeepAllAndCloseCursor(Cursor) :List<Symbol> + queryDeep(String, String) :List<Symbol> + readEntity(Cursor, int) :Symbol + readEntity(Cursor, Symbol, int) :void + readKey(Cursor, int) :Void + SymbolDao(DaoConfig) + SymbolDao(DaoConfig, DaoSession) # updateKeyAfterInsert(Symbol, long) :Void </pre>	<pre> PlanDao AbstractDao + TABLENAME :String = "PLAN" {readOnly} # bindValues(SQLiteStatement, Plan) :void + createTable(SQLiteDatabase, boolean) :void + dropTable(SQLiteDatabase, boolean) :void + getKey(Plan) :Void # isEntityUpdateable() :boolean + PlanDao(DaoConfig) + PlanDao(DaoConfig, DaoSession) + readEntity(Cursor, int) :Plan + readEntity(Cursor, Plan, int) :void + readKey(Cursor, int) :Void # updateKeyAfterInsert(Plan, long) :Void </pre>	<pre> UserDao AbstractDao + TABLENAME :String = "USER" {readOnly} # bindValues(SQLiteStatement, User) :void + createTable(SQLiteDatabase, boolean) :void + dropTable(SQLiteDatabase, boolean) :void + getKey(User) :Long # isEntityUpdateable() :boolean + readEntity(Cursor, int) :User + readEntity(Cursor, User, int) :void + readKey(Cursor, int) :Long # updateKeyAfterInsert(User, long) :Long + UserDao(DaoConfig) + UserDao(DaoConfig, DaoSession) </pre>
<pre> CategoryDao AbstractDao - daoSession :DaoSession + TABLENAME :String = "CATEGORY" {readOnly} # attachEntity(Category) :void # bindValues(SQLiteStatement, Category) :void + CategoryDao(DaoConfig) + CategoryDao(DaoConfig, DaoSession) + createTable(SQLiteDatabase, boolean) :void + dropTable(SQLiteDatabase, boolean) :void + getKey(Category) :Long # isEntityUpdateable() :boolean + readEntity(Cursor, int) :Category + readEntity(Cursor, Category, int) :void + readKey(Cursor, int) :Long # updateKeyAfterInsert(Category, long) :Long </pre>		
<pre> SymbolHistoricDao AbstractDao + TABLENAME :String = "SYMBOL_HISTORIC" {readOnly} # bindValues(SQLiteStatement, SymbolHistoric) :void + createTable(SQLiteDatabase, boolean) :void + dropTable(SQLiteDatabase, boolean) :void + getKey(SymbolHistoric) :Long # isEntityUpdateable() :boolean + readEntity(Cursor, int) :SymbolHistoric + readEntity(Cursor, SymbolHistoric, int) :void + readKey(Cursor, int) :Long + SymbolHistoricDao(DaoConfig) + SymbolHistoricDao(DaoConfig, DaoSession) # updateKeyAfterInsert(SymbolHistoric, long) :Long </pre>	<pre> ObservationDao AbstractDao + TABLENAME :String = "OBSERVATION" {readOnly} # bindValues(SQLiteStatement, Observation) :void + createTable(SQLiteDatabase, boolean) :void + dropTable(SQLiteDatabase, boolean) :void + getKey(Observation) :Long # isEntityUpdateable() :boolean + ObservationDao(DaoConfig) + ObservationDao(DaoConfig, DaoSession) + readEntity(Cursor, int) :Observation + readEntity(Cursor, Observation, int) :void + readKey(Cursor, int) :Long # updateKeyAfterInsert(Observation, long) :Long </pre>	<pre> SymbolPlanDao AbstractDao + TABLENAME :String = "SYMBOL_PLAN" {readOnly} # bindValues(SQLiteStatement, SymbolPlan) :void + createTable(SQLiteDatabase, boolean) :void + dropTable(SQLiteDatabase, boolean) :void + getKey(SymbolPlan) :Void # isEntityUpdateable() :boolean + readEntity(Cursor, int) :SymbolPlan + readEntity(Cursor, SymbolPlan, int) :void + readKey(Cursor, int) :Void + SymbolPlanDao(DaoConfig) + SymbolPlanDao(DaoConfig, DaoSession) # updateKeyAfterInsert(SymbolPlan, long) :Void </pre>

As classes DAO do Tagarela são: `CategoryDao`, `PlanDao`, `SymbolDao`, `UserDao`, `ObservationDao`, `SymbolPlanDao` e `SymbolHistoricDao`. Estas classes são geradas pelo greenDAO e são responsáveis pelo acesso e manipulação das entidades modelo. A manipulação é feita através de métodos que executam comandos de manipulação como *insert*, *delete* e *update*. Além dos comandos de manipulação citados anteriormente, as classes DAO também fornecem comandos de consulta disponibilizados por meio do acesso a `Query`. A

classe `Query` pode ser gerada por meio de uma classe *Builder* chamada `QueryBuilder` que permite que sejam adicionadas restrições para a consulta que for executada.

3.2.2.2 Pacote `br.com.furb.tagarela.view`

O pacote `view` é dividido em dois sub-pacotes, `dialogs` e `activities`. As classes que pertencem ao pacote `dialogs` tem como função permitir que o usuário faça decisões ou insira informações adicionais para algum propósito em uma janela na tela. O pacote `activities` contém as classes que representam componentes da aplicação do Tagarela. Ao contrário das classes do pacote `dialogs`, as `activities` se ajustam à tela do dispositivo. O pacote `dialogs` apresentado nas Figuras 18 e 19, contém todas as classes que herdam da classe `DialogFragment`.

Figura 18 – Diagrama de classes das classes `view.dialogs` do Tagarela

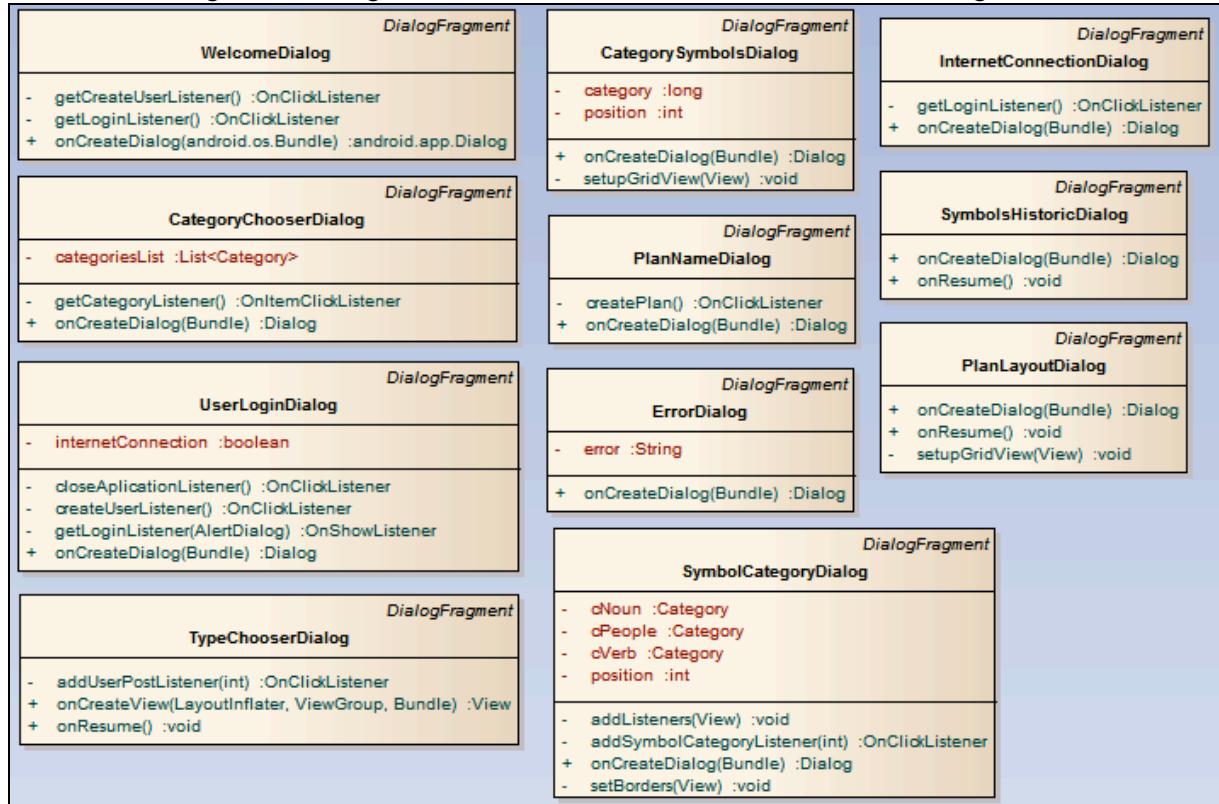
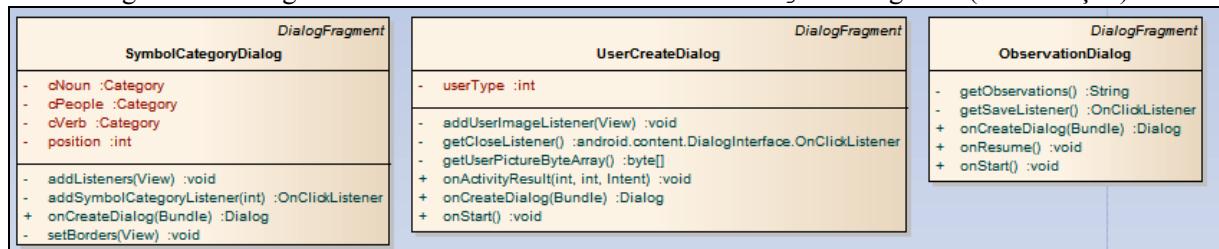


Figura 19 – Diagrama de classes das classes `view.dialogs` do Tagarela (continuação)



Estas classes representam as janelas de diálogo da aplicação. A classe `DialogFragment` é um *container* de diálogo que garante que todo o ciclo de vida da janela

seja realizado corretamente. As descendentes de `DialogFragment` sobrescrevem o método `onCreateDialog(Bundle savedInstanceState)` para retornar um diálogo customizado criado utilizando a *inner class* `Builder` da classe `AlertDialog`.

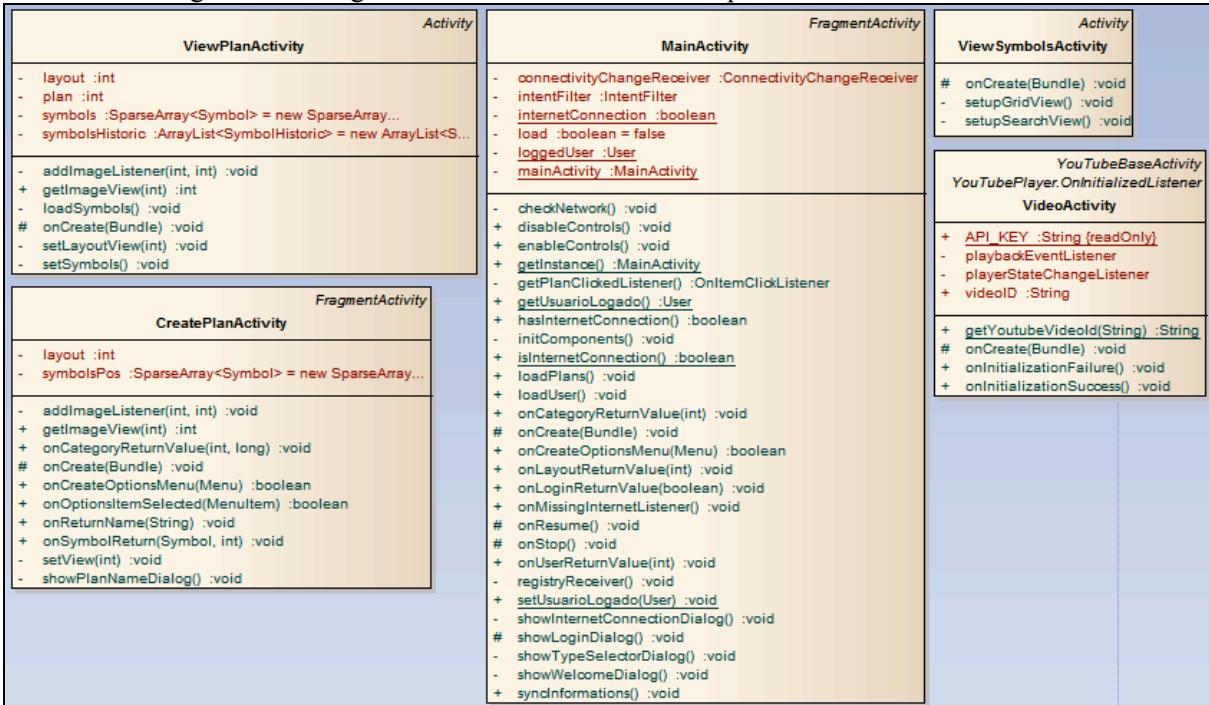
A seguir serão descritas as classes do pacote `view.dialogs` e a sua função:

- a) `WelcomeDialog`: classe que representa a janela de boas-vindas ao entrar no aplicativo. Tem como função definir a forma como o usuário irá entrar no aplicativo sendo possível entrar com um usuário já existente na base ou utilizar um novo usuário;
- b) `UserLoginDialog`: classe que representa uma janela de *login* para usuários já cadastrados. Tem a função de verificar se o usuário inserido é um usuário válido e também oferece ao usuário um botão para que vá para a janela de cadastro de usuário;
- c) `TypeChooserDialog`: classe que representa a janela de seleção de perfil durante o processo criação de usuário;
- d) `UserCreateDialog`: classe que representa a janela onde são inseridas as informações do usuário que será criado. Esta classe é responsável por finalizar a criação do usuário e realizar consistência dos campos informados;
- e) `CategoryChooserDialog`: classe que representa a janela de seleção de categoria de um símbolo. Ao abrir a janela são apresentadas as categorias disponíveis do aplicativo em uma lista;
- f) `SymbolCreateDialog`: classe que representa a janela de criação de símbolos. É encarregada de exibir os componentes para o usuário e de repassar os dados do usuário para a classe `SymbolCreateController`;
- g) `PlanLayoutDialog`: classe que representa a janela de seleção de *layouts* para o usuário. Ao selecionar um *layout* esta classe é responsável por notificar a atividade `MainActivity` para que o usuário seja encaminhado para a tela de criação de plano;
- h) `SymbolCategoryDialog`: esta classe é responsável por exibir as categorias do aplicativo para o usuário. Ao selecionar uma categoria, a classe envia a categoria selecionada para a classe `CategorySymbolsDialog`;
- i) `CategorySymbolsDialog`: classe que representa a janela que exibe os símbolos de uma determinada categoria. Ao selecionar o símbolo, esta classe envia o símbolo para a atividade na posição selecionada no layout;

- j) PlanNameDialog: classe que representa a janela onde será informado um nome para o plano que será criado. O usuário ao confirmar a criação do plano na janela a classe retorna o nome para a atividade onde o plano será efetivamente criado;
- k) InternetConnectionDialog: classe que representa uma janela que pode ser exibida ao iniciar o aplicativo informando o usuário que não há conexão com a internet. Ao fechar a janela, a classe envia a atividade uma notificação para que seja aberta a janela de *login*, sem a opção de criar usuário;
- l) ObservationDialog: esta classe representa uma janela que permite salvar observações para o usuário e também visualizar as informações já existentes. Ao fechar a janela, caso o usuário tenha digitado alguma informação no campo de observação, essa observação é salva no aplicativo e enviada para o servidor;
- m) SymbolHistoricDialog: esta classe representa a janela em que o histórico de símbolos é exibido. Ela é responsável por exibir uma lista com as imagens dos símbolos, os nomes e a data que foram utilizados;
- n) ErrorDialog: esta classe representa uma janela padrão para exibição de erros no aplicativo.

O pacote `activities` contém as atividades que representam as telas do aplicativo. Essas classes não foram projetadas com o intuito de serem utilizadas por outras aplicações, porém, a herança de `Activity` permitiria que fossem. Além da herança de `Activity`, as `activities` do Tagarela implementam diversas interfaces do pacote `interfaces` que as obriga a implementar métodos que recebem o retorno das janelas de diálogo, funcionando como o padrão de projeto *Observer*.

Figura 20 – Diagrama de classes das classes do pacote view.activities



A seguir serão descritas as classes correspondentes às atividades do Tagarela e sua função conforme a Figura 20:

- MainActivity:** esta classe representa a tela principal do aplicativo, a partir dela são abertas todas as outras janelas e telas do Tagarela. Na maior parte do aplicativo a interação entre as telas é realizada por meio dessa classe, que implementa diversas interfaces que possibilitam à ela receber os retornos de valores obtidos das classes *Dialog*. A *MainActivity* também realiza o controle do aplicativo de acordo com o estado da conectividade do aparelho com a internet, esse controle é possibilitado pela utilização da classe *ConnectivityChangeReceiver*;
- ViewSymbolsActivity:** esta classe representa a tela onde são exibidos os símbolos criados para o usuário atual. A classe inicializa o componente *GridView* e registra o *adapter* customizado *SymbolAdapter* que irá definir como os símbolos serão exibidos no componente;
- CreatePlanActivity:** esta classe representa a tela onde é exibido o *layout* para criação do plano. Este *layout* é montado dinamicamente pela classe de acordo com o selecionado pelo usuário. A classe também é responsável pela chamada do *Dialog PlanNameDialog* e pelo envio dos planos para sincronização;
- ViewPlanActivity:** esta classe representa a tela responsável pela visualização e interação com os planos. A classe é responsável por montar o *layout* da tela de acordo com o do plano selecionado e adicionar a ação de reprodução de som ao

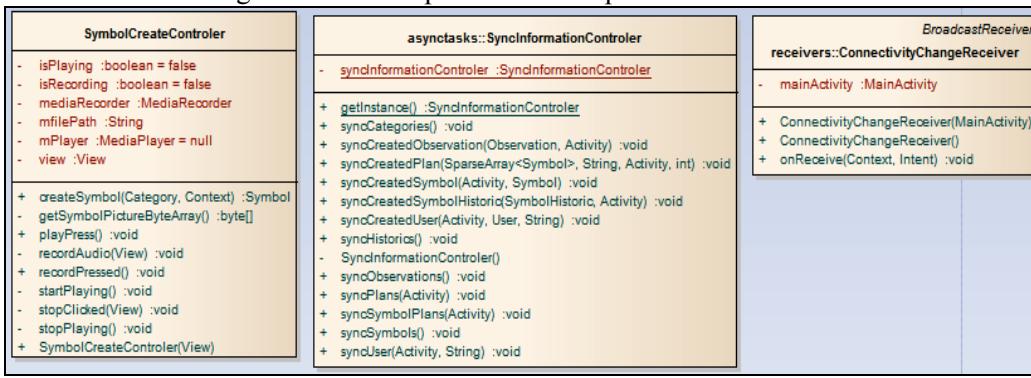
tocar em um símbolo;

- e) `VideoActivity`: esta classe representa a tela onde o vídeo do youtube associado ao símbolo será reproduzido. A classe é responsável por exibir o *player* do vídeo e reproduzi-lo.

3.2.2.3 Pacote `br.com.furb.tagarela.controller`

O pacote `controller` possui classes que controlam a aplicação e os dados. A Figura 21 apresenta as principais classes do pacote. Estas classes estão descritas a seguir:

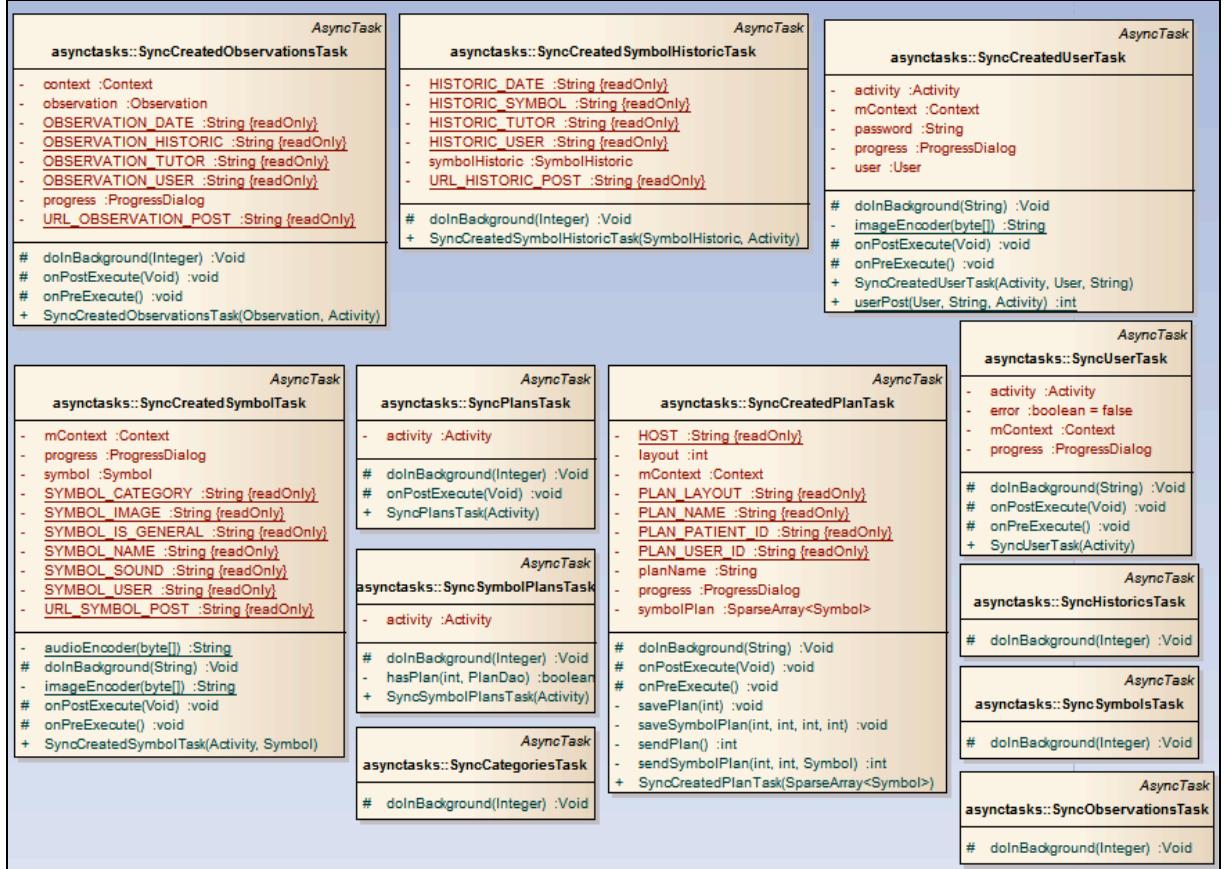
Figura 21 – Principais classes do pacote `controller`



- a) `SymbolCreateController`: esta classe é responsável pela criação de símbolos, também controla a reprodução e gravação do áudio associado ao símbolo na classe `SymbolCreateDialog`;
- b) `SyncInformationController`: esta classe é um *mediator* responsável pela execução das sincronizações realizadas nas classes descendentes de `AsyncTask`;
- c) `ConnectivityChangeReceiver`: esta classe é uma herança da classe `BroadcastReceiver` que tem como objetivo receber notificações do sistema operacional informando alterações no status do dispositivo. Por meio da sobrescrita do método `onReceive()`, a classe `ConnectivityChangeReceiver` consegue controlar o estado das funções do aplicativo quando a conexão com a internet é alterada.

No pacote `controller` também temos classes que herdam de `AsyncTask` e são utilizadas para enviar e receber dados do servidor.

Figura 22 – AsyncTasks do pacote controller



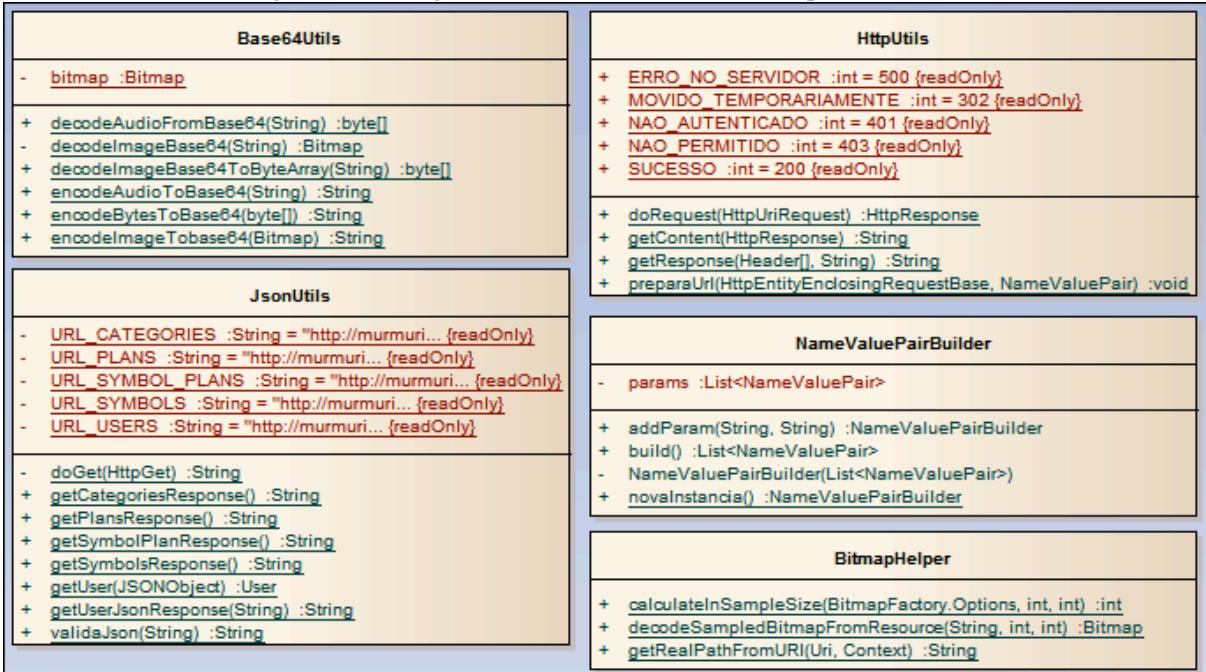
A Figura 22 apresenta as classes que utilizam essa herança, são elas:

SyncCategoriesTask,	SyncCreatedPlanTask,	SyncCreatedSymbolTask,
SyncCreatedUserTask,	SyncInformationController,	SyncPlansTask,
SyncSymbolPlansTask,	SyncSymbolsTask,	SyncUserTask,
SyncCreatedObservationsTask,	SyncUserTask,	SyncObservationsTask,
SyncCreatedSymbolHistoricTask.		SyncHistoricsTask

3.2.2.4 Pacote br.com.furb.tagarela.utils

O pacote utils é um pacote com classes utilitárias que o Tagarela utiliza para operações comuns de atividades e diálogos. As classes desse pacote são em sua maioria formadas por métodos estáticos. A Figura 23 apresenta o diagrama de classes do pacote utils.

Figura 23 – Diagrama de classes das classes do pacote utils



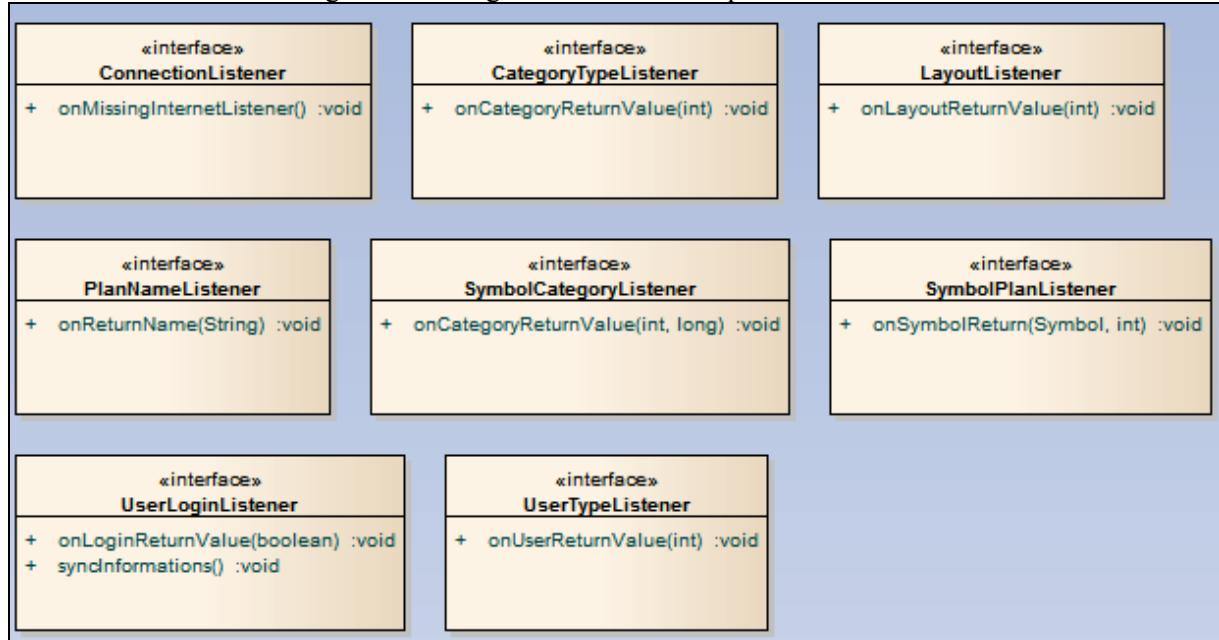
A seguir serão descritas as classes utilitárias, sua utilização e seus principais métodos:

- a) **Base64Utils**: esta é uma classe utilitária com métodos estáticos para codificação e decodificação de arquivos de mídia como imagens e áudio para Base64;
- b) **HttpUtils**: classe utilitária com métodos estáticos para envio de requisições HTTP para o servidor;
- c) **JsonUtils**: classe utilitária que realiza envios e requisições para o servidor através de objetos JSON;
- d) **BitMapHelper**: classe utilitária criada com o propósito de resolver problemas de `OutOfMemoryException` que ocorriam durante o carregamento de `BitMaps` no aplicativo. Este tipo de erro é muito comum em dispositivos móveis, pois, grande parte dos aplicativos atuais tentam carregar arquivos em resoluções maiores do que as resoluções que o dispositivo permite exibir. Foram criados métodos para retornar um `BitMap` na resolução adequada para utilização;
- e) **NameValuePaisBuilder**: é um *builder* da classe `NameValuePair`. Adiciona parâmetros em uma lista interna e retorna-a no método `build()` para que os parâmetros sejam adicionados às requisições HTTP.

3.2.2.5 Pacote br.com.furb.tagarela.interface

O pacote interfaces do Tagarela contém interfaces que possuem métodos para retornar valores das janelas de diálogo do Tagarela. As classes que implementam estas interfaces são as activities CreatePlanActivity e MainActivity.

Figura 23 – Diagrama de classes do pacote interfaces



A Figura 23 apresenta as interfaces e seus respectivos métodos. As interfaces não têm implementação de métodos e por esse motivo não são detalhadas aqui.

3.2.2.6 Pacote br.com.furb.tagarela.adapter

Este pacote contém as classes que herdam dos *adapters* padrões do Android BaseAdapter e ArrayAdapter.

Figura 24 – Classes do pacote adapter



A Figura 24 apresenta as classes do pacote. Essas classes foram criadas com o intuito de exibir itens personalizados nos componentes GridView e ListView do Android. A classe SymbolsAdapter é responsável por preparar o GridView da atividade ViewSymbolsActivity para exibir os símbolos do usuário. Já a classe ImageAdapter é utilizada para preparar o

`GridView` da classe `PlanLayoutDialog` para exibir as imagens que representarão os *layouts* que um plano pode ter. Por fim, a classe `ListViewSymbolAdapter` prepara o componente `ListView` da classe `SymbolsHistoricDialog` para exibir uma view que representa o histórico de um símbolo a cada linha do componente.

3.2.2.7 Pacote `br.com.furb.tagarela.listeners`

O pacote foi criado com o intuito conter as classes que implementam interfaces para utilização de *listeners*. A versão atual do Tagarela em Android, possui apenas a classe `SymbolPositionListener` que implementa a interface `OnClickListener`. Essa classe foi criada para que no momento que o usuário selecione um `ImageView` na criação do plano o *listener* abra o diálogo `SymbolCategoryDialog` passando a posição selecionada como parâmetro.

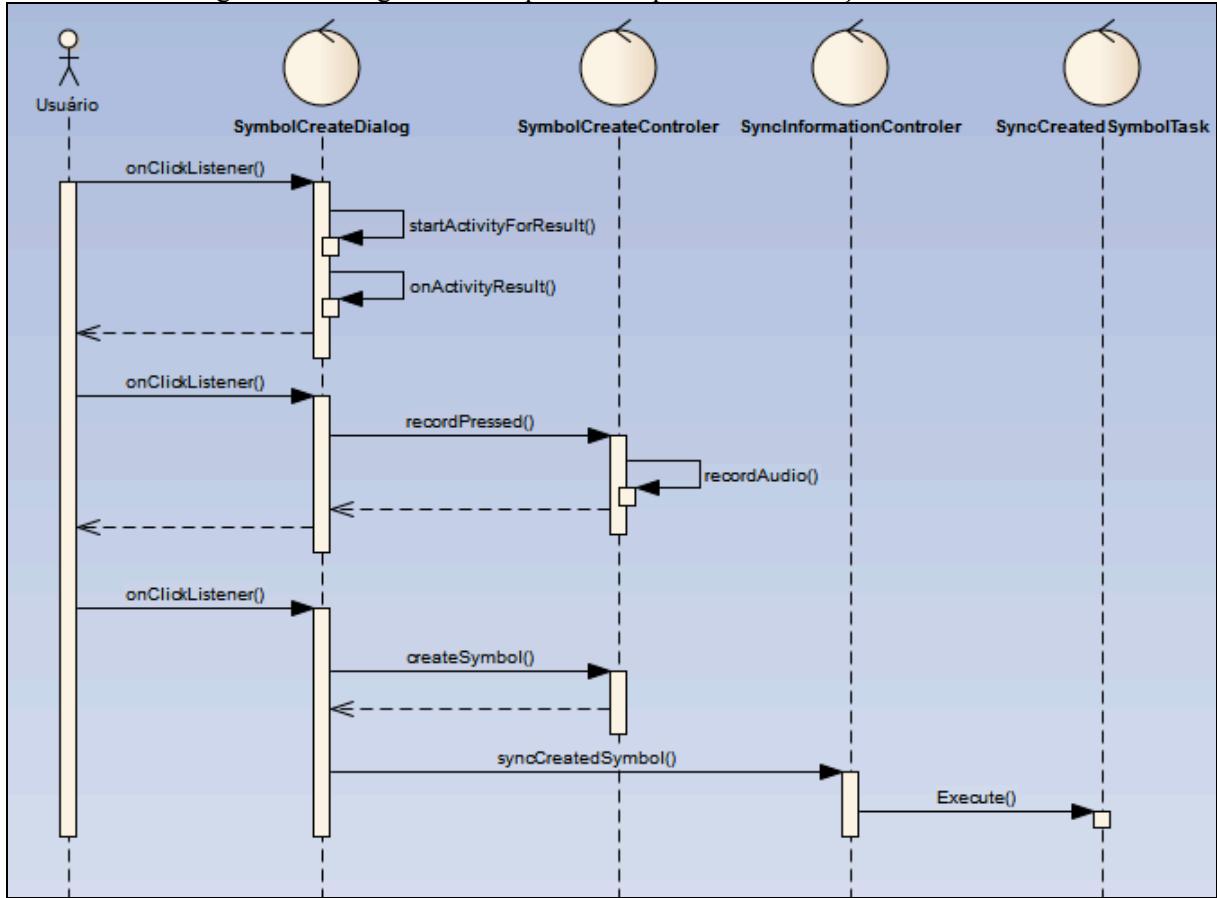
3.2.3 Diagramas de sequência

Os diagramas de sequência têm por objetivo apresentar a troca de mensagens entre as classes do Tagarela, simplificando a interpretação dos casos de uso. Nesta seção são apresentados os diagramas de sequência dos casos de uso `Criar símbolos` (UC02) e `Criar planos` (UC03).

3.2.3.1 Diagrama de sequência `Criar símbolos`

Para criar um símbolo é necessário selecionar a opção criar símbolo na atividade principal, selecionar a categoria dele e então preencher os dados do símbolo, como: nome do símbolo, a categoria (que já estará preenchida), a imagem que o representará e o áudio que será associado a imagem. A Figura 25 apresenta o diagrama de sequência do processo de criação de símbolos.

Figura 25 – Diagrama de sequência do processo de criação de símbolos



Na janela de criação de símbolo existe uma imagem padrão que ao ser tocada dispara um `OnClickListener`. O `OnClickListener` associado a imagem irá criar *intents* de imagem, que pode ser um explorador de arquivos ou a câmera do dispositivo e então irá chamar o método `startActivityForResult(Intent intent, int requestCode)` que irá abrir um diálogo para o usuário escolher qual recurso irá utilizar para retornar a imagem e então abrir a atividade correspondente. Quando o usuário selecionar uma imagem, será chamado o método `onActivityResult(int requestCode, int resultCode, Intent data)` que por sua vez irá obter o resultado da *intent* através do parâmetro `data` e então irá exibir a imagem no `ImageView`.

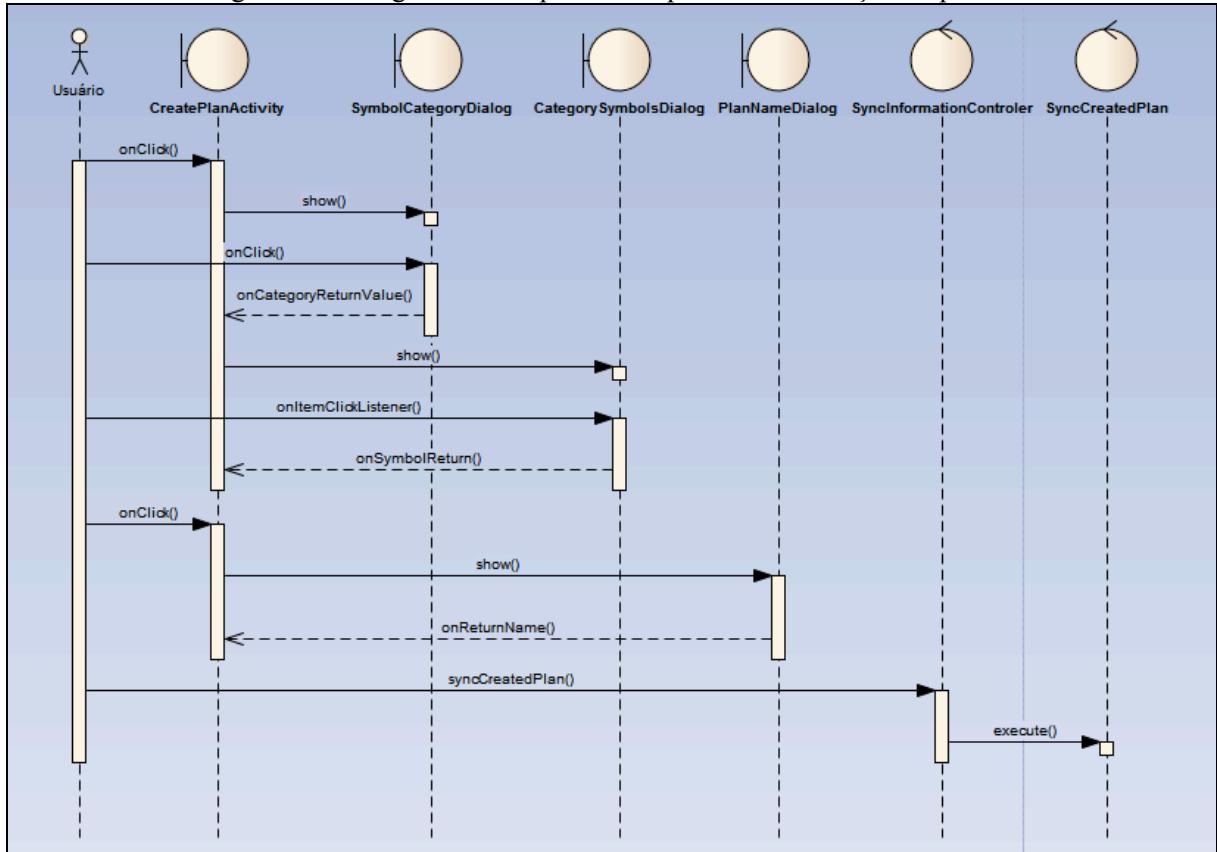
Após associar a imagem, o usuário também poderá vincular uma mensagem de áudio que será a descrição sonora do símbolo. Para isso, no momento que o usuário tocar o link gravar som será chamado o método `recordPressed()` da classe `SymbolCreateController`. Esse método é chamado tanto ao pressionar gravar som quanto parar, quando o método é executado, é verificado o estado da variável `isRecording` para identificar qual ação executar. No caso da gravação, é feita a chamada para o método `recordAudio()` que irá instanciar a classe `MediaRecorder` e gravar o som em um arquivo temporário.

Após preencher os campos, selecionar uma imagem e um som, só resta ao usuário confirmar a criação do símbolo através do botão `Salvar`. Assim que o botão é pressionado, o método `createSymbol(Category category, Context context)` irá retornar uma nova instância da classe `Symbol` com as informações do símbolo criado. O objeto `symbol` será enviado para o método `syncCreatedSymbol(Activity activity, Symbol symbol)` da classe `SyncInformationController`, que irá iniciar a `AsyncTask SyncCreatedSymbolTask` que sincroniza o símbolo criado com o servidor e após isso irá gravá-lo no banco de dados local.

3.2.3.2 Diagrama de sequência Criar planos

O caso de uso de criação de planos tem como pré-requisito a seleção de um *layout* para o plano e a criação de pelo menos um símbolo para o usuário. Ao entrar na atividade de criação de planos é apresentado o *layout* previamente selecionado com o número correspondente de `ImageViews` que representarão os símbolos. A Figura 26 apresenta o diagrama de sequência do processo de criação de planos.

Figura 26 – Diagrama de sequência do processo de criação de planos



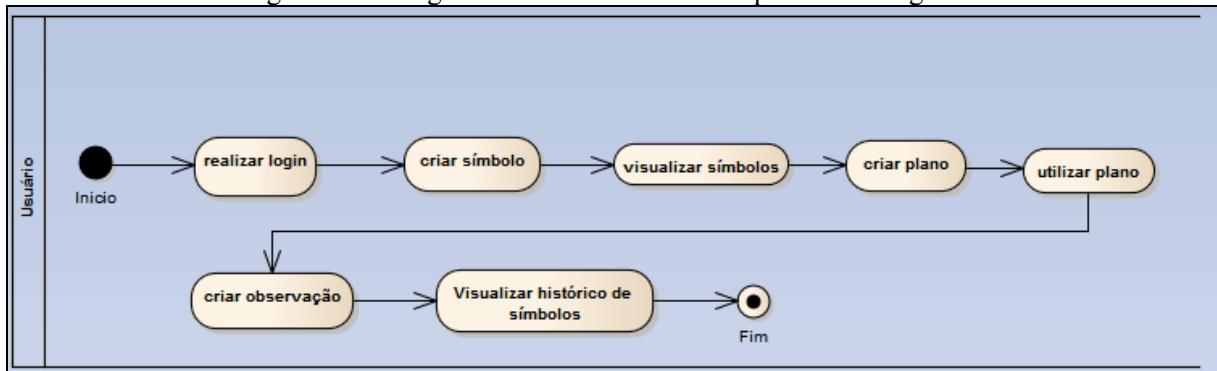
O primeiro passo para criar um plano é adicionar os símbolos que irão compô-lo. Esta seleção é feita através do toque nos `ImageViews` da tela. Ao tocar em um `ImageView` é

disparado o método `onClick(View v)` da classe `SymbolPositionListener`. Esse método irá instanciar o diálogo `SymbolCategoryDialog` que permitirá que o usuário selecione qual a categoria do símbolo que deseja adicionar ao plano. Ao selecionar uma categoria é retornado o *id* dela para a atividade através do método `onCategoryReturnValue(int position, long category)` que irá abrir a janela da classe `CategorySymbolDialog` com a chamada do método `show(FragmentManager fragmentManager, String tag)`. Esta janela irá exibir em um `GridView` todos os símbolos que o usuário possuir da categoria selecionada. Quando o usuário selecionar um dos símbolos, ele será retornado para a atividade através do método `onSymbolReturn` e então substituirá a imagem padrão do `ImageView`. Esse processo de seleção de símbolos poderá repetir-se até o usuário preencher todos os símbolos do *layout*. Após a seleção dos símbolos o usuário deverá tocar o *action button* `Adicionar Plano` que irá realizar a chamada do método `showPlanNameDialog()` para que seja aberta a janela para seleção do nome do plano através do método `show(FragmentManager fragmentManager, String tag)` da classe `PlanNameDialog`. Esta janela irá conter um `EditText` onde deverá ser colocado o nome do plano que será criado. Após preencher o nome o usuário deverá selecionar o botão `Criar Plano` que retorna através do método `onReturnName(String name)` o nome do plano. Este método irá chamar o método `SyncCreatedPlan(SparceArray<Symbol> symbolsPlan, String planName, Activity mContext, int layout)` da classe `SyncInformationController` que irá executar o método `execute()` da AsyncTask `SyncCreatedPlanTask`, que por fim irá enviar as configurações do plano criado para o servidor e inserir o plano na base local.

3.2.4 Diagrama de Atividades

O diagrama de atividades da Figura 27 apresenta de uma forma macro a utilização de todos os recursos do Tagarela sequencialmente.

Figura 27 – Diagrama de atividades do uso padrão do Tagarela



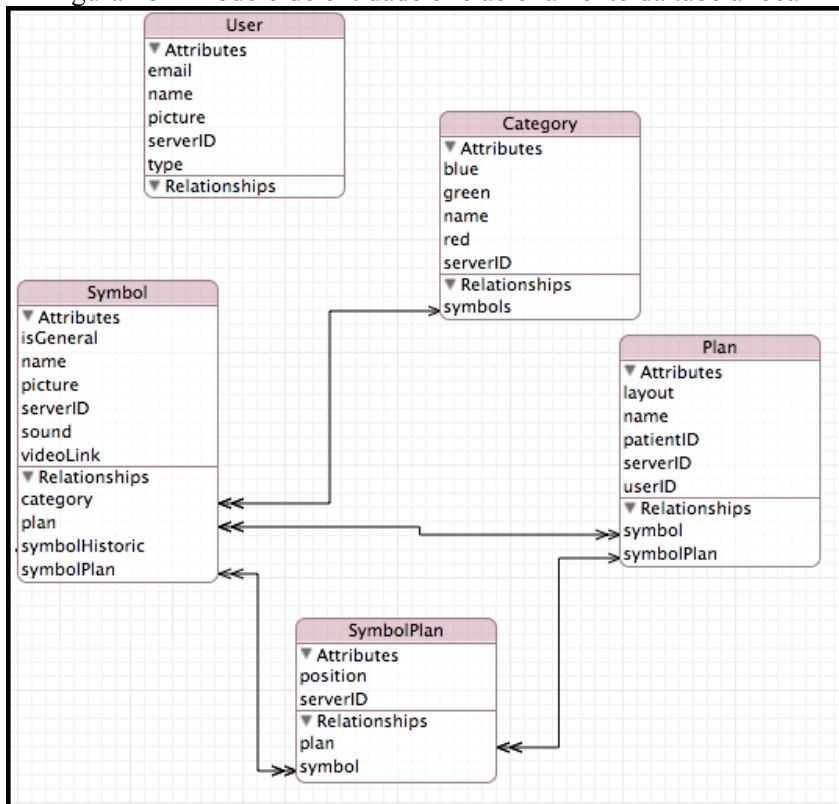
Conforme exposto na Figura 27, a inicialização do processo dá-se pelo *login* do usuário onde é inserido um *id* que será verificado pelo aplicativo e após a validação é apresentada a tela de boas-vindas. Então, o usuário cria um símbolo informando seus dados como: nome, categoria, imagem e som.

Após criar símbolos, o usuário visualiza-os através da janela de visualização de símbolos, podendo interagir com eles pelo toque e para que o aplicativo reproduza o som associado ao símbolo. Em seguida é iniciada a criação de plano, onde o usuário escolhe o *layout* e então começa a inserir símbolos, finalizando com a definição do nome e sua criação. Depois da criação do plano, o usuário pode interagir com o plano escolhido, registrar uma observação sobre a utilização do plano e por fim visualizar o histórico de símbolos utilizados.

3.2.5 Diagrama MER

A modelagem de dados do Tagarela é realizada através da ferramenta ORM greenDAO, desta forma as tabelas são mapeadas para classes em Java para sua manipulação. A representação das classes de modelo e a função de cada entidade podem ser vistas na seção 3.2.2.1. A Figura 28 apresenta como é o Modelo Entidade Relacionamento (MER) da aplicação.

Figura 28 – Modelo de entidade e relacionamento da tabela local



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas para o desenvolvimento do aplicativo e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O desenvolvimento do aplicativo foi realizado utilizando a linguagem Java junto da biblioteca de desenvolvimento para Android 4.1, conhecida por Jelly Bean ou também por API Level 17. O ambiente de desenvolvimento para a criação do trabalho foi o Eclipse em conjunto com o *plug-in* Android Development Tools (ADT). O ADT propicia diversos recursos de execução e depuração da aplicação para desenvolvedores. Por razões de praticidade, para depuração e execução de testes da aplicação não foi utilizado o emulador disponibilizado pelo ADT e sim um *tablet* da fabricante Samsung denominado Galaxy Note 2 10.1 modelo N8010.

O *tablet* Galaxy Note 2 10.1 possui processador *quad-core* Exynos 4412 com 1.4 Giga Hertz (GHz), processador gráfico Mali-400MP, memória RAM de 2 Giga Bytes (GB) e 16 GB de memória interna.

3.3.2 Gravação e reprodução de áudio

As tarefas de gravação e reprodução de áudio se mostraram muito eficientes com a utilização das APIs nativas do Android. Durante a implementação do trabalho houve a necessidade de mudar a codificação de áudio para que o som pudesse ser reproduzido tanto em dispositivos Android como em dispositivos com o sistema iOS, porém, a classe `MediaRecorder` é bastante dinâmica e torna-se simples modificar a codificação do áudio gravado.

A gravação de áudio é realizada na classe `SymbolCreateController` dentro do método `recordAudio(View view)`. O código fonte do método pode ser visualizado no Quadro 1 e será descrito em seguida.

Quadro 1 – Código fonte do método recordAudio

```

57  private void recordAudio(final View view) {
58      isRecording = true;
59      TextView rec = (TextView) view.findViewById(R.id.sound_rec);
60      rec.setText(R.string.rec_stop);
61      try {
62          if (mediaRecorder == null) {
63              mediaRecorder = new MediaRecorder();
64          } else {
65              mediaRecorder.reset();
66          }
67          mediaRecorder.setOnInfoListener(new OnInfoListener() {
68
69              @Override
70              public void onInfo(MediaRecorder mr, int what, int extra) {
71                  stopClicked(view);
72              }
73          });
74          mediaRecorder.set AudioSource(MediaRecorder.AudioSource.MIC);
75          mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
76          mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AAC);
77          mediaRecorder.setOutputFile(mfilePath);
78          mediaRecorder.setMaxDuration(5000);
79          mediaRecorder.prepare();
80          mediaRecorder.start();
81      } catch (Exception e) {
82          Log.e("MEDIA_RECORDER", "prepare() failed");
83      }
84  }
85 }
```

O método inicia definindo a variável de controle `isRecording` como `true`. Esta variável serve para que quando o usuário pressione o `TextView` parar gravação, o método `recordPressed()` faça a chamada para a finalização da gravação. Após isso é feita a edição do `TextView` que muda o texto para `Parar`, indicando que quando o usuário pressionar este componente novamente, será finalizada a gravação. Então, finalmente é iniciada a configuração do objeto da classe `MediaRecorder`. A classe `MediaRecorder` conforme visto na seção 2.4.1 pode ter sua configuração representada através de um diagrama de estados.

A configuração é iniciada com a verificação da instância do objeto. Em casos em que o objeto não esteja instanciado, o objeto receberá uma nova construção. Mas, caso não esteja nulo, para economizar recursos irá retornar para o estado inicial através da chamada do método `reset()` e então a configuração será refeita. A etapa inicial da configuração é avisar através da implementação do método `onInfo()` da interface `OnInfoListener` para que quando a gravação do som seja finalizada, seja chamado o método `stopClicked(View view)` e retorne o `TextView` ao estado inicial. Após isso, o microfone do dispositivo é definido como fonte de áudio utilizado para a gravação, através do método `set AudioSource(int audio_source)`. Em sequência, é definido o formato de saída de áudio para MPEG-4 no método `setOutputFormat(int output_format)` e a codificação para *Advanced Audio Coding* (AAC) para que a saída do áudio resulte em um arquivo com a extensão `.m4a`. Depois disso, é definido o diretório em que o arquivo será salvo através do

método `setOutputFile(String path)`. Em seguida, é utilizado o método `setMaxDuration(int milliseconds)` para que a gravação seja finalizada automaticamente quando atingir cinco segundos. Para finalizar, é realizada a chamada do método `prepare()` que irá preparar a classe para iniciar a gravação, que será realizada em seguida pelo método `start()`. O Quadro 2 apresenta o método para reprodução de áudio da classe `SymbolCreateControler`.

Quadro 2 – Código fonte do método `startPlaying()`

```

104 private void startPlaying() {
105    .isPlaying = true;
106     TextView rec = (TextView) view.findViewById(R.id.sound_play);
107     rec.setText(R.string.rec_stop);
108     mPlayer = new MediaPlayer();
109     try {
110         mPlayer.setDataSource(mfilePath);
111         mPlayer.prepare();
112         mPlayer.start();
113         mPlayer.setOnCompletionListener(new OnCompletionListener() {
114             @Override
115             public void onCompletion(MediaPlayer mp) {
116                .isPlaying = false;
117                 TextView playText = (TextView)
118                 view.findViewById(R.id.sound_play);
119                 playText.setText(R.string.play_sound);
120             }
121         });
122     } catch (IOException e) {
123         Log.e("MEDIA_RECORDER", "prepare() failed");
124     }
125 }
```

Assim como o método `startRecording()`, o primeiro passo é definir o valor de um atributo que irá controlar a chamada dos métodos quando o usuário pressionar o botão reproduzir ou parar, nesse caso o atributo `isPlaying` receberá `true`. Após isto, é realizada a alteração do *label* do `TextView` de reprodução para `Parar` e então é realizada a construção do objeto `MediaPlayer`. Segundo, é inicializada a configuração do objeto `MediaPlayer`. Primeiro é definido o caminho do arquivo que será reproduzido através do método `setDataSource(String path)`, depois é realizada a preparação do objeto pelo método `prepare()` e, por fim, é feito o início da reprodução através do método `start()`.

3.3.3 Envio de informações ao servidor

O envio de informações ao servidor é a rotina que envolve mais recursos de programação diferentes da versão inicial do Tagarela para iOS. Durante o envio de informações para o servidor são utilizados: *Threads* (`AsyncTask`), *WebServices*, codificação de *array* de *bytes* em Base64 e leitura de arquivos JSON.

O Quadro 3 apresenta a codificação da classe `SyncCreatedSymbolTask`, uma das responsáveis pelo envio de informações ao servidor.

Quadro 3 – Código fonte da classe SyncCreatedSymbolTask

```

21  public class SyncCreatedSymbolTask extends AsyncTask<String, Void, Void> {
22      private static final String SYMBOL_USER = "private_symbol[user_id]";
23      private static final String SYMBOL_SOUND =
24          "private_symbol[sound_representation]";
25      private static final String SYMBOL_IMAGE =
26          "private_symbol[image_representation]";
27      private static final String SYMBOL_CATEGORY = "private_symbol[category_id]";
28      private static final String SYMBOL_IS_GENERAL = "private_symbol[isGeneral]";
29      private static final String SYMBOL_NAME = "private_symbol[name]";
30      private static final String URL_SYMBOL_POST = "http://murmuring-falls-
31 7702.herokuapp.com/private_symbols/";
32      private ProgressDialog progress;
33      private Context mContext;
34      private Symbol symbol;
35
36      public SyncCreatedSymbolTask(Activity activity, Symbol symbol) {
37          this.mContext = activity;
38          this.symbol = symbol;
39      }
40
41      private static String audioEncoder(byte[] audio) {
42          return Base64Utils.encodeBytesToBase64(audio).replaceAll("\\"+, "@");
43      }
44
45      private static String imageEncoder(byte[] imagem) {
46          return
47          Base64Utils.encodeImageToBase64(BitmapFactory.decodeByteArray(imagem, 0,
48          imagem.length)).replaceAll("\\"+, "@");
49      }
50
51      @Override
52      protected void onPreExecute() {
53          progress = new ProgressDialog(mContext);
54          progress.setMessage("Aguarde...");
55          progress.show();
56      }
57
58      @Override
59      protected Void doInBackground(String... params) {
60          if (symbol != null) {
61              try {
62                  HttpPost post = new HttpPost(URL_SYMBOL_POST);
63                  post.addHeader("Accept", "application/json");
64                  post.addHeader("Content-Type", "application/x-www-form-
65                  urlencoded");
66                  final NameValuePairBuilder parametros =
67                      NameValuePairBuilder.novaInstancia();
68                  parametros.addParam(SYMBOL_NAME, symbol.getName());
69                  addParam(SYMBOL_IS_GENERAL, String.valueOf(0));
70                  addParam(SYMBOL_CATEGORY, String.valueOf(symbol.getCategoryID()));
71                  addParam(SYMBOL_IMAGE, imageEncoder(symbol.getPicture()));
72                  addParam(SYMBOL_SOUND, audioEncoder(symbol.getSound()));
73                  addParam(SYMBOL_USER,
74                      String.valueOf(MainActivity.getUsuarioLogado().getServerID()));
75                  HttpUtils.prepareUrl(post, parametros.build());
76                  HttpResponse response = HttpUtils.doRequest(post);
77                  if (response.getStatusLine().getStatusCode() == 201) {
78                      JSONObject returnSymbol = new
79                      JSONObject(HttpUtils.getContent(response));
80                      symbol.setServerID(returnSymbol.getInt("id"));
81
82                      DaoProvider.getInstance(null).getSymbolDao().insert(symbol);
83
84                  }
85              } catch (Exception e) {
86                  e.getMessage();
87                  e.printStackTrace();
88              }
89          }
90      }
91      return null;
92  }
93  @Override
94  protected void onPostExecute(Void unused) {
95      progress.dismiss();
96  }
97 }
```

O envio e recebimento de mensagens no aplicativo é realizado através de classes que estendem `AsyncTasks`. As `AsyncTasks` realizam pequenas operações em *background* durante a execução do aplicativo.

A execução de uma `AsyncTask` ocorre em quatro passos: `onPreExecute`, `doInBackground`, `onProgressUpdate` e `onPostExecute`. Na classe `SyncCreatedSymbolTask` o método `onProgressUpdate` não é implementado pois não é necessário que nenhuma atualização de progresso seja apresentada para o usuário.

O método `onPreExecute` encontrado na linha 52 do código, cria um `ProgressDialog` para que o usuário não possa realizar outra operação durante o envio do símbolo.

Após a execução do método anterior, é realizado o procedimento principal da classe no método `doInBackground` iniciado na linha 62. Primeiramente é consistido o objeto `symbol` que será enviado. Caso este não seja nulo a execução continua. Após a validação, na linha 65 é instanciado um objeto da classe `HttpPost` com a URL de envio do símbolo. Este objeto é responsável por enviar a requisição para o servidor. Em seguida, nas linhas 65 e 66, é atualizado o cabeçalho da requisição definindo o tipo de conteúdo que será enviado e a codificação utilizada. Então, das linhas 70 até 75, um objeto da classe `NameValuePairBuilder` é atualizado com atributos do símbolo que será enviado para o servidor. Para os parâmetros `symbol_image` e `symbol_sound` serem enviados, eles precisam ser transformados em textos de Base64. Para isso, as mídias são transformadas em *arrays* de `byte` e então são convertidas para texto por meio da classe `Base64` nos métodos `audioEncoder` e `imageEncoder` nas linhas 41 e 45. Quando todos os parâmetros estiverem preenchidos, são adicionados à requisição através do método `prepareURL` da classe utilitária `HTTPUtils` na linha 78. Finalmente na linha 79, a requisição é realizada para o servidor utilizando o método `doRequest(HttpPost post)` e retornando um objeto `HttpResponse` com a resposta do servidor. O código da resposta é verificado e se o retorno for igual ao código 201, significando sucesso, o símbolo começa a ser salvo na base local. Para inserir o símbolo localmente, primeiramente é montado um objeto JSON a partir do conteúdo da resposta do servidor. Então, é obtido o ID do símbolo retornado pelo servidor e o objeto `symbol` é atualizado para que seja inserido na base local pelo método `insert(Symbol symbol)` da classe `SymbolDAO` na linha 84.

O método `onPostExecute` da linha 94, é executado após a operação `doInBackground` ser finalizada e em sua execução fecha o `ProgressDialog` utilizando o método `dismiss()`.

3.3.4 Design Patterns

Na elaboração do projeto foram utilizados *design patterns* para resolver problemas comuns, adequar-se a API do Android e também para melhorar a legibilidade do código. Abaixo estão descritos os padrões utilizados e sua localização. Alguns padrões comuns como o *Composite* e o *Observer* não serão descritos, pois são padrões utilizados naturalmente mesmo que não exista conhecimento destes.

- f) *singleton*: padrão utilizado para manter apenas uma instância de um objeto durante o uso do aplicativo, implementado nas classes: `SyncInformationController` e `DaoProvider`.
- g) *adapter*: é utilizado para converter chamadas de um método para uma forma que outra classe ou API compreenda. No Tagarela foram criados adaptadores para personalizar a visualização de itens em `GridViews` e `ListsViews`. O padrão foi utilizado nas classes: `ImageAdapter`, `SymbolAdapter` e `ListViewSymbolAdapter`.
- h) *builder*: o padrão builder é utilizado nas classes de criação de diálogo através da *inner class* `Builder` da classe `FragmentDialog` e é utilizado para separar a construção do objeto da representação, podendo criar objetos diferentes de acordo com cada implementação.

3.3.5 Operacionalidade da implementação

As principais atividades do aplicativo desenvolvido estão relacionadas à criação de um plano de atividades e da interação do paciente com o plano criado. Porém, para o usuário realizar estas atividades é necessário que algumas etapas sejam previamente executadas, como a criação do usuário, a criação dos símbolos que irão compor o plano, e por fim criar o plano utilizando um *layout* adequado. Além disso, o usuário pode escrever e visualizar observações e também visualizar o histórico dos símbolos utilizados. Esta seção apresenta resumidamente a operacionalidade destas etapas.

3.3.5.1 Criação de usuários

Ao iniciar o aplicativo a janela de boas-vindas é a primeira a ser apresentada, questionando o usuário se deseja realizar *login* no aplicativo ou se deseja criar um novo. A Figura 29 mostra a janela de boas vindas do aplicativo.

Figura 29 – Janela de boas vindas do Tagarela

Bem vindo ao Tagarela

Você já possui um usuário no Tagarela?

Não	Sim
-----	-----

Então, o usuário seleciona uma das opções disponíveis. Na tela de *login* é apresentado um campo para a inserção do *login* e dois botões. O primeiro botão irá criar um novo usuário e o segundo confirma o *login* para o *id* inserido no campo da tela.

Para criar um novo usuário é necessário escolher qual o tipo de usuário será criado. Ao selecionar a opção *Criar novo usuário* é aberta a janela para a seleção do tipo do usuário. Então, após a seleção é exibida a janela de criação. A janela de criação contém os campos que compõem os dados do usuário. Após preencher todos os campos o botão *OK* confirma a criação do novo usuário. A Figura 30a apresenta a janela de *login* do usuário e a figura 30b apresenta a janela de criação. Em seguida a Figura 31 exibe a janela para a seleção do tipo do usuário.

Figura 30 – Janelas de *login* e criação de usuário

Login de usuário

ID do usuário

[Não tenho usuário](#) [Confirmar](#)

Informe os campos abaixo

Nome

Email

Senha

[Cancelar](#) [Salvar](#)

(a) Janela de *login* do Tagarela

(b) Janela de criação de usuários

Figura 31 – Janela de seleção de tipo de usuário

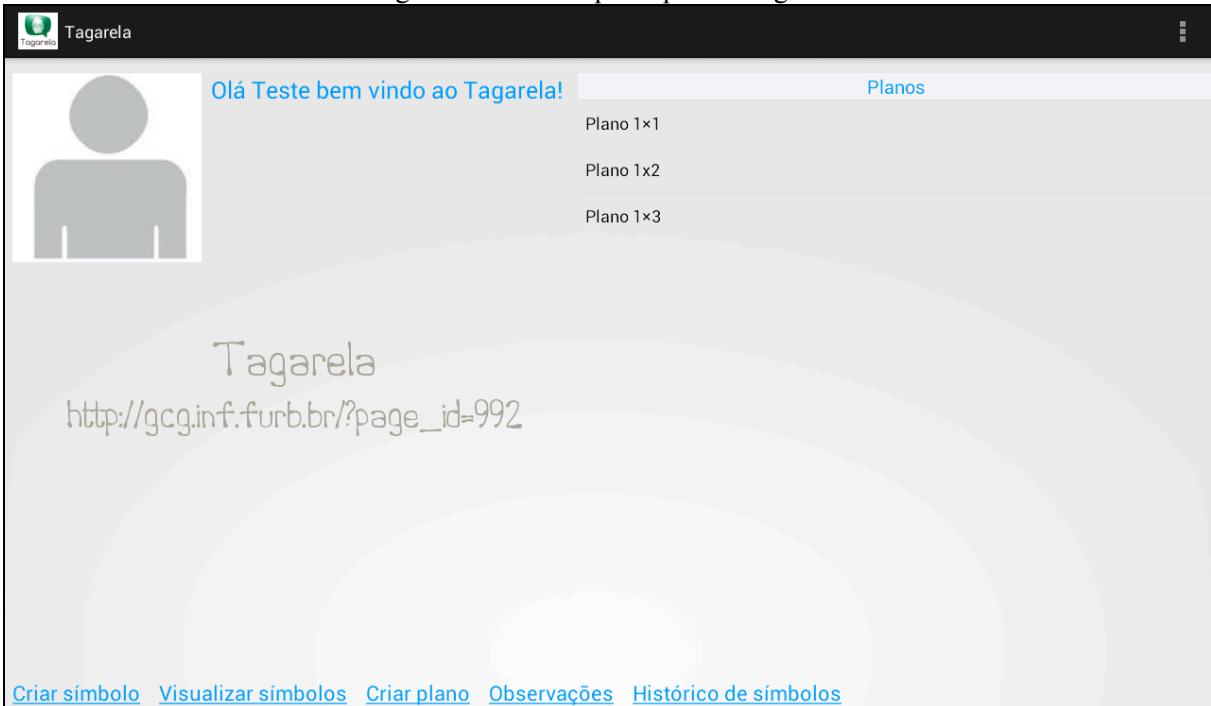
Selecionar tipo de usuário

ESPECIALISTA	PACIENTE	TUTOR
--------------	----------	-------

3.3.5.2 Tela principal

Após o usuário entrar no aplicativo é aberta a tela principal do Tagarela, onde são apresentadas as opções de Criar símbolo, Visualizar símbolos, Criar plano, Observações, Histórico de símbolos e selecionar um plano para utilização.

Figura 32 – Janela principal do Tagarela



3.3.5.3 Criação de símbolos

A criação de símbolos é realizada em duas etapas. A primeira etapa é a seleção da categoria à qual o símbolo pertence. A segunda é a definição das características do símbolo através dos atributos: significado da imagem, *link* para vídeo, categoria, imagem e áudio. Após inserir todos os atributos, o usuário cria o símbolo ao tocar o botão **Salvar**, ou cancela a sua criação ao tocar o botão **Cancelar**. A Figura 33(a) apresenta a janela com a lista para seleção da categoria e a Figura 33(b) apresenta a janela de criação de símbolos.

Figura 33 – Janelas para a criação de símbolo

<p>Seleciona uma categoria</p> <hr/> <p>Pessoas</p> <hr/> <p>Verbos</p> <hr/> <p>Substantivos</p>	 <div style="border: 2px solid green; padding: 2px; margin-top: 5px;">Comer</div> <hr/> <div style="margin-top: 5px;">Link para video</div> <hr/> <div style="margin-top: 5px;">Verbos</div> <hr/> <div style="text-align: right; margin-top: 10px;"> Gravar som Tocar </div> <div style="text-align: center; margin-top: 10px;"> Cancelar Salvar </div>
--	--

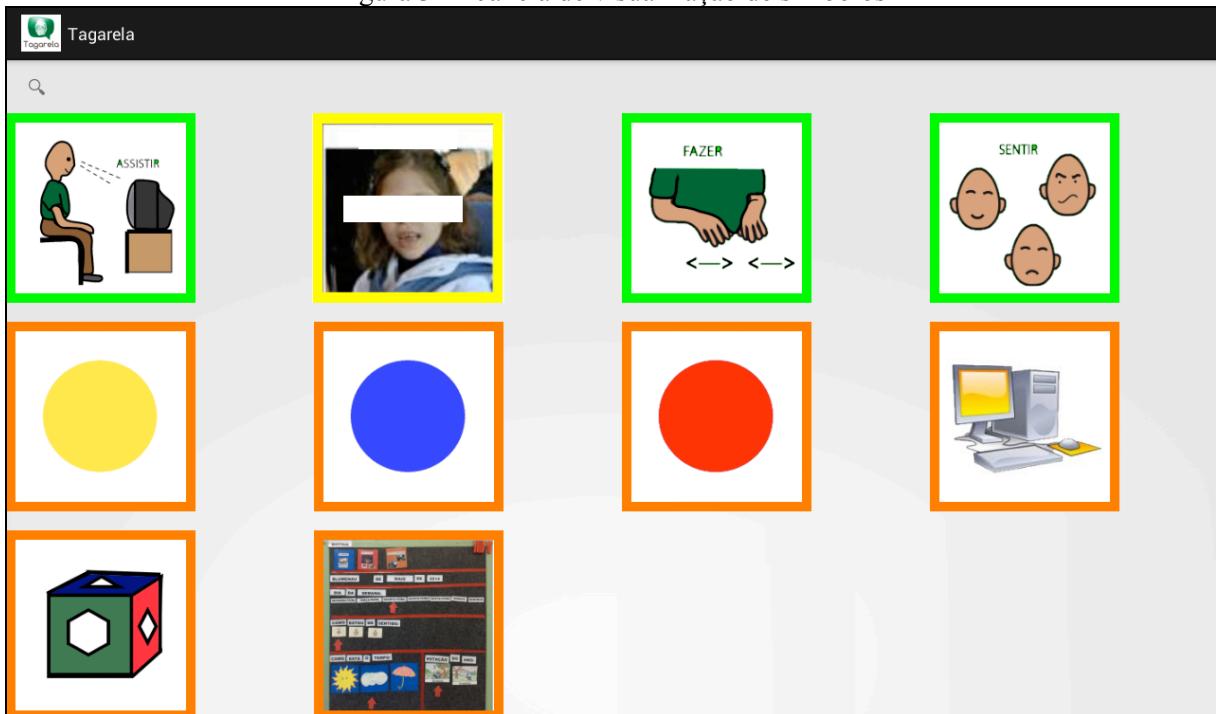
(a) janela de seleção de categoria

(b) janela de criação de símbolo

3.3.5.4 Visualização dos símbolos disponíveis

A partir da tela principal, o usuário pode também visualizar todos os símbolos disponíveis a ele. A categoria do símbolo pode ser identificada por meio da cor da borda apresentada ao redor da imagem. Além disso, o usuário pode também interagir com os símbolos por meio do toque, fazendo com que o som associado ao símbolo seja reproduzido. A Figura 34 expõe a janela de visualização de símbolos.

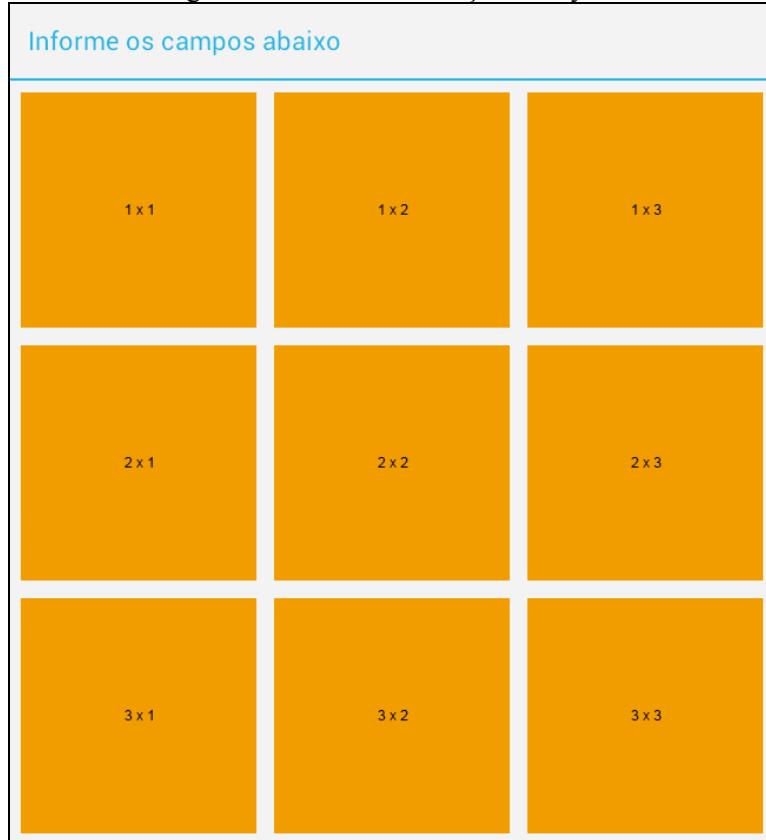
Figura 34 – Janela de visualização de símbolos



3.3.5.5 Criação de planos

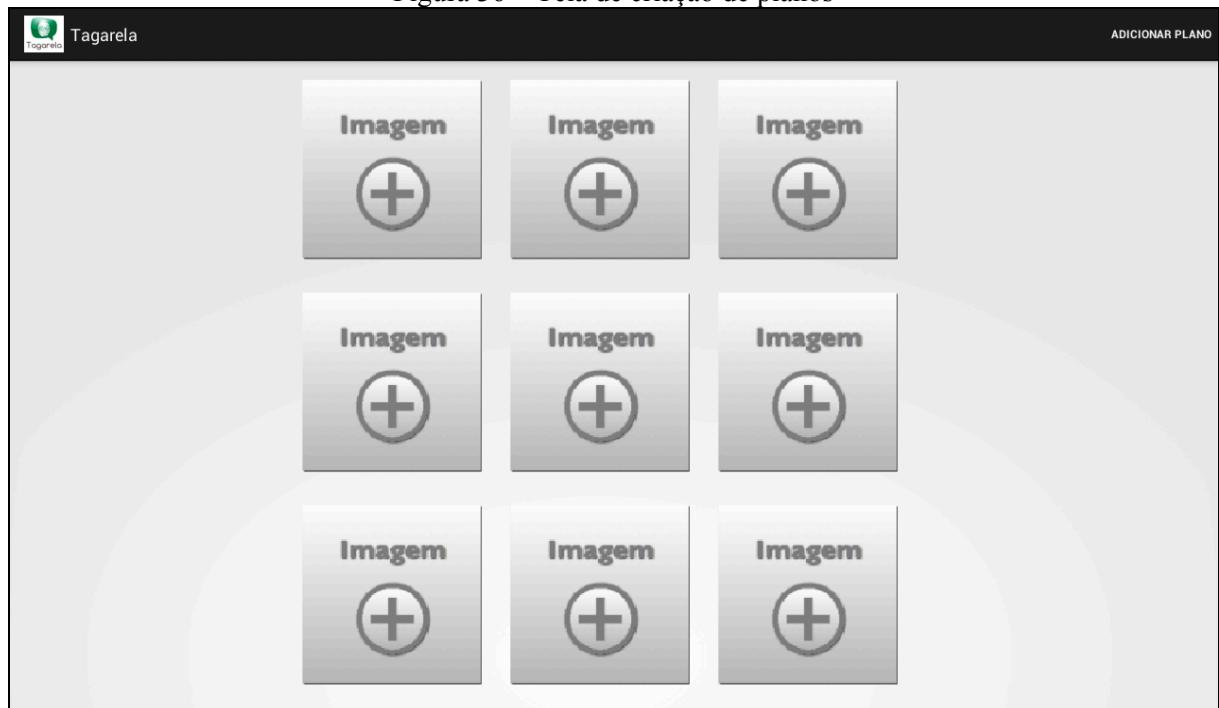
A criação de planos é uma das principais atividades do Tagarela. É onde são definidas as atividades do usuário e os símbolos que ele irá interagir. A partir da tela principal, o usuário pode acessar a tela de criação de planos ao clicar sobre o item **Criar plano**, então, será exibida a janela para a seleção do layout do plano conforme a Figura 35.

Figura 35 – Janela de seleção de layout



Após selecionar o layout, é exibida a tela de criação de planos, onde o usuário passa a inserir os símbolos que irão compor o plano, de acordo com o *layout* selecionado. A Figura 36 apresenta a tela de criação de planos.

Figura 36 – Tela de criação de planos

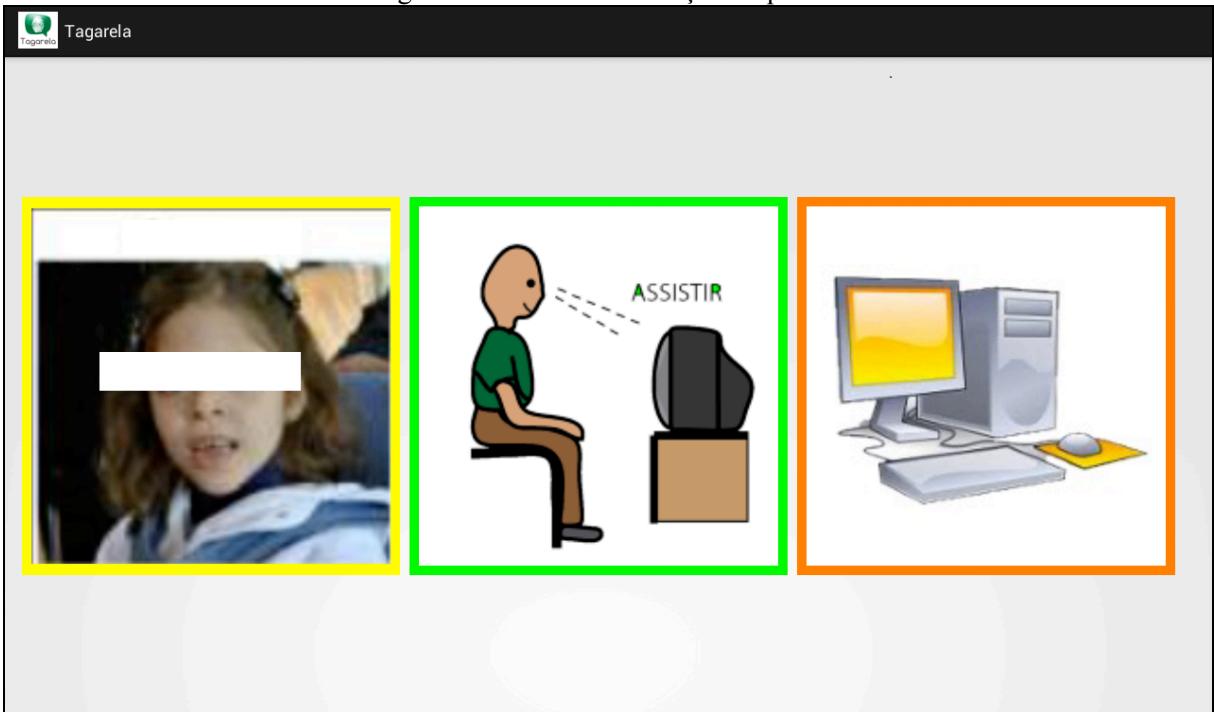


Depois de selecionar os símbolos que compõem o plano, o usuário seleciona o botão **Adicionar plano**, então, é exibida uma janela para inserir o nome do plano criado. Após inserir o nome, o usuário toca o botão **criar** e o plano é criado.

3.3.5.6 Utilização de pranchas

Para utilizar uma prancha já disponível ao usuário, deve-se selecionar na tela principal um plano. Quando o plano é selecionado é apresentada a tela de visualização de plano, onde o *layout* selecionado e todos os símbolos que compõem a prancha são apresentados. A Figura 37 apresenta a tela de utilização de planos.

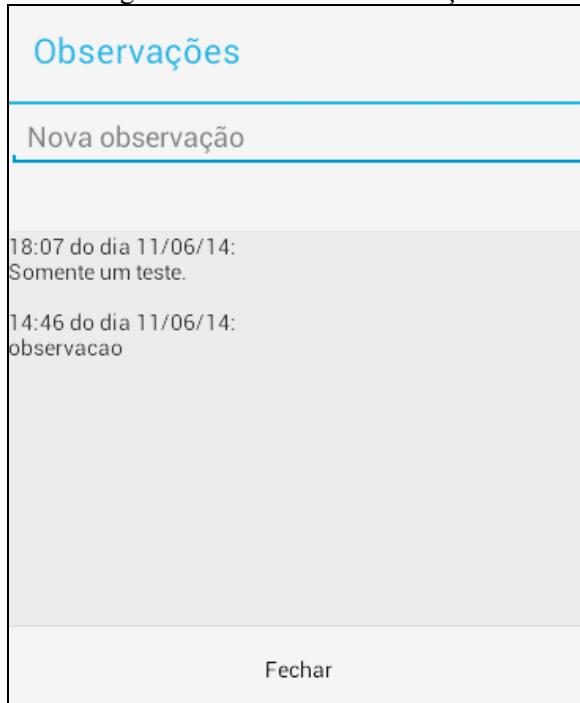
Figura 37 – Tela de utilização de planos



3.3.5.7 Visualização e criação de observação

No Tagarela, é possível visualizar e criar observações para o usuário. Este recurso é interessante por permitir uma interação entre as pessoas que auxiliam o paciente devido a sincronização de informações. Para visualizar as observações, o usuário deve selecionar a opção **Observações** na tela principal do aplicativo, então, uma janela é aberta com as observações salvas para o usuário. A janela também possui um campo de edição de texto que permite ao usuário digitar uma nova observação. Quando a janela é fechada, a observação é salva e sincronizada. A Figura 38 apresenta a janela de observações.

Figura 38 – Janela de observações



3.3.5.8 Histórico de símbolos

O Tagarela permite aos usuários visualizar o histórico dos símbolos utilizados. O acesso a este histórico é realizado pelo link [Histórico de símbolos](#) da tela principal. Em cada registro do histórico é exibido a imagem do símbolo, o nome e a data e hora em que foi utilizado. A Figura 39 apresenta a janela de histórico de símbolos.

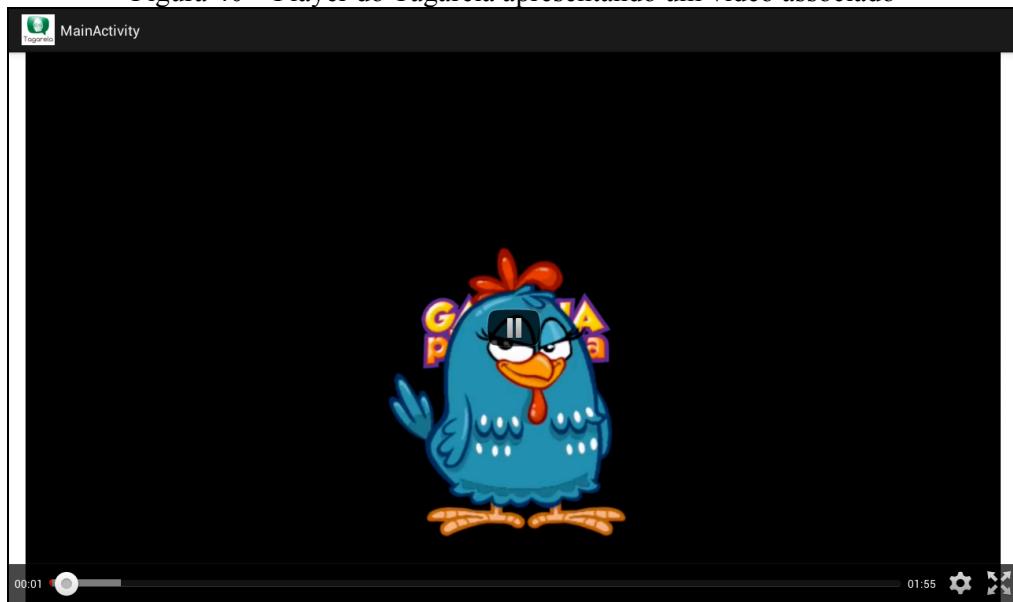
Figura 39 – Janela de histórico de símbolos



3.3.5.9 Visualização de vídeos do youtube

O Tagarela em Android, permite aos usuários reproduzirem os vídeos vinculados aos símbolos através da propriedade `link para vídeo`. Ao realizar um toque longo em um símbolo que contenha um vídeo associado, se o dispositivo estiver conectado à internet, o aplicativo abre um visualizador e inicia a reprodução do vídeo. A Figura 40 apresenta a janela de visualização do vídeo.

Figura 40 – Player do Tagarela apresentando um vídeo associado



3.4 RESULTADOS E DISCUSSÃO

Este trabalho apresenta o desenvolvimento do aplicativo de comunicação alternativa Tagarela na plataforma Android utilizando recursos da plataforma como: *webservices*, *asynctasks (Threads)*, componentes customizáveis e acesso a base de dados por meio de uma ferramenta de terceiros.

Para analisar os resultados do aplicativo, na seção 3.4.1 são apresentados os resultados da utilização do aplicativo no Centro Municipal de Educação Alternativa (CEMEA) da cidade de Blumenau e as considerações da professora de apoio permanente (PAP) da Escola Básica Municipal Machado de Assis.

Neste trabalho, buscou-se reproduzir o Tagarela em iOS para a plataforma Android. Para analisar os resultados obtidos, na seção 3.4.2 foram efetuados comparativos visuais entre a versão desenvolvida neste trabalho e a sua correspondente da plataforma iOS.

Na seção 3.4.3, foram realizados comparativos com os trabalhos correlatos e o desenvolvido. Por fim na seção 3.4.4 foram realizados testes de consumo de memória durante a execução de algumas funcionalidades do aplicativo.

3.4.1 Utilização da aplicação em ambiente real

A cidade de Blumenau conta com o CEMEA, que é ligado à Secretaria de Educação e atende crianças público alvo da Educação Especial (com deficiência e/ou transtornos globais do desenvolvimento) matriculadas na rede de ensino pública e privada do município. Hoje são atendidos aproximadamente 145 alunos neste espaço e os atendimentos ocorrem duas vezes por semana.

Para realizar a utilização neste ambiente (CEMEA), o aplicativo foi apresentado às coordenadoras do centro e em seguida, apresentado às fonoaudiólogas, identificadas aqui como Fonoaudióloga 1, Fonoaudióloga 2 e Fonoaudióloga 3, que apresentaram interesse no aplicativo e ofereceram-se a utilizar o Tagarela durante algumas duas seções com seus pacientes. A utilização foi realizada por dois pacientes com diagnósticos diferentes, aqui chamados de Paciente 1 e Paciente 2. O Paciente 1, é diagnosticado com falta de fala por razões psicológicas, o Paciente 2 é diagnosticado com autismo.

Para preparar o ambiente, foram criados os símbolos e planos antes do atendimento dos pacientes. Inicialmente foi necessário explicar como funcionava o aplicativo para utilizar as suas funcionalidades, porém, após alguns minutos o auxílio não foi mais necessário. Observou-se que o aplicativo mostra lentidão durante a sincronização e criação de símbolos e planos em conexões com a internet de baixa capacidade e isto compromete de certa forma a experiência do usuário ao utilizar o recurso.

Ao iniciar a sessão do Paciente 1, a fonoaudióloga apresentou o aplicativo e mostrou ao paciente como seriam as atividades com a utilização do Tagarela. Ele demonstrou curiosidade pelo aplicativo. Durante a utilização o paciente utilizou quatro pranchas diferentes e um total de 15 símbolos. Após a utilização, a fonoaudióloga registrou um histórico sobre a utilização e visualizou o histórico de símbolos utilizados na sessão. Foi identificada certa lentidão ao utilizar o histórico de símbolos. Esta lentidão deve-se a implementação do *adapter* utilizado para preencher o componente `ListView` do Android. Foram realizados ajustes na rotina, como a utilização de *cache* em uma estrutura `HashMap`, porém o desempenho não apresentou grandes melhorias.

A sessão realizada com o Paciente 2 diagnosticado com autismo demonstrou maior facilidade em compreender os símbolos quando foi utilizada uma prancha onde todos os símbolos eram fotos de objetos reais. A Fonoaudióloga 1, descreve que a utilização desse tipo de ferramenta estimula a concentração do paciente, o que é considerado um ótimo resultado para crianças diagnosticadas com autismo.

Foi apresentado às fonoaudiólogas um formulário para relatar suas experiências e conclusões sobre o Tagarela, estes formulários estão disponíveis no Apêndice B. Em geral, as fonoaudiólogas consideraram que o aplicativo atende as suas necessidades mesmo que parcialmente.

Quanto à utilização de som ao interagir com símbolos, as fonoaudiólogas consideram um recurso que ajuda muito durante a utilização do aplicativo, pois, o estímulo auditivo mantém o interesse das crianças para participar da atividade. Existem crianças que irritam-se com este tipo de estímulo, mas como o vínculo sonoro ao símbolo é opcional, o aplicativo permite contornar esta situação.

A facilidade de uso do aplicativo foi considerada boa em sua maioria. O motivo relatado pela Fonoaudióloga 1 e reforçado pela Fonoaudióloga 2 é que a forma como são apresentados os planos limita o uso destes a crianças alfabetizadas.

Entre as considerações positivas da Fonoaudióloga 1 sobre o aplicativo em Android, foi observado que o dinamismo para a criação dos símbolos ao permitir utilizar fotos reais e imagens, torna o aplicativo adaptável para diferentes diagnósticos de falha de comunicação. A CA pode ser usada tanto para diagnósticos psicológicos quanto para deficiências como o autismo, a paralisia cerebral e a deficiência intelectual. Cada um desses diagnósticos pode mudar a percepção do paciente quanto a utilização dos símbolos. Por exemplo o Paciente 1, consegue assimilar bem símbolos que utilizam desenhos para representar seu significado, pois sua fala é comprometida apenas por fatores psicológicos. O Paciente 2, autista, apresentou dificuldades para assimilar símbolos em que o significado não era representado em fotos com objetos reais. O apoio sonoro, também é considerado um fator que estimula atenção do paciente na atividade. Por fim, a Fonoaudióloga 1 cita a possibilidade de visualizar o histórico de símbolos utilizados como uma ferramenta útil para verificar como está sendo a utilização do aplicativo pelos pacientes em outros locais.

As considerações da Fonoaudióloga 3 relatam que a possibilidade de sincronização dos recursos no dispositivo é muito útil. A fonoaudióloga explica que certas vezes o profissional precisa criar um símbolo com foto de algo que o paciente tenha em casa, como por exemplo, seu quarto, a cozinha ou algum brinquedo. Muitas vezes o profissional não tem acesso a esse tipo de material, dessa forma, com a utilização do Tagarela, o profissional pode solicitar à família que crie símbolos de situações como essa. E o profissional, ao utilizar o aplicativo durante uma sessão pode sincronizar estes símbolos e ter acesso a eles. Outro ponto é o dinamismo do aplicativo, pois, permite criar planos e símbolos de acordo com as necessidades do paciente.

A Fonoaudióloga 2 considera entre os pontos positivos o estímulo da linguagem do paciente. A profissional reforça que linguagem é diferente da fala, pois ao utilizar um aplicativo como o Tagarela o paciente está estimulando seu olhar, gestos, fala e sua interação social. O aplicativo também estimula o vínculo, pois ao precisar de auxílio para utilizá-lo, o paciente cria um vínculo entre ele e o terapeuta. A fonoaudióloga afirma que mesmo que o profissional pare de utilizar este aplicativo, o vínculo será preservado.

Entre os pontos negativos e sugestões, o principal ponto foi a disposição dos planos aos usuários. A utilização de uma lista com os nomes dos planos restringe a utilização do aplicativo aos pacientes alfabetizados. A sugestão dada para contornar este problema é que na tela inicial, seja apresentado um símbolo que represente cada plano. Como exemplo, a Figura 60 encontrada no Apêndice C, apresenta símbolos que representam cada prancha contida no caderno de pranchas.

As fonoaudiólogas observaram a necessidade de poder percorrer a lista de pranchas, pois existem pacientes como o Paciente 1 que utiliza um caderno de comunicação, que é formado por várias pranchas organizadas em forma de caderno, no qual o usuário pode percorrê-lo para achar o símbolo desejado, as fotos do caderno utilizado pelo Paciente 1 podem ser encontradas no Apêndice C. As fonoaudiólogas, também sugeriram que fosse possível configurar a reprodução do som dos símbolos para tocar só após o paciente formar uma sentença completa, encadeando a fala. Pois, em estágios mais avançados de uso elas gostariam que o paciente assimile o significado da sentença por completo e não apenas a palavra. Por fim, a necessidade da internet pelo aplicativo foi relatado que em muitas situações pode ser um fator limitador para o uso do aplicativo. Este ponto já foi corrigido e o aplicativo permite o uso da maioria das funcionalidades do aplicativo sem conexão com a internet.

O projeto Tagarela, atualmente busca atender aos três primeiros estágios do PCA. Mas, conforme exposto no capítulo 2.2, as sugestões de encadeamento de fala e utilização de um caderno de comunicação fazem parte do quarto estágio.

O aplicativo também foi apresentado à PAP da Escola Básica Municipal Machado de Assis. A professora também considera importante a assimilação das figuras a seus correspondentes em objetos reais, e ressaltou que a utilização de figuras para representar objetos deve ser utilizada visando a representação atual do objeto para o paciente. Por exemplo, a representação de uma televisão não deve utilizar uma figura com o antigo modelo CRT e sim o modelo em tela plana, pois atualmente este é o padrão mais comum. Outro ponto observado é a forma como são apresentadas as ações em figuras onde a descrição é

fazer uma ação. É comum estas figuras apresentarem a ação após ter sido executada, segundo a professora, isso atrapalha o entendimento. Na última observação encontrada, a PAP relata que os símbolos correspondentes aos verbos (ações) parecem muito com a representação da ação em libra, o que torna o entendimento mais abrangente.

3.4.2 Comparação de interface gráfica

Um dos principais objetivos propostos na criação deste trabalho era que o resultado do aplicativo desenvolvido fosse visualmente semelhante à sua versão na plataforma iOS. São destacados nesta seção a comparação das telas na plataforma iOS com as telas da plataforma Android. As figuras de 41 a 52 mostram as comparações. Ao lado esquerdo estão as telas na plataforma iOS a direita na plataforma Android.

Figura 41 – Comparação das janelas iniciais do aplicativo

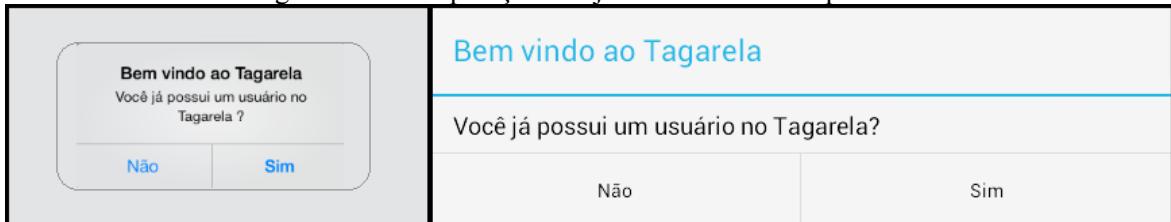


Figura 42 – Comparação das janelas de seleção de tipo de usuário



Figura 43 – Comparação das janelas de *login* de usuário

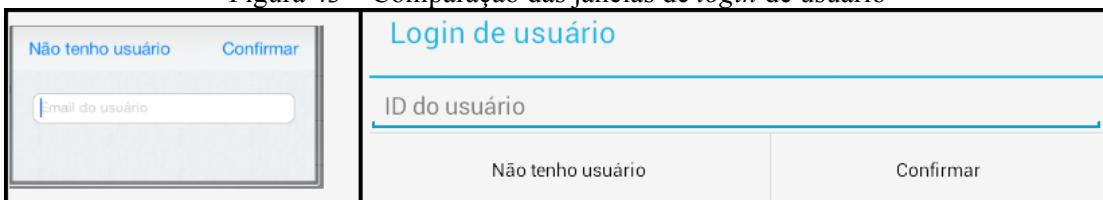


Figura 44 – Comparação das janelas de criação de usuário

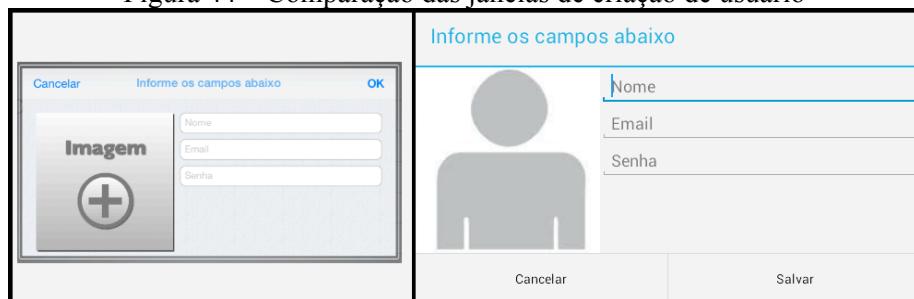


Figura 45 – Comparação das telas principais do aplicativo

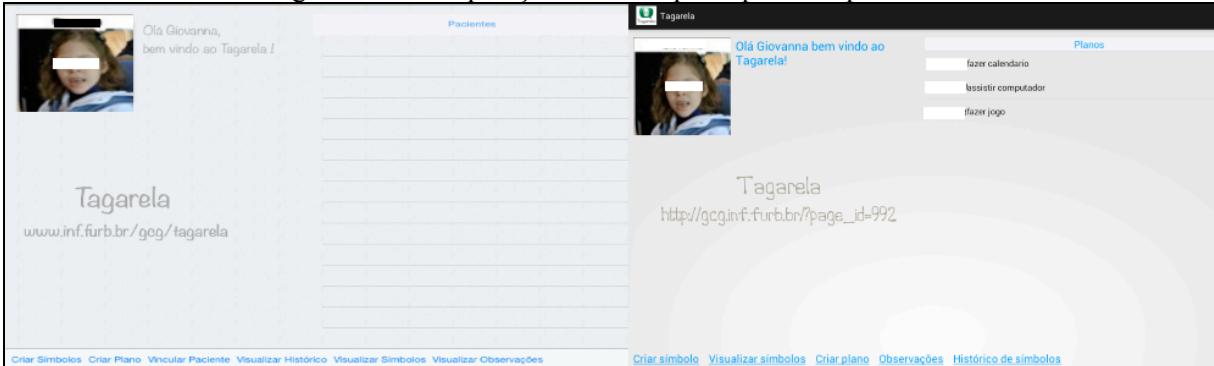


Figura 46 – Comparação das janelas de seleção de categorias de símbolo

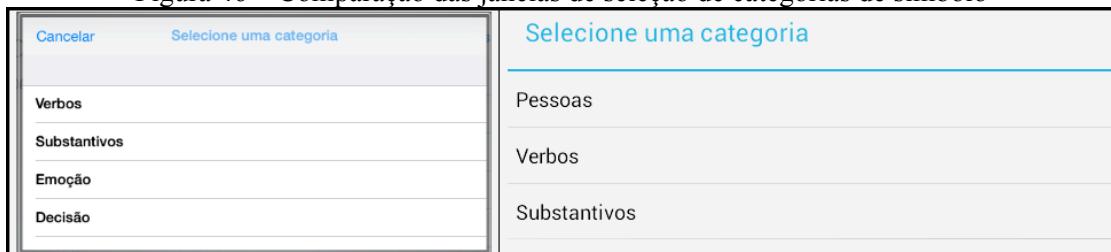


Figura 47 – Comparação das janelas de criação de símbolos do aplicativo



Figura 48 – Comparação das janelas de visualização de símbolos

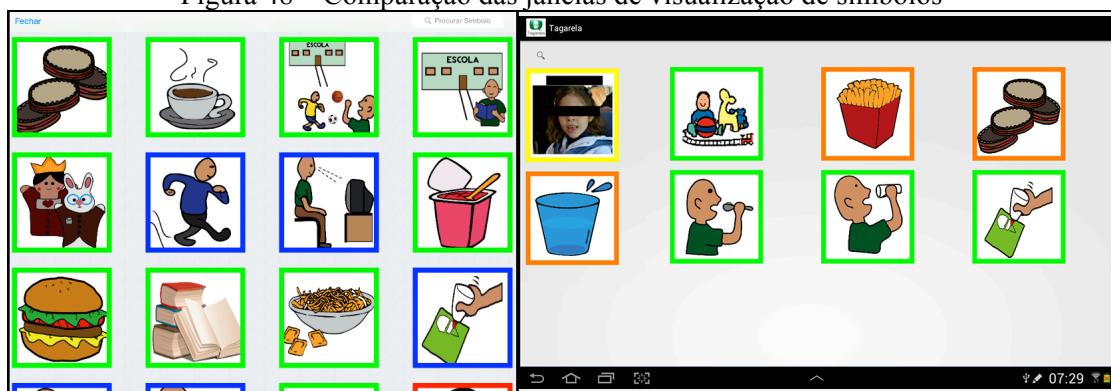


Figura 49 – Comparação das janelas de seleção de *layout* para novo plano



Figura 51 – Comparação das telas de criação de planos com *layout* 3 x 3

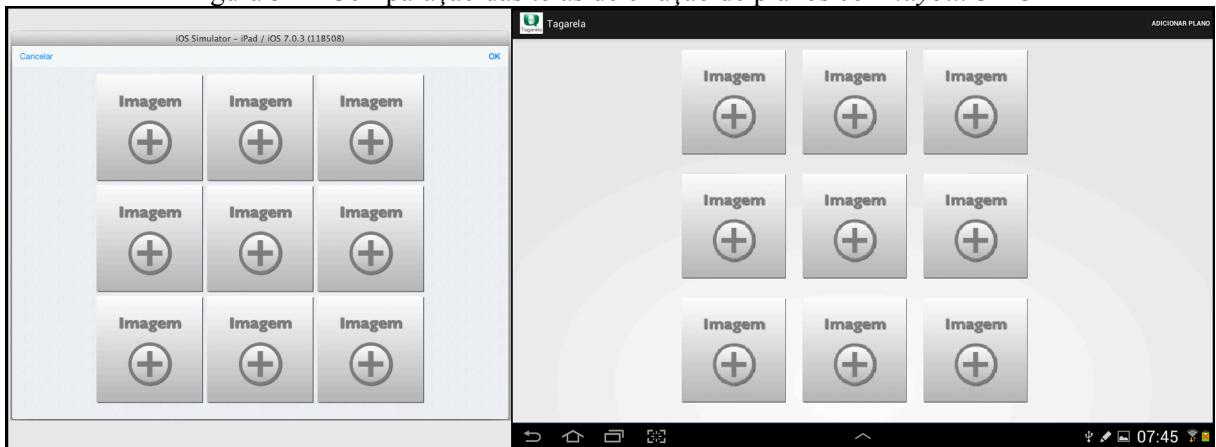
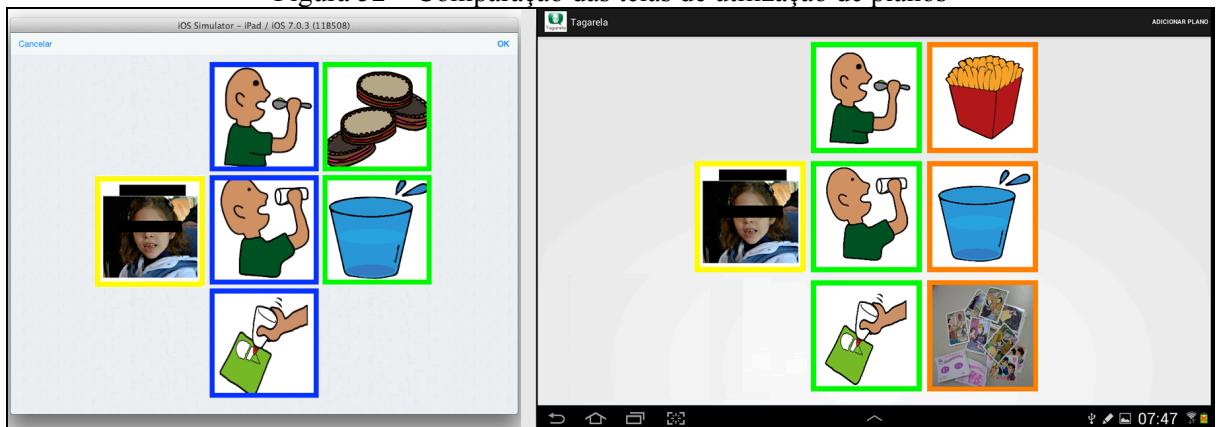


Figura 52 – Comparação das telas de utilização de planos



Como pode ser visto nas imagens acima, conclui-se que não ocorreram mudanças significativas na interface do aplicativo entre a plataforma iOS e Android.

3.4.3 Comparação entre o trabalho desenvolvido e os trabalhos correlatos

Nesta seção são abordadas as características e funcionalidades do trabalho desenvolvido em comparação com suas versões em iOS e os trabalhos correlatos, as quais

estão ilustradas no Quadro 4. Este quadro reúne as características utilizadas por Fabeni(2012) e também novas características implementadas no Tagarela em suas versões Android e iOS.

Quadro 4 – Comparação do Tagarela em Android com as versões iOS e trabalhos correlatos

Características / Trabalhos Correlatos	Tagarela Android	Tagarela iOS v1	Tagarela iOS v2	JABTalk	Tap2Talk	Proloquo2Go
criação de perfil de usuário		X*	X			
associação de áudio a símbolos	X	X	X	X	X	X
criação de planos de atividade	X	X	X	X	X	X
troca de mensagens entre os envolvidos		X				
possibilidade de impressão das pranchas		X				X
histórico de observações do paciente	X	X	X			
histórico de uso dos símbolos	X	X	X			
criação de símbolos via WEB	X**		X**			
sincronização de dados do usuário com um servidor	X		X			
seleção de layout de planos	X		X			
Permite a visualização de vídeos	X					

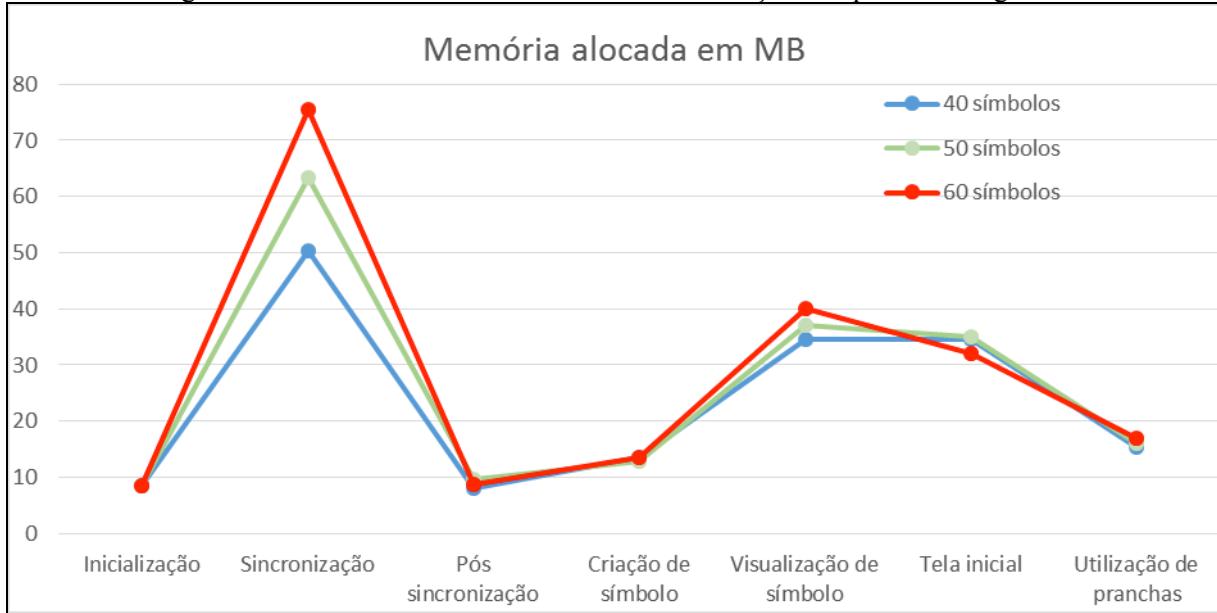
* A utilização de perfis na primeira versão do iOS não era eficaz;
 ** O Tagarela WEB está em desenvolvimento, a primeira versão em iOS não sincroniza dados com o servidor, logo, não permite a utilização de símbolos criados via WEB.

Através do exposto, conclui-se que o Tagarela está à frente dos outros projetos de comunicação alternativa para dispositivos móveis analisados como trabalhos correlatos. É possível identificar também que apesar de ter muitas funcionalidades, o Tagarela em Android ainda precisa desenvolver a interação entre os usuários através da utilização de perfis para que alcance sua versão iOS mais atual.

3.4.4 Utilização de memória e desempenho

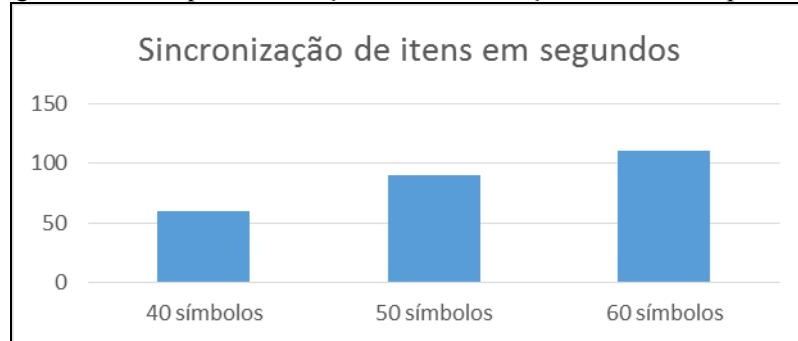
Os testes de consumo de memória do aplicativo foram realizados durante sua utilização em algumas funcionalidades. Foi verificado o consumo de memória com quantidades diferentes de símbolos e na seguinte ordem: ao iniciar o aplicativo, ao sincronizar os dados do usuário, após a sincronização, ao visualizar símbolos, retornar para a tela principal e, finalmente, ao utilizar um plano. A Figura 53 apresenta um gráfico contendo os resultados obtidos.

Figura 53 – Consumo de memória durante a utilização do aplicativo Tagarela



Através da figura é possível observar que existe alterações relevantes na alocação de memória durante a sincronização dependendo da quantidade de símbolos sincronizados. Um dos motivos identificados para essa diferença é a chamada ao *webservice*, que retorna de uma só vez todos os símbolos encontrados no servidor sem a distinção de usuário. Este retorno é interpretado como um único arquivo JSON, que contém todos os símbolos, imagens e áudios do servidor, tendo o seu tamanho proporcional ao número de símbolos. A Figura 54 expõe o tempo de execução dos métodos de sincronização do aplicativo.

Figura 54 – Tempo de execução de sincronização de itens do aplicativo



Assim como a alocação de memória, o tempo de transferência é proporcional ao número de itens e ao tamanho das imagens de cada símbolo. Observou-se que uma *query* realizada diretamente na base de dados do servidor o tempo de consulta é superior a 10 segundos. Os testes de sincronização foram realizados em uma internet banda larga com 10Mbps.

4 CONCLUSÕES

Neste trabalho foi desenvolvida uma versão do projeto Tagarela na plataforma Android. O projeto tem como objetivo criar ferramentas de comunicação alternativa através do uso de tecnologia assistiva.

Através da utilização do aplicativo por fonoaudiólogos, foi possível realizar uma análise qualitativa que indica que o aplicativo pode atender à necessidade dos profissionais e de pacientes com diferentes diagnósticos. Diante disso, conclui-se que o objetivo de criar uma versão do Tagarela para Android foi alcançado com sucesso.

Entre os pontos altos da implementação deste trabalho está a sincronização de dados com o servidor. A sincronização permite que os usuários consigam utilizar o aplicativo em diferentes dispositivos sem perder os dados, pois, todos são sincronizados durante a inicialização do aplicativo. A customização do *layout* dos planos foi um novo recurso implementado e permite que as atividades do paciente se adequem melhor a cada necessidade. A implementação do vínculo de vídeos aos símbolos permite que o aplicativo também seja utilizado como um reproduutor de fácil acesso a pessoas com deficiência.

Durante a implementação observou-se alguns fatos interessantes. A facilidade da utilização das APIs da plataforma Android, a importância e os benefícios da utilização das ferramentas ORM para desenvolvimento móvel. Graças a facilidade de utilização das bibliotecas de mídia do Android, foi possível ler e gravar os recursos de áudio dos símbolos de forma compatível com dispositivos da Apple. Já a ferramenta ORM greenDAO foi fundamental para a manipulação de dados durante o desenvolvimento do aplicativo, abstraindo um dos pontos que acreditava-se ser complexo na implementação.

Contudo, é indispensável falar das dificuldades de uso e instabilidade do emulador de dispositivos Android fornecidos pelo ADT. O desenvolvimento de um aplicativo com a utilização do simulador é inviável e os resultados obtidos não são confiáveis.

Para finalizar, espera-se que o projeto Tagarela continue em constante evolução e aperfeiçoamento, pois a iniciativa claramente irá contribuir muito para todas as pessoas envolvidas no processo de aprendizado dos portadores de deficiência.

4.1 EXTENSÕES

Sugerem-se as seguintes extensões para trabalhos futuros:

- a) permitir a utilização de perfis para cada tipo de usuário como: especialista, tutor e paciente;
- b) permitir que o usuário faça a movimentação dos símbolos nas pranchas;

- c) adaptar o aplicativo para maior diversidade de telas;
- d) melhorar o desempenho durante a sincronização de itens;
- e) atualizar a forma de exibição de planos;
- f) permitir encadear a reprodução de áudio.

REFERÊNCIAS BIBLIOGRÁFICAS

- ADA. **Americans with disabilities ACT of 1990.** [S.I.], 1990. Disponível em: <<http://www.ada.gov/pubs/ada.htm>>. Acesso em: 18 abr. 2012.
- APPLE INC. **Proloquo2Go.** [S.I.], 2012. Disponível em: <<http://itunes.apple.com/us/app/proloquo2go/id308368164?mt=8>>. Acesso em: 18 abr. 2012.
- BERSCH, Rita et al. **Atendimento educacional especializado – Deficiência Física.** Brasília, 2007. Disponível em: <http://portal.mec.gov.br/seesp/arquivos/pdf/aee_df.pdf>. Acesso em: 30 mai. 2012.
- FRANÇA, Rodrigo. **Aplicativos de tecnologia assistiva.** Blumenau, FURB, 23 mar. 2012. Entrevista concedida a Darlan De Marco.
- FABENI, Alan. F. C. **Tagarela:** aplicativo para comunicação alternativa no ios. Blumenau, FURB, 26 jul. 2013. Entrevista concedida a Darlan De Marco.
- FABENI, Alan. F. C. **Tagarela:** aplicativo para comunicação alternativa no ios. 2012. 106f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.
- GREENDAO. **greenDAO - android ORM for SQLite.** [S.I.], 2013. Disponível em: <<http://greendao-orm.com/blog/>>. Acesso em: 01 de dez. de 2013.
- GOOGLE. **JABStone.** [S.I.], 2012a. Disponível em: <https://play.google.com/store/apps/details?id=com.jabstone.jabtalk.basic&feature=search_result#t=W251bGwsMSwxLDEsImNvbS5qYWJzdG9uZS5qYWJ0YWxrLmJhc2ljIl0>. Acesso em: 05 mai. 2012.
- _____. **What is Android?** [S.I.], 2012b. Disponível em: <<http://developer.android.com/guide/basics/what-is-android.html>>. Acesso em: 15 maio 2013.
- JABSTONE. **JABstone:** giving every child a voice. [S.I.], 2012. Disponível em: <<http://www.jabstone.com/>> Acesso em: 18 abr. 2012.
- NATIONS, United. **Factsheet on persons with disabilities.** [S.I.], 2013. Disponível em: <<http://www.un.org/disabilities/default.asp?id=18>>. Acesso em: 13 maio 2014.
- OLIVEIRA, Luiza M. B. **Cartilha do censo 2010 - pessoas com deficiência.** Brasília, 2012. Disponível em: <<http://www.pessoacomdeficiencia.gov.br/app/sites/default/files/publicacoes/cartilha-censo-2010-pessoas-com-deficienciareduzido.pdf>>. Acesso em Jun. 2014.
- PELOSI, Miryam. **Tecnologia assistiva.** Rio de Janeiro, 2011. Disponível em: <<http://www.comunicacaoalternativa.com.br/tecnologia-assistiva>>. Acesso em: 18 mar. 2012.
- RADABAUGH, Mary P. **Study on the financing of assistive technology devices of services for individuals with disabilities - a report to the president and the congress of the United State, National Council on Disability.** Washington, 1993. Disponível em: <<http://www.eric.ed.gov/PDFS/ED355696.pdf>>. Acesso em: 18 abr. 2012.
- SARTORETTO, Mara L.; BERSCH, Rita. **O que é tecnologia assistiva?** Porto Alegre, 2012. Disponível em: <<http://www.assistiva.com.br>>. Acesso em: 20 abr. 2012.

TAGARELA. **Tagarela**: rede social de comunicação alternativa. Blumenau, 2014. Disponível em: <<http://gcg.inf.furb.br/tagarela>>. Acesso em: 10 maio 2014.

TAPTOTALK. **TapToTalk**: give your child a voice. [S.I.], 2014. Disponível em: <<http://www.taptotalk.com/>>. Acesso em: 15 jul. 2014.

APÊNDICE A – Descrição dos casos de uso, cenários e fluxos dos casos de uso

Os casos de uso do aplicativo Tagarela estão descritos a seguir:

- a) UC01 – Criar usuários, este caso de uso permite criar um novo usuário selecionando o tipo de perfil desejado (especialista, tutor ou paciente). Apresenta os seguintes campos para que o usuário preencha: nome, e-mail e senha. O aplicativo também vai permitir que o usuário escolha uma imagem do dispositivo ou tire uma nova foto para colocar no perfil. Após a criação o aplicativo envia os dados do usuário para o servidor (UC21).
- b) UC02 – Criar símbolos, este caso de uso permite criar os símbolos que futuramente serão utilizados para a criação dos planos (UC03). Os símbolos que são criados pelo usuário, são de uso exclusivo daquele usuário, ou seja, não deve ser exibido para outros usuários do Tagarela. O acesso à janela de criação de símbolos deverá ser feito através do link `criar símbolo` da tela principal do Tagarela. Para criar um símbolo, o usuário deverá preencher os seguintes campos: nome, categoria, áudio (opcional) e imagem.
- c) UC03 - Criar planos, este caso de uso permite criar planos com os símbolos disponíveis. O acesso à janela de criação de símbolos deverá ser feito após a seleção do *layout* do plano ao clicar no link `criar plano` da tela principal do Tagarela. Para criar um plano o usuário deverá escolher um *layout* (UC05), e em seguida adicionar os símbolos ao *layout* daquele plano. Após concluir a inclusão de símbolos o usuário deverá preencher o nome do plano e então o plano será criado. Após a criação, o plano deverá ser enviado ao servidor (UC23).
- d) UC04 – Criar observação, este caso de uso permite criar observações para o usuário. O acesso à janela de observação deverá ser feito através do link `Histórico de observações` da tela principal do Tagarela. Para criar uma observação o usuário deverá abrir a janela de histórico de observações e preencher o `TextEdit` com a nova observação, ao clicar no botão `Salvar observação` a informação é salva. Após a criação, a observação deverá ser enviada ao servidor (UC24).
- e) UC05 – Selecionar o *layout* do plano, este caso de uso permite selecionar o *layout* do plano que será criado. A janela de seleção de *layouts* deverá ser acessada ao clicar no link `criar plano` da tela principal do Tagarela. Nessa janela o usuário

poderá escolher *layouts* com até nove símbolos sendo o limite de três símbolos por linha/coluna. Após a seleção do *layout* o usuário inicia a criação do plano (UC03). Os *layouts* estarão disponíveis no formato “linha e coluna” como por exemplo: 1x1 (para apenas um símbolo na primeira linha), 2x1 (para dois símbolos na primeira linha), 2x2 (para quatro símbolos, dois na primeira linha e dois na segunda linha) e 3x3 (para três símbolos por linha/coluna sendo o máximo permitido).

- f) UC06 – Visualizar lista de planos, este caso de uso permite visualizar uma lista com todos os planos do usuário. A lista deverá ser exibida na tela inicial do usuário e a descrição dos planos na lista deverá ser o atributo nome.
- g) UC07 – Visualizar símbolos. Este caso de uso permite visualizar os símbolos criados. A tela de visualização de símbolos deverá ser acessada ao clicar no link *visualizar símbolos* da tela principal do Tagarela. Após isso, uma janela deverá ser exibida com os símbolos do usuário e ao redor dos símbolos uma borda da cor de sua categoria (UC09).
- h) UC08 – Interagir com símbolos. Este caso de uso permite interagir com os símbolos através do toque. Ao tocar um símbolo através da tela de *visualização de planos ou visualização de símbolos* o aplicativo deverá reproduzir o som associado ao símbolo (UC10) e gerar o histórico de utilização do símbolo (UC11).
- i) UC09 – Exibir borda em símbolos. Este caso de uso estabelece que ao exibir um símbolo, ele deverá ser contornado com uma borda com a cor da sua categoria (UC02).
- j) UC10 – Tocar som associado ao símbolo. Este caso de uso estabelece que ao interagir com um símbolo (UC08), o aplicativo deverá tocar o áudio utilizado durante a criação do símbolo (UC02).
- k) UC11 – Registrar histórico de símbolos utilizados. Este caso de uso estabelece que ao interagir com um símbolo (UC08), o aplicativo deverá gerar um histórico de utilização na base local com as informações do símbolo utilizado, data e hora da utilização. Após isso, o aplicativo deve enviar este histórico para o servidor (UC20).
- l) UC12 – Visualizar observações. Este caso de uso permite visualizar as observações do usuário. O acesso à janela de observação deverá ser feito através do link *Histórico de observações* da tela principal do Tagarela. As informações exibidas para o usuário devem ser a hora e minuto que a observação

foi criada, o dia e a descrição da observação.

- m) UC13 – Visualizar o histórico dos símbolos utilizados. Este caso de uso permite visualizar o histórico de utilização dos símbolos do usuário. O acesso à janela de observação deverá ser feito através do link Histórico de símbolos da tela principal do Tagarela. A janela deve trazer uma lista com a imagem do símbolo, o nome, a hora e a data de utilização.
- n) UC14 – Realizar *login*. Este caso de uso permite realizar *login* no sistema utilizando um usuário já existente através do seu ID. Após a validação do *login*, o sistema deve receber do servidor os dados do usuário (UC17), símbolos (UC16), planos (UC15), observações (UC18) e histórico de símbolos do usuário (UC19).
- o) UC15 – Receber os planos do usuário do servidor. Este caso de uso permite que ao realizar o *login* (UC14), o aplicativo deverá realizar a sincronização dos planos do usuário. As informações sincronizadas serão: nome do plano, símbolos e *layout*. Após receber as informações, o aplicativo deve verificar se as informações ainda não existem na base local, em caso negativo, deve salvar o plano localmente.
- p) UC16 – Receber os símbolos do usuário do servidor. Este caso de uso permite que ao realizar o *login* (UC14), o aplicativo deverá realizar a sincronização dos símbolos do usuário. As informações sincronizadas serão: nome, categoria, id, imagem e áudio. Após receber essas informações o aplicativo deve verificar se as informações ainda não existem na base local, em caso negativo, deve salvar o símbolo localmente.
- q) UC17 – Receber dados do usuário armazenados no servidor. Este caso de uso permite que ao realizar o *login* (UC14), o aplicativo deverá realizar a sincronização das informações do usuário. As informações sincronizadas serão: nome, id, foto e perfil de usuário. Após receber essas informações o aplicativo deve verificar se as informações ainda não existem na base local, em caso negativo, deve salvar os dados localmente.
- r) UC18 – Receber observações do usuário do servidor. Este caso de uso permite que ao realizar o *login* (UC14), o aplicativo deverá realizar a sincronização das observações criadas para o usuário. As informações sincronizadas serão: descrição da observação, data de utilização e id. Após receber essas informações o aplicativo deve verificar se as informações ainda não existem na base local, em caso negativo, deve salvar as observações localmente.
- s) UC19 – Receber histórico de símbolos do usuário do servidor. Este caso de uso

permite que ao realizar o *login* (UC14), o aplicativo deverá realizar a sincronização do histórico dos símbolos utilizados pelo usuário. As informações sincronizadas serão: id do símbolo, data de utilização e id. Após receber essas informações o aplicativo deve verificar se as informações ainda não existem na base local, em caso negativo, deve salvar o histórico localmente.

- t) UC20 – Enviar histórico do símbolo para o servidor. Este caso de uso estabelece que ao gerar o histórico de utilização do símbolo (UC11), o aplicativo deve enviar os dados: id do símbolo, a data e o usuário para o servidor no formato JSON.
- u) UC21 – Enviar dados do usuário para o servidor. Este caso de uso estabelece que ao criar um usuário (UC01), o aplicativo deve enviar os dados: nome, e-mail, tipo de usuário, senha e foto para o servidor no formato JSON.
- v) UC22 – Enviar símbolo do usuário para o servidor. Este caso de uso estabelece que ao criar um símbolo (UC02), o aplicativo deve enviar os dados: nome, categoria, imagem e áudio para o servidor no formato JSON
- w) UC23 – Enviar plano para o servidor. Este caso de uso estabelece que ao criar um plano (UC03), o aplicativo deve enviar para o servidor os dados: nome, *layout*, símbolos e posição dos símbolos que compõem o plano. Os dados devem ser enviados no formato JSON.
- x) UC24 – Enviar observação do usuário para o servidor. Este caso de uso estabelece que ao criar uma observação (UC04), o aplicativo deve enviar para o servidor os dados: descrição, id do usuário, id do tutor e data de criação.
- y) UC25 – Visualizar planos. Este caso de uso permite visualizar um plano selecionado a partir da lista de planos (UC06). O aplicativo deverá exibir os símbolos do plano na mesma posição e *layout* do plano e permitir a interação com os símbolos (UC08).
- z) UC26 – Visualizar vídeos do YouTube. Este caso de uso permite visualizar o vídeo vinculado na propriedade *link para vídeo* do símbolo. Quando o usuário interagir com o símbolo (UC08) através de um *click* longo, o aplicativo deverá apresentar e reproduzir o vídeo associado.

Dos quadros 5 ao 30 a seguir, são apresentados os passos de execução e detalhamento dos fluxos de cada caso de uso.

Quadro 5 – Caso de uso Criar usuários

UC01 – Criar usuários	
Pré-condições	Conexão com a internet.
Cenário Principal	<p>1. O Usuário abre o aplicativo.</p> <p>2. O Usuário seleciona a opção Não quando questionado se já possui usuário na tela de boas-vindas.</p> <p>3. O aplicativo abre a janela para selecionar o tipo de usuário e então o Usuário seleciona um dos tipos apresentados.</p> <p>4. O aplicativo abre a janela de criação de usuário, então, o Usuário preenche os campos do cadastro.</p> <p>5. O Usuário pressiona botão Salvar para salvar as informações do usuário.</p>
Fluxo Alternativo	<p>No passo 2, caso o Usuário selecione a opção Sim quando questionado se já possui usuário na tela de boas-vindas:</p> <p>2.1. O aplicativo abre a janela de <i>login</i> e o Usuário seleciona a opção não tenho usuário.</p> <p>2.2. O aplicativo abre a janela para selecionar o tipo de usuário e então o Usuário seleciona um dos tipos apresentados.</p> <p>2.3. Retorna ao passo 3 do cenário principal</p>
Exceção	No passo 4 do cenário principal o Usuário não preenche os campos obrigatórios e tenta salvar, o aplicativo irá exibir uma mensagem alertando que os campos devem ser preenchidos.
Exceção 2	Caso usuário entre no aplicativo sem estar conectado à internet a seguinte mensagem é apresentada: “A conexão com a internet não foi encontrada a execução do Tagarela será limitada”.
Pós-condições	Informações do paciente salvas e o aplicativo apresenta a tela principal.

Quadro 6 – Caso de uso Criar símbolos

UC02 – Criar símbolos	
Pré-condições	Usuário conectado no aplicativo
Cenário Principal	<p>1. O Usuário seleciona a opção Criar símbolos na tela principal.</p> <p>2. O aplicativo abre a janela para seleção de categoria e o Usuário seleciona uma delas.</p> <p>3. O aplicativo abre a janela para criação de símbolos.</p> <p>4. O Usuário preenche os campos do cadastro.</p> <p>5. O Usuário pressiona o botão Salvar.</p>
Fluxo Alternativo	No passo 3 o Usuário pode cancelar ou voltar a ação realizada.
Exceção	No passo o 4 o Usuário não preenche o nome e tenta salvar o símbolo. Uma mensagem é exibida ao Usuário com o texto: Informações inválidas.
Pós-condições	Símbolo salvo na base local e no servidor.

Quadro 7 – Caso de uso Criar Planos

UC03 – Criar planos	
Pré-condições	1. Usuário conectado no aplicativo. 2. O Usuário precisa ter símbolos criados (UC03). 3. O Usuário precisa ter selecionado o <i>layout</i> do plano (UC05).
Cenário Principal	1. O aplicativo abre a tela de criação de planos com as figuras padrões, então, o Usuário toca uma das imagens. 2. O aplicativo abre uma janela para selecionar a categoria e o Usuário seleciona uma delas. 3. O aplicativo exibe uma janela com os símbolos da categoria selecionada, então, o Usuário seleciona um dos símbolos. 4. O aplicativo retorna a tela de criação e aplica o símbolo selecionado à imagem selecionada. 5. O Usuário repete os passos de 3 a 6 até que não tenha mais símbolos para inserir. 6. O Usuário pressiona o botão ADICIONAR PLANO. 7. O aplicativo abre a janela para inserir o nome do plano, então, o Usuário insere o nome. 8. O Usuário pressiona o botão Salvar.
Fluxo Alternativo	O usuário depois do passo 1 pode pressionar o botão voltar do dispositivo e então o aplicativo retorna para tela inicial.
Exceção	Não há.
Pós-condições	Plano salvo na base local e no servidor.

Quadro 8 – Caso de uso criar observação

UC04 – Criar observação	
Pré-condições	1. Usuário conectado no aplicativo.
Cenário Principal	1. O Usuário seleciona a opção Observações na janela principal. 2. A janela de observações é aberta. 3. O Usuário digita uma observação no TextEdit. 4. O Usuário pressiona o botão fechar.
Fluxo Alternativo	No passo 3 o usuário pressiona o botão fechar.
Exceção	Não há.
Pós-condições	Observação salva na base local e no servidor.

Quadro 9 – Caso de uso Selecionar o layout do plano

UC05 – Selecionar o layout do plano	
Pré-condições	2. Usuário conectado no aplicativo.
Cenário Principal	5. O Usuário seleciona a opção criar plano na janela principal. 6. O aplicativo abre a janela para seleção de layout e o Usuário seleciona um deles.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	Tela de criação de planos com o layout selecionado.

Quadro 10 – Caso de uso Visualizar lista de planos

UC06 – Visualizar lista de planos	
Pré-condições	1. O Usuário deve possuir planos.
Cenário Principal	1. O Usuário realiza <i>login</i> no aplicativo.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	Ao abrir a tela inicial uma lista com todos os planos do usuário é exibida no lado direito da tela.

Quadro 11 – Caso de uso Visualizar símbolos

UC07 – Visualizar símbolos	
Pré-condições	1. O Usuário conectado no aplicativo.
Cenário Principal	1. O Usuário seleciona a opção Visualizar símbolos na tela inicial.
Fluxo Alternativo	Não há.
Exceção	Se o usuário não possuir símbolos para exibir será aberta uma janela com a mensagem: “Não há itens para exibir.”
Pós-condições	O aplicativo abre uma janela com todos os símbolos criados pelo usuário.

Quadro 12 – Caso de uso Interagir com símbolos

UC08 – Interagir com símbolos	
Pré-condições	1. O Usuário deve ter um símbolo criado (UC02).
Cenário Principal	1. O Usuário executa o UC07. 2. O Usuário toca um dos símbolos do plano ou da tela de visualização.
Fluxo Alternativo	1. O Usuário executa o UC25. 2. O Usuário toca um dos símbolos do plano ou da tela de visualização.
Exceção	Não há.
Pós-condições	O UC10 é executado. Caso o usuário esteja visualizando um plano, o UC20 é executado.

Quadro 13 – Caso de uso Exibir borda em símbolos

UC09 – Exibir borda em símbolos	
Pré-condições	1. O Usuário deve ter um símbolo criado (UC02). 2. Executar qualquer ação que visualize símbolos.
Cenário Principal	1. Visualizar um símbolo.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	O símbolo é exibido com uma borda com a cor da categoria do símbolo.

Quadro 14 – Caso de uso Reproduzir som associado ao símbolo

UC10 – Reproduzir som associado ao símbolo	
Pré-condições	1. UC08 executado
Cenário Principal	1. O aplicativo reproduz o som associado ao símbolo.
Fluxo Alternativo	Não há.
Exceção	1. Se o símbolo não possuir som associado, nenhum som é reproduzido.
Pós-condições	Som associado ao símbolo é reproduzido.

Quadro 15 – Caso de uso Registrar histórico de símbolos utilizados

UC11 – Registrar histórico de símbolos utilizados	
Pré-condições	1. Usuário conectado no aplicativo. 2. O Usuário precisa ter símbolos criados UC02. 3. O Usuário precisa ter planos criados UC03.
Cenário Principal	1. Selecionar um símbolo quando um plano estiver sendo visualizado (UC25).
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	O histórico do símbolo utilizado é salvo na base local.

Quadro 16 – Caso de uso Visualizar observações

UC12 – Visualizar observações	
Pré-condições	1. UC04 executado.
Cenário Principal	1. O Usuário seleciona a opção Observações na tela principal. 2. A janela de observações é aberta.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	As observações do usuário são exibidas na janela.

Quadro 17 – Caso de uso Visualizar histórico de símbolos utilizados

UC13 – Visualizar histórico de símbolos utilizados	
Pré-condições	1. UC08 executado.
Cenário Principal	1. O Usuário seleciona a opção Histórico de símbolos na tela principal. 2. A janela de histórico de símbolos é aberta.
Fluxo Alternativo	Não há.
Exceção	Se o usuário não tiver interagido com nenhum símbolo através de um plano, uma janela com a seguinte mensagem é exibida: “Não há itens para exibir.”
Pós-condições	É apresentada para o usuário uma janela com os símbolos e a data de utilização.

Quadro 18 - Caso de uso Realizar login

UC14 – Realizar login	
Pré-condições	O Usuário já deve ter executado o UC01.
Cenário Principal	<ol style="list-style-type: none"> O Usuário abre o aplicativo. O Usuário seleciona a opção Sim na tela de boas-vindas. O aplicativo abre a janela de <i>login</i> e o Usuário informa um <i>id</i> válido. O Usuário pressiona o botão Confirmar.
Fluxo Alternativo	<p>Se o usuário não possuir conexão com a internet.</p> <ol style="list-style-type: none"> O Usuário abre o aplicativo É exibida uma janela com a mensagem: “A conexão com a internet não foi encontrada a execução do Tagarela será limitada” O Usuário pressiona o botão OK O aplicativo abre a janela de <i>login</i> e o Usuário informa um <i>id</i> válido. O Usuário pressiona o botão Confirmar.
Exceção	No passo 3 do cenário principal o Usuário informa um <i>id</i> inválido e pressiona o botão confirmar. O aplicativo exibe uma mensagem alertando o usuário que foi informado um <i>id</i> inválido.
Pós-condições	O aplicativo apresenta a tela principal.

Quadro 19 – Caso de uso Receber os planos do usuário armazenados no servidor

UC15 – Receber os planos do usuário armazenados no servidor	
Pré-condições	1. Não há
Cenário Principal	1. O Usuário realiza <i>login</i> no aplicativo conectado à internet.
Fluxo Alternativo	1. O Usuário realiza <i>login</i> no aplicativo sem conexão com à internet.
Exceção	Caso o aplicativo não possua conexão com a internet os planos não são sincronizados.
Pós-condições	Planos do usuário no servidor sincronizados na base local do dispositivo.

Quadro 20 – Caso de uso Receber os símbolos armazenados no servidor

UC16 – Receber os símbolos armazenados no servidor	
Pré-condições	1. Não há.
Cenário Principal	1. O Usuário realiza <i>login</i> no aplicativo conectado à internet.
Fluxo Alternativo	1. O Usuário realiza <i>login</i> no aplicativo sem conexão com à internet.
Exceção	Caso o aplicativo não possua conexão com a internet os símbolos não são sincronizados.
Pós-condições	Símbolos do servidor sincronizados na base local do dispositivo.

Quadro 21 – Caso de uso Receber dados do usuário armazenados no servidor

UC17 – Receber dados do usuário armazenados no servidor.	
Pré-condições	1. UC01 executado.
Cenário Principal	1. O aplicativo ao realizar o <i>login</i> do usuário busca as informações do usuário no servidor. 2. O aplicativo sincroniza as informações com as da base local.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	É apresentada a tela a principal com os dados do usuário.

Quadro 22 – Caso de uso Receber as observações do servidor

UC18 – Receber as observações do servidor	
Pré-condições	1. Não há.
Cenário Principal	1. O Usuário realiza <i>login</i> no aplicativo conectado à internet.
Fluxo Alternativo	1. O Usuário realiza <i>login</i> no aplicativo sem conexão com à internet.
Exceção	Caso o aplicativo não possua conexão com a internet as observações não são sincronizadas.
Pós-condições	Observações do servidor sincronizadas na base local do dispositivo.

Quadro 23 – Caso de uso Receber o histórico de símbolos

UC19 – Receber o histórico de símbolos do servidor	
Pré-condições	1. Não há.
Cenário Principal	1. O Usuário realiza <i>login</i> no aplicativo conectado à internet.
Fluxo Alternativo	1. O Usuário realiza <i>login</i> no aplicativo sem conexão com à internet.
Exceção	Caso o aplicativo não possua conexão com a internet os históricos não são sincronizados.
Pós-condições	Históricos de símbolos do servidor sincronizados na base local do dispositivo.

Quadro 24 – Caso de uso Enviar histórico de símbolo para o servidor

UC20 – Enviar histórico símbolo para o servidor	
Pré-condições	1. UC08 executado. 2. Aplicativo conectado à internet
Cenário Principal	1. O aplicativo envia as informações do plano para o servidor em formato JSON.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	Histórico do símbolo salvo no servidor.

Quadro 25 – Caso de uso Enviar dados do usuário para o servidor

UC21 – Enviar dados do usuário para o servidor	
Pré-condições	1. UC01 executado.
Cenário Principal	1. O aplicativo envia as informações do usuário para o servidor em formato JSON.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	Dados do usuário salvos no servidor.

Quadro 26 – Caso de uso Enviar símbolo para o servidor

UC22 – Enviar símbolo para o servidor	
Pré-condições	1. UC01 executado.
Cenário Principal	1. O aplicativo envia as informações do símbolo para o servidor em formato JSON.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	Símbolo salvo no servidor.

Quadro 27 – Caso de uso Enviar plano para o servidor

UC23 – Enviar plano para o servidor	
Pré-condições	2. UC03 executado.
Cenário Principal	2. O aplicativo envia as informações do plano para o servidor em formato JSON.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	Plano salvo no servidor.

Quadro 28 – Caso de uso Enviar observação para o servidor

UC24 – Enviar observação para o servidor	
Pré-condições	3. UC03 executado.
Cenário Principal	3. O aplicativo envia as informações da observação para o servidor em formato JSON.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	Observação salva no servidor.

Quadro 29 – Caso de uso Visualizar planos

UC25 – Visualizar planos	
Pré-condições	1. Usuário conectado no aplicativo. 2. O Usuário deve possuir planos.
Cenário Principal	1. O Usuário seleciona um plano na lista de planos.
Fluxo Alternativo	Não há.
Exceção	Não há.
Pós-condições	O plano é exibido conforme criado no UC03.

Quadro 30 – Caso de uso Visualizar vídeos do youtube

UC26 – Visualizar vídeos do YouTube	
Pré-condições	<ol style="list-style-type: none"> 1. Usuário conectado no aplicativo com internet 2. Usuário estar visualizando um plano com símbolo que possui link para o youtube.
Cenário Principal	<ol style="list-style-type: none"> 1. O Usuário faz um toque longo em um símbolo com um vídeo vinculado.
Fluxo Alternativo	Com o dispositivo sem internet <ol style="list-style-type: none"> 1. Usuário faz um toque longo em um símbolo com um vídeo vinculado.
Exceção	Com internet de baixa capacidade ou restrição de proxy o aplicativo abre o visualizador, porém, é exibida a mensagem: <i>Ocorreu um erro</i>
Pós-condições	É aberta uma tela para a visualização do vídeo.

APÊNDICE B – Formulários sobre experiência de utilização do Tagarela

Nas figuras 55 a 60 a seguir, são apresentados os formulários preenchidos pelas fonoaudiólogas do CEMEA com suas considerações sobre a utilização do Tagarela.

Figura 55 – Formulário preenchido pela fonoaudióloga Luana Viola da Cruz

 FURB – Universidade Regional de Blumenau Curso de Ciência da Computação – Bacharelado Projeto Tagarela – gcg.inf.furb.br/tagarela TCC – Tagarela: Aplicativo de comunicação alternativa na plataforma Android
Questionário de Avaliação
1. Nome: <u>Luana Viola da Cruz</u> CFA - 36036.
2. Considerando uma plataforma de comunicação alternativa. O aplicativo Tagarela para Android atende as suas necessidades? <input checked="" type="checkbox"/> Sim <input type="checkbox"/> Parcialmente <input type="checkbox"/> Não
3. A utilização do som associado ao símbolo ajudou no aprendizado do usuário? <input type="checkbox"/> Não ajudou <input type="checkbox"/> Ajudou pouco <input type="checkbox"/> Ajudou <input checked="" type="checkbox"/> Ajudou muito
4. Como você avalia a usabilidade (facilidade de uso) do aplicativo? <input type="checkbox"/> Ruim <input type="checkbox"/> Regular <input type="checkbox"/> Boa <input checked="" type="checkbox"/> Ótima
5. Com base nas respostas anteriores, quais foram os principais "pontos positivos" encontrados durante o uso do aplicativo? * Permite uso de fotos compartilhadas pela família. * Permite a criação de vários pacotes. * Dinâmico.
6. Com base nas respostas anteriores, quais foram os principais "pontos negativos" encontrados durante o uso do aplicativo? * Nenhum.

Figura 56 – Formulário preenchido pela fonoaudióloga Luana Viola da Cruz (continuação)

7. Com base nas respostas anteriores, quais seriam as suas "sugestões" para melhorar o aplicativo? * Uso de internet.
--

Figura 57 – Formulário preenchido pela fonoaudióloga Ana Maria Philipps dos Santos

 FURB – Universidade Regional de Blumenau
 Curso de Ciência da Computação – Bacharelado
 Projeto Tagarela – gcp.inf.furb.br/tagarela
 TCC – Tagarela: Aplicativo de comunicação alternativa na
 plataforma Android

Questionário de Avaliação

- Nome: Ana Maria Philippo dos Santos / CEFET 2020
- Considerando uma plataforma de comunicação alternativa. O aplicativo Tagarela para Android atende as suas necessidades?
 - Sim
 - Parcialmente
 - Não
- A utilização do som associado ao símbolo ajudou no aprendizado do usuário?
 - Não ajudou
 - Ajudou pouco
 - Ajudou
 - Ajudou muito
- Como você avalia a usabilidade (facilidade de uso) do aplicativo
 - Ruim
 - Regular
 - Boa
 - Ótima
- Com base nas respostas anteriores, quais foram os principais "pontos positivos" encontrados durante o uso do aplicativo?

ESTIMULAR LÍNGUA E MUDAR HABITOS
ESTIMULAR VÍNCULO
- Com base nas respostas anteriores, quais foram os principais "pontos negativos" encontrados durante o uso do aplicativo?

NÃO TEM!

Figura 58 – Formulário preenchido pela fonoaudióloga Ana Maria Philipp dos Santos (continuação)

7. Com base nas respostas anteriores, quais seriam as suas "sugestões" para melhorar o aplicativo?

ENCADAR COMUNICAÇÃO
ORGANIZAR MELHOR OS PLANOS

Figura 59 – Formulário preenchido pela fonoaudióloga Binca Kleis de Carvalho

 FURB – Universidade Regional de Blumenau
Curso de Ciência da Computação – Bacharelado
Projeto Tagarela – gcg.inf.furb.br/tagarela
TCC – Tagarela: Aplicativo de comunicação alternativa na plataforma Android

Questionário de Avaliação

1. Nome: Binca Kleis de Carvalho CRF 310.306
2. Considerando uma plataforma de comunicação alternativa. O aplicativo Tagarela para Android atende as suas necessidades?
 - Sim
 - Parcialmente
 - Não
3. A utilização do som associado ao símbolo ajudou no aprendizado do usuário?
 - Não ajudou
 - Ajudou pouco
 - Ajudou
 - Ajudou muito
4. Como você avalia a usabilidade (facilidade de uso) do aplicativo
 - Ruim
 - Regular
 - Boa
 - Ótima
5. Com base nas respostas anteriores, quais foram os principais "pontos positivos" encontrados durante o uso do aplicativo?
 - Utilização de imagens reais.
 - Apelo sensorial relacionado a imagem.
 - Possibilidade de visualizar históricos de símbolos.
6. Com base nas respostas anteriores, quais foram os principais "pontos negativos" encontrados durante o uso do aplicativo?
 - Dificuldade por parte das crianças em mudar de planos, uma vez que utilizam linguagem escrita para identificar planos, e não símbolos.

Figura 60 – Formulário preenchido pela fonoaudióloga Binca Kleis de Carvalho (continuação)

7. Com base nas respostas anteriores, quais seriam as suas "sugestões" para melhorar o aplicativo?
 - Utilizar símbolos para identificar os planos, assim como permitir a trocação de um plano para o outro.

APÊNDICE C – Caderno de comunicação

As figuras 61, 62 e 63 apresentam o caderno de comunicação utilizado por um aluno do CEMEA.

Figura 61 – Caderno de comunicação



Figura 62 – Caderno de comunicação (continuação)

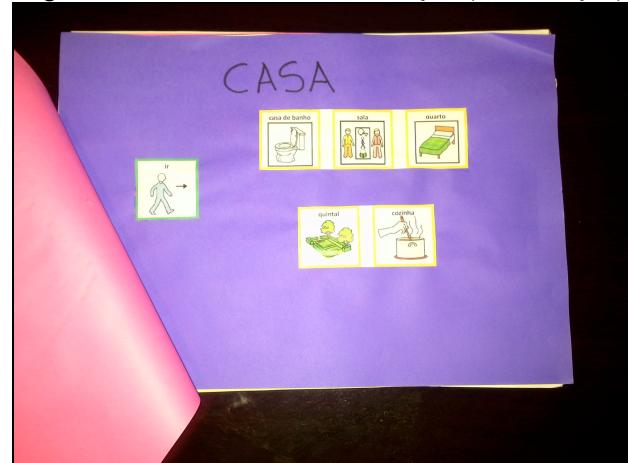


Figura 63 – Caderno de comunicação (continuação)



APÊNDICE D – Utilização do Tagarela por usuário com autismo

A Figura 64 apresenta uma criança com autismo utilizando o Tagarela.

Figura 64 – Criança com autismo utilizando o Tagarela

