UNIVERSIDADE FEDERAL DO MARANHÃO

CURSO DE CIÊNCIA DA COMPUTAÇÃO

EIDER MATHEUS COSTA DINIZ

ANÁLISE E PROJETO DE UM SISTEMA DE DIÁLOGO PARA UMA APLICAÇÃO MÓVEL VOLTADA AO DOMÍNIO DA SAÚDE

São Luís

2013

EIDER MATHEUS COSTA DINIZ

ANÁLISE E PROJETO DE UM SISTEMA DE DIÁLOGO PARA UMA APLICAÇÃO MÓVEL VOLTADA AO DOMÍNIO DA SAÚDE

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Francisco José da Silva e Silva

São Luís

2013

Diniz, Eider Matheus Costa.

Análise e projeto de um sistema de diálogo para uma aplicação móvel voltada ao domínio da saúde / Eider Matheus Costa Diniz. – São Luís, 2013.

75 f.

Impresso por computador (fotocópia).

Orientador: Francisco José Silva e Silva.

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2013.

1. Computação móvel – interação humano computador. 2. Redes Sociais. 3. Saúde. I. Título.

CDU 004.5

EIDER MATHEUS COSTA DINIZ

ANÁLISE E PROJETO DE UM SISTEMA DE DIÁLOGO PARA UMA APLICAÇÃO MÓVEL VOLTADA AO DOMÍNIO DA SAÚDE

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Francisco José da Silva e Silva

Aprovada em 15 de Outubro de 2013

BANCA EXAMINADORA:

Prof. Dr. Francisco José da Silva e Silva (orientador)

Universidade Federal do Maranhão

Prof. Dr. José de Ribamar Braga Pinheiro Júnior

Universidade Federal do Maranhão

Prof. Me. Carlos Eduardo Portela Serra de Castro

Universidade Federal do Maranhão

Resumo

Uma rede social móvel pode ser definida como uma estrutura social cujos membros se relacionam em grupos e interagem através de tecnologias de informação e comunicação com dispositivos portáteis de computação que dispõem de acesso a tecnologias sem fio. Na área da saúde, redes sociais podem ser utilizadas com a finalidade de promover o intercâmbio de informações entre pacientes e profissionais, conduzindo coletivamente ações relacionadas à assistência e educação médicas.

Este trabalho apresenta uma proposta de interação humano-computador baseada em sistemas de diálogo falado para uma aplicação móvel voltada ao domínio da saúde, que visa o acompanhamento à distância do estado de saúde de pacientes pertencentes a comunidades carentes que apresentam limitações de interação com interfaces textuais. A aplicação está inserida no contexto do projeto MobileHealthNet, desenvolvido em parceria pelo Laboratório de Sistemas Distribuídos da Universidade Federal do Maranhão e o *Laboratory for Advanced Collaboration* da Pontifícia Universidade Católica do Rio de Janeiro, que tem por objetivo desenvolver um *middleware* que viabilize a implantação de redes sociais móveis e simplifique o desenvolvimento de aplicações colaborativas na área da saúde. Ao longo do texto estão descritos conceitos relacionados a redes sociais, computação móvel, sistemas de diálogo e uma contextualização do projeto, bem como a análise e projeto da interação humano-computador proposta para a aplicação.

Palavras-chave: computação móvel, sistemas de diálogo, interação humano-computador, redes sociais, saúde.

Abstract

A mobile social network can be defined as a sócial structure whose members relate and interact in groups through information and communication technologies with portable computing devices that have access to wireless technologies.

This work presents a propose of human-computer interaction based on spoken dialogue systems for a mobile application focused to the health domain, which aims at distance monitoring of the health status of patients belonging to poor communities that have limitations of interaction with textual interfaces. The application is inserted in the context of the MobileHealthNet project, developed in partnership by the Distributed Systems Laboratory at the Universidade Federal do Maranhão and the Laboratory for Advanced Collaboration of the Pontifícia Universidade Católica do Rio de Janeiro which aims develop a middleware that makes possible for the deployment of mobile social networks and simplify the development of collaborative applications in the health domain. Throughout the text are described concepts related to social networks, mobile computing, dialogue systems and a contextualization of the project, as well as the analysis and design of the human-computer interaction proposed for the application

Keywords: mobile computing, dialogue systems, human-computer interaction, social networking, health.

Lista de Figuras

Sumário

1 INTRODUÇÃO

O crescimento populacional é um assunto que periodicamente retorna à pauta de discussões. Entretanto, paralelamente à mudança positiva da população mundial ao longo dos anos, o aumento significativo do uso de dispositivos móveis, sobretudo de *smartphones*, vem requerendo uma atenção especial.

Recentemente, diversas pesquisas realizadas por empresas, companhias e consultorias especializadas no mercado de tecnologia e telecomunicações refletem o avanço da tecnologia móvel**.** A Nielsen, por exemplo, revelou que trinta e seis por cento dos entrevistados com acesso à Internet ao redor do mundo reportaram possuir um *smartphone* [1]. Esses dados são corroborados pela Ericsson*,* que afirma estar entre quinze e vinte por cento a representação de *smartphones* dentre os telefones celulares em uso atualmente no mundo. A companhia ainda afirma que o número de usuários com *smartphone* era de pouco mais de um bilhão no fim de 2012 e, segundo cálculos realizados em seu mais recente relatório anual, ultrapassará a marca de três bilhões até o fim de 2018 [2]. Somente no Brasil, segundo projeção divulgada pela IDC, foram vendidas mais de quinze milhões de unidades destes aparelhos em 2012 [3]. Além dos *smartphones*, os pontos de acesso móveis também aumentam gradativamente. Até 2014, o Brasil deve ter algo em torno de cento e vinte quatro milhões de pontos de acesso à banda larga móvel, segundo revela pesquisa da Huawei em parceria com a Teleco [4]. Este cenário impulsiona cada vez mais o desenvolvimento de aplicações móveis.

O novo referencial de conteúdo gerado pelo usuário proporcionou uma nova abordagem aos meios de comunicação utilizados para interação social. Aliado aos benefícios proporcionados com o advento da chamada Web 2.0, como compartilhamento de informações midiáticas (texto, áudio, vídeo, imagem, animação, etc.) entre usuários de forma mais interativa, tornou-se determinante no surgimento das redes sociais, responsáveis pela criação de um termo amplamente difundido nos dias atuais conhecido como mídias sociais. Assim, uma definição plausível para o termo é a de um grupo de aplicações para a Internet construída com base nos fundamentos da Web 2.0 que permitem a criação e a troca de conteúdo gerado pelo usuário [5]. Em outras palavras, os usuários interagem e colaboram

entre si assumindo o papel de criadores de conteúdos dentro da comunidade virtual estabelecida a partir da mídia social.

Em alta no atual cenário do mundo virtual, a rede social é um dos tipos de mídias sociais de maior popularidade. A ideia principal consiste em uma estrutura social onde os membros se relacionam em grupos [6] de forma instantânea, o que possibilita interação entre pessoas distantes entre si através de tecnologias da informação e comunicação. Assim, experiências que vão de simples contatos informais entre amigos e familiares a compartilhamento de estratégias de atuação profissionais ultrapassam as barreiras geográficas e temporais. Por isso, uma rede social pode ser considerada como um meio de comunicação em uma estrutura social definida [7].

Com o intuito de incorporar mobilidade às redes sociais surgiram as chamadas redes sociais móveis, isto é, redes sociais manipuladas por dispositivos portáteis de computação com acesso a tecnologias sem fio. Neste cenário, além de agregar a capacidade de acesso a qualquer hora e em qualquer lugar às redes sociais [8], diversas informações de contexto podem ser exploradas direta ou indiretamente, tais como localização, disponibilidade e ações do usuário, clima e temperatura locais, entre outras, obtidas através de dispositivos usualmente contidos em *smartphones* como câmera, receptor GPS e sensores como giroscópio e acelerômetro.

Uma área fértil para a aplicação dos conceitos relacionados à rede social é a saúde. O termo *Mobile Health* caracteriza a prática de medicina e atendimento de saúde através de dispositivos móveis [9]. Neste contexto, pode ser definida como um grupo de pessoas, em sua estrutura social construída coletivamente, que utiliza tecnologias da informação e comunicação com o propósito de conduzir coletivamente ações relacionadas à assistência médica e à educação [10]. Em outras palavras, as redes sociais móveis podem ser utilizadas para promover o intercâmbio de informações, a colaboração e a integração social entre os diversos agentes envolvidos no processo de atendimento à saúde [11]. O uso de dispositivos móveis permite o contato entre pacientes e profissionais de saúde independente de suas localizações físicas, além do acompanhamento constante de informações referentes ao estado de saúde de cada paciente. Assim, estabelece-se um canal de

comunicação paralelo ao encontro presencial exigido pela consulta, tão rotineiras ao longo de um tratamento e que pode se tornar um agravante para pacientes de comunidades carentes que enfrentam dificuldades de acesso ao hospital, seja por residirem longe ou por não possuírem meios financeiros suficientes para o deslocamento. Vale ressaltar a utilização de telefones celulares como meio tecnológico mais acessível a comunidades carentes e distantes, tendo em vista os recursos, funcionalidades e constantes facilidades de uso e de aquisição proporcionados por estes dispositivos. Diante deste contexto, a utilização de dispositivos móveis se mostra uma alternativa plausível ao desenvolvimento de aplicações no âmbito da saúde.

As aplicações para redes sociais móveis voltadas ao domínio da saúde consistem em manter uma interação entre profissionais da saúde e pacientes, podendo ser utilizadas com o objetivo de promover o acompanhamento de tratamentos. Estas aplicações são importantes no sentido de ajudar a evitar que suas condições se agravem, ou que corram risco de morte, através do simples monitoramento regular do estado de suas doenças e de uma possível prevenção de crises [12]. Entretanto, especialmente no caso de pacientes pertencentes a comunidades carentes, a interação através de interfaces textuais, predominantemente utilizadas em aplicações móveis, pode se tornar um obstáculo a mais em seu acompanhamento. As próprias doenças, que afetam diretamente os membros superiores, por exemplo, podem constituir um empecilho para este tipo de interação. Sistemas de diálogo, ou agentes conversacionais, são sistemas de computador capazes de conversar com uma pessoa através de uma estrutura coerente tanto para o canal de entrada como para o de saída, por meio de textos, gráficos, gestos e outros meios de comunicação, inclusive o que as pessoas se habituaram a praticar em quase todos os momentos da vida: a habilidade de falar e entender conversas [13]. Sendo assim, os sistemas de diálogo falado compreendem uma alternativa eficaz ao desenvolvimento de interfaces para sistemas móveis. A ideia é proporcionar uma interação simultaneamente simples, fluida e robusta, isto é, estabelecer uma comunicação rápida entre usuário e sistema sem necessidade de prévio aprendizado.

1.1 Projeto MobileHealthNet

Em busca de viabilizar a interação entre pessoas que fazem uso de diferentes dispositivos e tecnologias, várias infraestruturas conhecidas como *middlewares* são desenvolvidas para agilizar a implementação destas aplicações. Este trabalho está inserido no contexto do MobileHealthNet [14], um projeto desenvolvido em parceria pelo Laboratório de Sistemas Distribuídos da UFMA e o *Laboratory for Advanced Collaboration* da PUC-Rio, tendo por objetivo geral avançar o estado da arte em sistemas de *middleware* para redes sociais móveis, utilizando uma infraestrutura de software para a criação de novos serviços e aplicações de redes sociais móveis voltados para a área da saúde. Este projeto conta com o apoio institucional do Hospital Universitário da UFMA (HUUFMA). Em particular, duas unidades do HUUFMA estão diretamente envolvidas com o desenvolvimento do projeto: o Programa de Assistência a Pacientes Asmáticos (PAPA) e a Casa da Dor, esta última especializada no tratamento de pacientes que sofrem com dores crônicas, independentemente de sua etiologia.

O MobileHealthNet tem por objetivo o desenvolvimento de um sistema de *middleware* que permita a construção de redes sociais móveis e facilite o desenvolvimento de serviços colaborativos para o setor da saúde, a troca de experiências e a comunicação entre pacientes e profissionais de saúde, além de uma melhor gestão dos recursos da saúde por órgãos governamentais. Este projeto foi concebido visando ser uma aplicação especial a comunidades carentes e distantes. Neste contexto, o termo “distante” refere-se não somente à distância física entre a comunidade e os centros de atendimento de saúde, mas também à dificuldade em realizar este deslocamento devido a precariedade do acesso ou indisponibilidade de meios de transportes adequados.

1.2 Objetivos

Os objetivos gerais deste trabalho são o estudo dos principais conceitos relacionados ao desenvolvimento de sistemas de diálogo falado para ambientes

computacionais móveis e a implementação de um estudo de caso deste tipo de sistema voltado ao domínio da saúde, cuja finalidade é manter uma interação entre profissionais de saúde e pacientes.

Os objetivos específicos compreendem:

- O estudo dos conceitos gerais relativos às redes sociais móveis e suas aplicações na área da saúde, conceitos de interação humano-computador para ambientes computacionais móveis e sistemas de diálogo;

- A prospecção e estudo de linguagens e ferramentas voltadas ao desenvolvimento de sistemas de diálogo falado;

- Análise e projeto de uma interação humano-computador baseada em diálogo falado.

1.3 Organização do trabalho

O conteúdo deste documento está organizado da seguinte forma:

- O capítulo 2 (dois) enfoca os fundamentos de redes sociais móveis, a computação móvel e suas principais características, os sistemas de diálogo e suas aplicações e a arquitetura do *middleware* MobileHealthNet, bem como as funcionalidades disponibilizadas em cada serviço, com ênfase na aplicação colaborativa para o acompanhamento à distância do estado de saúde de pacientes;

- O capítulo 3 (três) apresenta a arquitetura do sistema operacional Android e algumas noções básicas de implementação utilizando o Google Android SDK e o Dragon Mobile SDK, responsáveis por possibilitar o desenvolvimento de

aplicações na plataforma Android e de sistemas de diálogo falado, respectivamente;

- O capítulo 4 (quatro) aborda os aspectos de análise e projeto da interação humano-computador proposta para a aplicação colaborativa, seu desenvolvimento na plataforma Android e sua integração com o *middleware* MobileHealthNet.

- O capítulo 5 (cinco) encerra o trabalho relatando as considerações finais e apresentando sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A análise e o projeto de uma interação humano-computador baseada em sistemas de diálogo falado para a aplicação móvel voltada ao domínio da saúde pressupõe conhecimentos consistentes acerca de redes sociais móveis, sistemas de diálogo e do *middleware* MobileHealthNet, utilizado pela aplicação. Neste capítulo serão abordadas as fundamentações destes conhecimentos necessárias ao entendimento do trabalho.

2.1 Redes sociais móveis

As redes sociais móveis resultam da união de três áreas, a saber, redes sociais, computação móvel e sistemas de ciência de contexto [15]. O ambiente de uma rede social proporciona aos seus membros relacionamentos decorrentes de interações virtuais independentemente do lugar onde estejam, estabelecendo um novo canal de comunicação além da estritamente pessoal. Por sua vez, a computação móvel permite que esta comunicação seja estabelecida a qualquer hora e em qualquer lugar através devido ao suporte de mobilidade provido por dispositivos portáteis com acesso à conectividade sem fio. Para completar, o intercâmbio de informações de contexto é viabilizado através dos sistemas de ciência de contexto. Assim, é possível obter do usuário informações como localização geográfica, temperatura, entre outras. A figura a seguir ilustra a representação das redes sociais móveis.

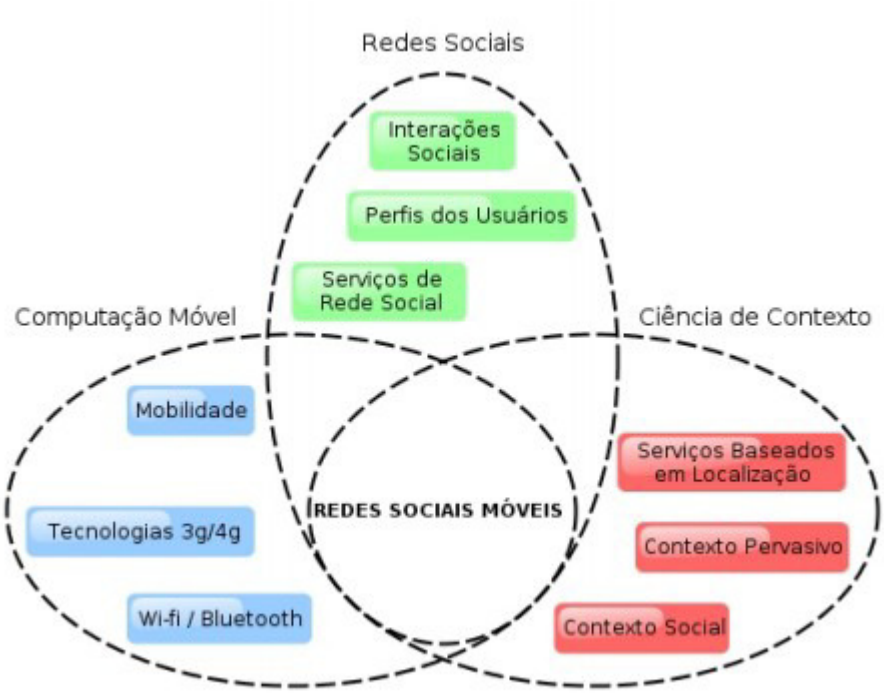

Figura 2.1 – Representação das redes sociais móveis

Fonte: TELES et al. [16]

De acordo com a figura é possível notar que as redes sociais móveis podem ser vistas como uma combinação das três áreas brevemente descritas anteriormente. O resultado é a agregação dos benefícios proporcionados por cada uma delas de forma simultânea, provendo funcionalidades para criar perfis que representam entidades, as quais podem relacionar-se socialmente trocando informações.

O uso de redes sociais móveis foi intensificado a partir do surgimento da Web 2.0 e torno dela surgiram pesquisas que resultaram em grandes avanços. A rede passou a ser orientada a oferecer suporte à informações de métricas referentes ao organismo do usuário, tais como batimentos cardíacos, pressão sanguínea e temperatura, que podem auxiliar profissionais de saúde no acompanhamento do estado de saúde de pacientes que são submetidos a tratamentos específicos, além do compartilhamento destas informações entre os demais profissionais envolvidos, favorecendo uma avaliação coletiva dos dados obtidos. Assim, redes sociais na área da saúde possibilitam uma combinação dos vários agentes envolvidos no processo de atenção à saúde, incluindo profissionais da saúde (médicos, enfermeiros,

fisioterapeutas, terapeutas ocupacionais, entre outros), pesquisadores da saúde (professores e alunos de graduação e de pós-graduação), pacientes e seus familiares, assim como membros da comunidade em geral.

São inúmeras as aplicações de redes sociais entre estes agentes. Entre pesquisadores de saúde, por exemplo, é possível o compartilhamento informações e posicionamento sobre pesquisas e estudos acadêmicos realizados colaborativamente sobre os mais variados temas. Os pacientes e seus familiares, por suas vezes, podem interagir com outros familiares que estejam sob o mesmo diagnóstico médico promovendo a troca de experiências e opiniões a respeito do tratamento específico. Aos profissionais de saúde é favorecido o intercâmbio de informações e melhorias na coordenação das atividades atribuídas aos integrantes da equipe. Por fim, entre profissionais de saúde e pacientes são viabilizadas aplicações que utilizam, por exemplo, sensores de dispositivos móveis que permitem o monitoramento do estado de saúde do paciente. Estes dados podem ser enviados aos profissionais de saúde que, por suas vezes, podem orientar os pacientes de acordo com o que for coletado.

A motivação do uso de redes sociais móveis a saúde surgiu da constante necessidade de locomoção dos profissionais da área entre vários pontos de um ou mais hospitais, como é o caso dos agentes de saúde responsáveis pela atenção básica do Programa Saúde da Família [17], e da utilidade dos dispositivos móveis em estabelecer comunicação independente de localização física ou temporal e coletar dados referente ao corpo humano através de sensores móveis que permitem o monitoramento contínuo de informações como taxa de glicemia, pressão arterial, entre outras, disponibilizadas em tempo real.

2.2 Sistemas de diálogo

A popularização dos dispositivos móveis como *smartphones* e *tablets* devido às constantes facilidades de aquisição proporcionou o surgimento de inúmeras aplicações com as mais variadas finalidades e formas de interação. Assim, saber qual a lanchonete mais próxima, a quantidade de calorias de um sanduíche e até

mesmo o nome de seu idealizador são questões que podem ser solucionadas imediatamente através de alguns poucos toques na tela do dispositivo móvel ou mesmo utilizando a própria voz como em uma conversa comum entre duas pessoas. Esta última interação é proporcionada através dos chamados “sistemas de diálogo falado” (do inglês, s*poken dialog system*) que são sistemas computacionais compostos por microfones e alto-falantes como canais de entrada e saída, respectivamente, que geram uma “conversa” com o usuário em linguagem natural. Através da ação conjunta de seus componentes, o sistema é capaz de entender e prover o que o usuário está requerendo apenas processando o áudio proveniente de sua voz.

Em virtude da praticidade, muitas aplicações móveis já se mobilizam no intuito de implantar sistemas de diálogo falado como forma de interação com o usuário. Além de ser informado sobre determinada refeição, chegar a um determinado destino sem conhecer o caminho, por exemplo, tornou-se uma tarefa intuitiva com a chegada dos guias móveis com suporte a comandos de voz, sendo particularmente bastante útil uma vez que a atenção de um condutor deve estar totalmente focada na estrada e desvios dela periodicamente para um mapa, por exemplo, pode ser perigoso em alguns momentos do trajeto. Apesar da tendência, desenvolver sistemas de diálogo falado não é uma tarefa trivial, tendo em vista precisão requerida no reconhecimento da fala, primordial em sistemas baseados em comandos de voz, por exemplo. Variações históricas, linguísticas e socioculturais são apenas alguns dos muitos fatores que impõem limitações a este tipo de sistema, pois cada pessoa fala de um jeito.

Um sistema de diálogo normalmente divide o processamento da voz em funcionalidades sequenciais justamente devido à complexidade que este tipo de interação envolve. O gerenciamento do diálogo também deve ser pensado considerando a aplicação a ser desenvolvida. A seguir, estão descritos os principais componentes de um sistema de diálogo, bem como os seus possíveis tipos de controle.

2.2.1 Principais componentes de um sistema de diálogo falado

Um sistema de diálogo falado é composto por módulos seriais especificamente projetados para realização de determinada função, isto é, cada componente é responsável por executar uma tarefa e transferir seu resultado de saída para a entrada do módulo seguinte, formando um ciclo conforme mostra a arquitetura a seguir.

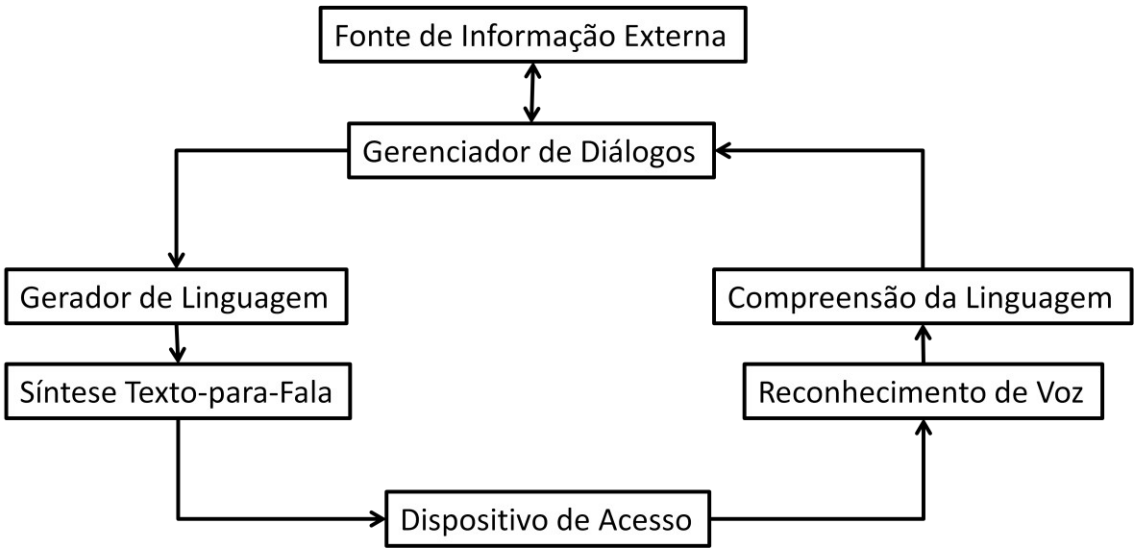

Figura 2.2 – Arquitetura de um sistema de diálogo falado

Fonte: M. F. McTear; T. V. Ramam [13]

De acordo com esta arquitetura básica, nota-se que a voz do usuário consiste na entrada do sistema sendo capturada por um dispositivo de acesso, como um *smartphone*, *tablet* ou outro dispositivo móvel, através de um microfone embutido. O áudio então é repassado ao módulo de reconhecimento de voz (do inglês, *Speech Recognition*), responsável por reconhecer a fala do usuário através da conversão do áudio recebido em uma determinada sequencia de palavras que só terão atribuição de significados sintáticos e semânticos quando encaminhadas ao módulo de compreensão da linguagem (do inglês, *Language Understanding*). Com os significados atribuídos, a função do gerenciamento de diálogo (do inglês, *Dialog Manager*) é gerar uma resposta ao usuário com base na análise do contexto dos

significados das palavras recebidas e da consulta a uma fonte de Informação externa, isto é, dados solicitados ao usuário para o caso da resposta gerada se tratar de uma confirmação de entendimento ou uma pergunta adicional para esclarecimento do mesmo. Este processo de verificação é conhecido como *Grounding*. De posse deste entendimento, o módulo gerador de linguagem (do inglês, *Language Generation*) é responsável por selecionar palavras para construir a resposta ao usuário. Finamente, o módulo de síntese texto-para-fala (do inglês, *Text-to-Speech Synthesis*) é responsável por transformar em áudio a resposta gerada ao usuário através de um alto-falante.

Por ser uma arquitetura dividida em módulos sequenciais e com funcionalidades específicas é intuitivo notar que o desempenho de cada componente é fator determinante para o resultado dos módulos seguintes. Assim, se a voz do usuário não for bem capturada pelo dispositivo móvel, por exemplo, o trabalho do módulo de reconhecimento de voz estará comprometido, assim como a função do módulo de compreensão da linguagem e dos demais componentes, prejudicando o resultado que o usuário espera do diálogo. Nem todos os sistemas de diálogo seguem à risca esta arquitetura, podendo ser constituídos de módulos mesclados ou até mesmo desconsiderados.

2.2.2 Tipos de controle de diálogo

Entre as funções atribuídas ao módulo de gerenciamento de diálogo está o tipo de controle do seu fluxo. Um deles é o baseado em estados, que tem como característica a simplicidade por possuir um fluxo de perguntas e respostas pré-determinado. A estrutura deste tipo de controle consiste em uma grafo, onde os nós representam as perguntas e a transição (aresta) representam as possibilidades de resposta, cada uma indicando um possível caminho para o diálogo. A figura a seguir demonstra um exemplo deste tipo de controle.

Figura 2.3 – Estrutura de diálogo de um sistema finito baseado em estados

Fonte: M. F. McTear; T. V. Ramam [13]

Neste exemplo, o sistema pergunta para o usuário qual o destino de sua viagem e depois analisa a resposta recebida. Se o usuário não confirmar a verificação do destino, a pergunta é refeita e a resposta analisada novamente; caso contrário, o sistema segue para a próxima pergunta seguida de uma nova análise de resposta. Devido à sua limitação este tipo de controle normalmente está associado a aplicações comerciais.

Outro tipo de controle do diálogo é o baseado em *frames*. Ele difere do controle baseado em estados por não precisar seguir uma sequencia e pela possibilidade de captura de mais de uma informação por vez. Por exemplo, a pergunta “Qual o destino da viagem?” feita pelo sistema é respondida pelo usuário da seguinte forma: “Eu quero ir para Recife no domingo”. A resposta contém não apenas o destino, mas o dia e o horário desejados pelo usuário. Esta flexibilidade é proporcionada pelos sistemas baseados em *frames* através dos seguintes componentes: o *frame* (modelo) que registra o que o sistema precisa do usuário, uma gramática de reconhecimento extensa e um algoritmo de controle de diálogo determinante da ação a ser tomada pelo sistema [13]. Através dos espaços (*slots*)

que compõem o *frame* é possível ao sistema saber se todos os itens foram preenchidos pelo usuário ou não. A gramática seria responsável por analisar a estrutura da resposta. No exemplo acima, assumindo que os itens são destino, dia e horário, a gramática saberia que a estrutura da resposta contém os itens destino e dia, podendo indicar que o próximo passo do sistema seria perguntar “Qual o horário?” para preencher o único item não preenchido pelo usuário.

Finalmente, existe o controle de diálogo baseado em agentes. Trata-se de um método que utiliza técnicas de Inteligência Artificial com o fim de prever a próxima possível pergunta do usuário. Um exemplo é a pergunta “Há voo para Recife?” feita pelo usuário para um sistema de determinada companhia aérea e recebendo a seguinte resposta: “Não, mas há voos para São Luís e Fortaleza. Você quer viajar para algum destes lugares?”. Este tipo de sistema é bem mais complexo, pois não há como prever os dados solicitados ou requeridos pelo usuário.

2.3 *Middleware* MobileHealthNet

As aplicações de software propostas pelo *middleware* MobileHealthNet [14] são baseados em um conjunto de requisitos gerados a partir de reuniões periódicas com os profissionais da saúde envolvidos no projeto. Essas reuniões têm como objetivo o conhecimento do domínio e das principais necessidades de ambos os núcleos (PAPA e Casa da Dor). Através de técnicas de extração de requisitos, como entrevistas e *brainstorms*, em conjunto com os profissionais de saúde, é realizado o processo de obtenção dos principais requisitos funcionais e não funcionais das aplicações.

A infraestrutura de *software* baseia-se nos princípios dos chamados Métodos Ágeis de Desenvolvimento de *Software*, em conjunto com o CLASP (*Comprehensive*, *Lightweight Application Security Process*). Aliadas, estas metodologias possibilitam a fabricação de software de forma incremental, iterativa, adaptativa e segura.

Ao final do processo de desenvolvimento realiza-se a verificação e validação do *software*, para certificação de que este atende os requisitos especificados e que

esteja de acordo com as expectativas do cliente. Um importante aspecto levado em consideração no contexto do projeto é precisamente a utilização de interações humano-computador apropriadas que se adaptem aos perfis dos usuários do projeto, considerados bastante heterogêneos, visto que alguns possuem uma alta familiaridade com sistemas computacionais e outros nunca tiveram contato com esses dispositivos.

Entre os benefícios esperados com o MobileHealthNet estão um melhor fluxo de informação e maior colaboração entre profissionais dos diversos níveis de atendimento à saúde; melhorias nos níveis de comprometimento e informação do paciente, o que contribui com o processo terapêutico [18]; melhoria na qualidade da tomada de decisão relativa ao tratamento por parte dos pacientes; melhor gestão de pacientes portadores de doenças crônicas, melhor suporte emocional aos pacientes e redução de custos do sistema de atendimento à saúde, visando a diminuição da necessidade de deslocamentos e a ocorrência de complicações ao longo do tratamento de pacientes.

A arquitetura utilizada como base para o desenvolvimento dos protótipos de aplicações é organizada em camadas, conforme ilustra a figura a seguir.

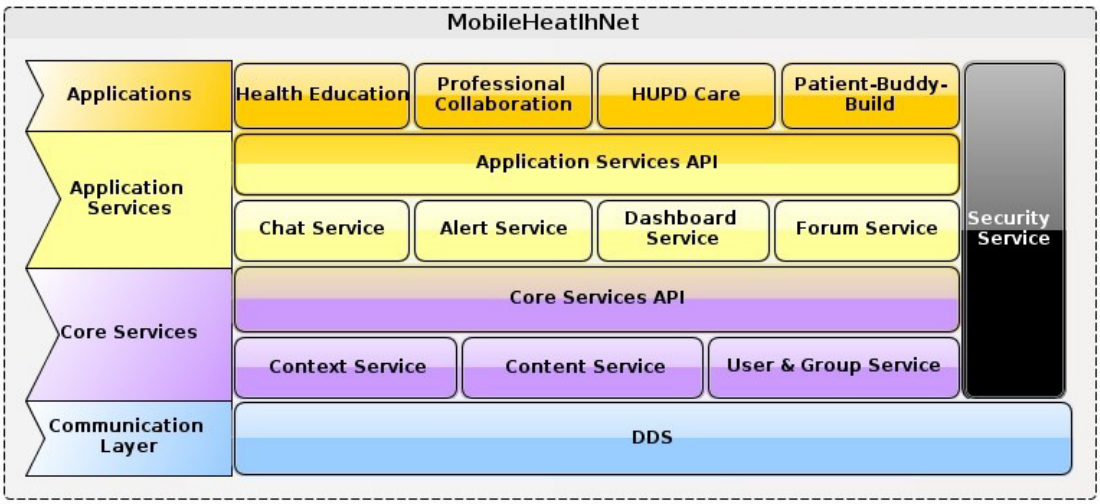

Figura 2.4 – Arquitetura do MobileHeathNet

Fonte: TELES, A.S et al. [14]

Através da figura é possível observar que o *middeware* é composto por quatro camadas, a saber, Communication *Layer*, *Core Services*, *Security Services* e *Application Services*, cujas atribuições estão descritas nas seções a seguir.

2.3.1 *Communication Layer*

Responsável pelo provimento de mecanismos de comunicação entre os componentes que compões as camadas superiores, esta camada é baseada na especificação DDS (do inglês, *Data Distribution Service*) do *Object Management Group* [19], um padrão para comunicação *publish*/*subscribe* de informações em sistemas distribuídos. Nesta abordagem, existe um espaço de dados global destinado à troca de informações na rede, onde a geração e o consumo dessas informações consiste em publicar (escrever) e subscrever (ler), respectivamente, neste espaço.

O DDS foi idealizado sobre um modelo *Data-Centric Publish-Subscribe* (DCPS) baseado em tópicos. Cada tópico representa um conjunto de informações sobre uma determinada entidade ou ação na rede social, como solicitar informações sobre um usuário, efetuar *login* no sistema, entre outras. Aplicações que necessitam compartilhar informações com outras aplicações pode utilizar o espaço global de dados informando suas intenções de publicar ou subscrever dados que são relacionados a um ou mais tópicos de interesse dos participantes [15].

A escolha do DDS como base para a comunicação no MobileHealthNet se deve ao fato de sua implementação estar centrada no paradigma de interação *publish/subscribe* para o envio e recebimento de informações em tempo real. No âmbito de ambientes móveis, este paradigma é mais eficiente que o tradicional *request/response* por não necessitar de uma conexão constante com um servidor para requisições e respostas. Esta eficiência é constatada levando-se em consideração a característica intermitente da conexão sem fio, sujeita a eventuais desconexões.

As interações ocorrem da seguinte forma. No cliente móvel, um componente denominado MobileHealthNet on Android (MOBHA) provê a interface de

comunicação entre aplicações em execução no dispositivo e os serviços da rede social providos por um servidor. Em comum com o servidor e o cliente está a camada MobileHealthNet *Framework*, considerada o centro da comunicação por conter o conjunto de tópicos, publicadores, subscritores e as políticas de qualidade de serviço utilizados para efetuar a comunicação no *middleware*.

2.3.2 *Core Services*

Esta camada disponibiliza serviços básicos a serem utilizados por aplicações e serviços específicos do *middleware*. Para isso, dispõe de três componentes: *Context Service*, *Content Service e User & Group Service.*

O *Context Service* é responsável pelo armazenamento e disponibilização de diversas informações de contexto. A localização, por exemplo, pode ser utilizada para determinar os membros da rede presentes na vizinhança com os quais o usuário costuma interagir.

O *Content Service* tem como meta principal o compartilhamento de mídia entre usuários da rede social entre usuários da rede social. Este serviço permite identificar cada mídia, como textos, fotos, áudios, vídeos, entre outros, com metainformações definidas pela aplicação, disponibilizando funcionalidades como *upload* ou *download* de mídias, apagar conteúdos, renomeá-lo, alterar as metainformações (metadados) associados, buscar as mídias através do nome, de metadados ou de seu identificador na base de dados, entre outros.

O *User & Group Service* gerencia contas de usuários e grupos, assim como relacionamentos entre usuários da rede. Este serviço permite a autenticação do usuário através de uma conta cadastrada na base de dados do *middleware*. Uma vez autenticado, o usuário poderá usufruir de todas as aplicações e todos os serviços disponibilizados pela rede social com apenas uma única credencial.

Todas as informações do usuário são criptografadas para garantir a segurança destes tipos de dados trafegados na rede. As funcionalidades presentes nesse serviço é a adição de usuário, remoção, solicitação da informação de um usuário e edição de suas informações. Ao serem cadastrados, os usuários precisam

ser definidos como um paciente ou como um profissional da saúde em razão da diferença entre algumas das informações associadas a cada um deles no momento da inclusão de seus perfis.

2.3.3 *Application Services*

As responsabilidades da camada *Application Sevices* estão inseridas no contexto de serviços típicos de redes sociais, tais como publicação de mensagens em murais (*Dashboard Service*), onde se encontram publicações mais recentes realizadas pelos usuários e que podem ser visualizadas pelas pessoas designadas por quem publica, fórum (*Fórum Service*), no qual os usuários criam tópicos sobre determinados assuntos para debate, e *chat* (*Chat Service*), no qual ocorre a comunicação entre dois ou mais usuários, além de um serviço de notificações para os usuários (*Alert Service*), utilizado para alertar a criação de novas publicações no mural ou de mensagens enviadas através do *chat*.

2.3.4 *Security Services*

Uma vez que a publicação de informações constantemente destinadas a um usuário ou grupo específico e a possibilidade de informações referentes a interações sociais serem alvo de ataques para serem utilizadas com fins maliciosos [20], a segurança e a privacidade dos dados são requisitos essenciais no contexto das redes sociais. Para suprir esta necessidade, transversal a todo o código gerado no *middleware* existe a camada *Security Services*, que garante a realização de todas as operações apenas por usuários autenticados.

Além da autenticação, a camada provê mecanismos para o estabelecimento de canais seguros de comunicação no DDS fundamentada nas propriedades básicas da integridade e confidencialidade dos dados. Vale ressaltar que a segurança do *middleware* segue o Manual de Certificação para Sistemas de Registro Eletrônico em Saúde da Sociedade Brasileira de Informática na Saúde.

2.3.5 *Applications*

Por fim, existem aplicações projetadas para utilização do *middleware* (*Applications)*: *HealthEducation*, utilizada para o compartilhamento de arquivos multimídia com o objetivo de aprimorar a educação de pacientes e profissionais da saúde através de mídias com conteúdo educacional; Professional Collaboration, aplicação que visa diminuir a distância entre os profissionais da saúde através de recursos como *chat* multiusuário, fórum de discussão e serviço de notificação que permita informar sobre a urgência de se obter o resultado de um exame, discutir a respeito do tratamento e acompanhamento de um determinado paciente; HUPD Care, que tem por objetivo explorar os conceitos das redes sociais móveis para promover a assistência prestada por especialistas responsáveis pelo atendimento de alta complexidade a profissionais da atenção básica no atendimento a casos específicos; Patient-Buddy-Build, aplicação que possibilita a criação de questionários simples e práticos a serem respondidos por pacientes para informarem periodicamente o estado de sua doença, mantendo-se, assim, a interação médico-paciente além dos encontros presenciais ou consultas, o que reduz a necessidade de visitas pós-diagnóstico e melhora a efetividade no acompanhamento de pacientes. A aplicação constitui-se o propósito da interação humano-computador proposta neste trabalho e será detalhada no próximo capítulo.

Neste capítulo foram apresentadas as fundamentações das redes sociais móveis e sua aplicação na área da saúde, componentes, funcionamento e utilização dos sistemas de diálogo falado como forma de interação humano-computador em dispositivos móveis, e a arquitetura do *middleware* MobileHealthNet, assim como o detalhamento da aplicação Patient-Buddy-Build. No próximo capítulo serão brevemente abordados o sistema operacional Android e o pacote de desenvolvimento Dragon Mobile SDK, utilizado para implementação de sistemas de diálogo falado.

3 DESENVOLVIMENTO COM ANDROID E DRAGON MOBILE SDK

Os dispositivos móveis, assim como os demais ambientes computacionais, necessitam de um sistema que gerencie os demais aplicativos e serviços executados. Conhecidos como sistemas operacionais móveis, estes são responsáveis por tarefas altamente complexas devendo ser projetados nos mínimos detalhes para garantir o pleno funcionamento do dispositivo. Na seção a seguir será abordado o sistema operacional Android, um dos sistemas mais conhecidos e utilizados atualmente em dispositivos móveis.

3.1 Sistema operacional Android

Um dos sistemas operacionais móveis de maior popularidade, sem dúvidas, é o Android. Consolidado no mercado como uma plataforma rápida e segura, o sistema do Google é mantido pela OHA (*Open Handset Aliance*), um grupo com mais de 30 (trinta) empresas unidas na busca da inovação no desenvolvimento de aplicações e serviços.

O sistema oferece suporte a diversas aplicações que são desenvolvidas no intuito de proporcionar ao usuário os mais variados tipos de experiências no uso do dispositivo móvel que vão de simples bate-papo em redes sociais, já que é nativamente integrado a elas e a serviços em nuvem, a jogos de última geração de empresas consagradas com versões específicas para a plataforma.

Para promover o surgimento de novas aplicações a todo momento, o Google disponibiliza gratuitamente um pacote de desenvolvimento de aplicações baseado na linguagem de programação Java e em arquivos no formato XML *(Extensible Markup Language),* denominado Android SDK, para garantir ao usuário a flexibilidade ao buscar aplicativos que atendam suas necessidades. Este pacote é composto por um conjunto de ferramentas e APIs que favorecem o desenvolvimento de aplicativos, que são compilados em *bytecodes* e executados em uma máquina virtual Dalvik, responsável por executar aplicações distribuídas em formato binário em qualquer dispositivo móvel. O código aberto do sistema faz com que grandes

empresas de tecnologia móvel utilizem-no de forma personalizada de acordo com seus próprios projetos.

Diante deste cenário não é surpresa o alto índice de ativação do sistema nos dispositivos móveis. Aliado ao fato do custo acessível de boa parte dos dispositivos que executam o Android como sistema operacional e do *middleware* MobileHealthNet possuir maior integração com o Android SDK, esta foi a plataforma escolhida para o desenvolvimento da interação humano-computador baseada em sistemas de diálogo para a aplicação Patient-Buddy-Build. Para tal, foram utilizados o Android SDK 21.0.1 e a Google API 17 (versão 4.2.2) do sistema.

3.1.1 Arquitetura do sistema Android

A figura a seguir ilustra a arquitetura do sistema Android em camadas.

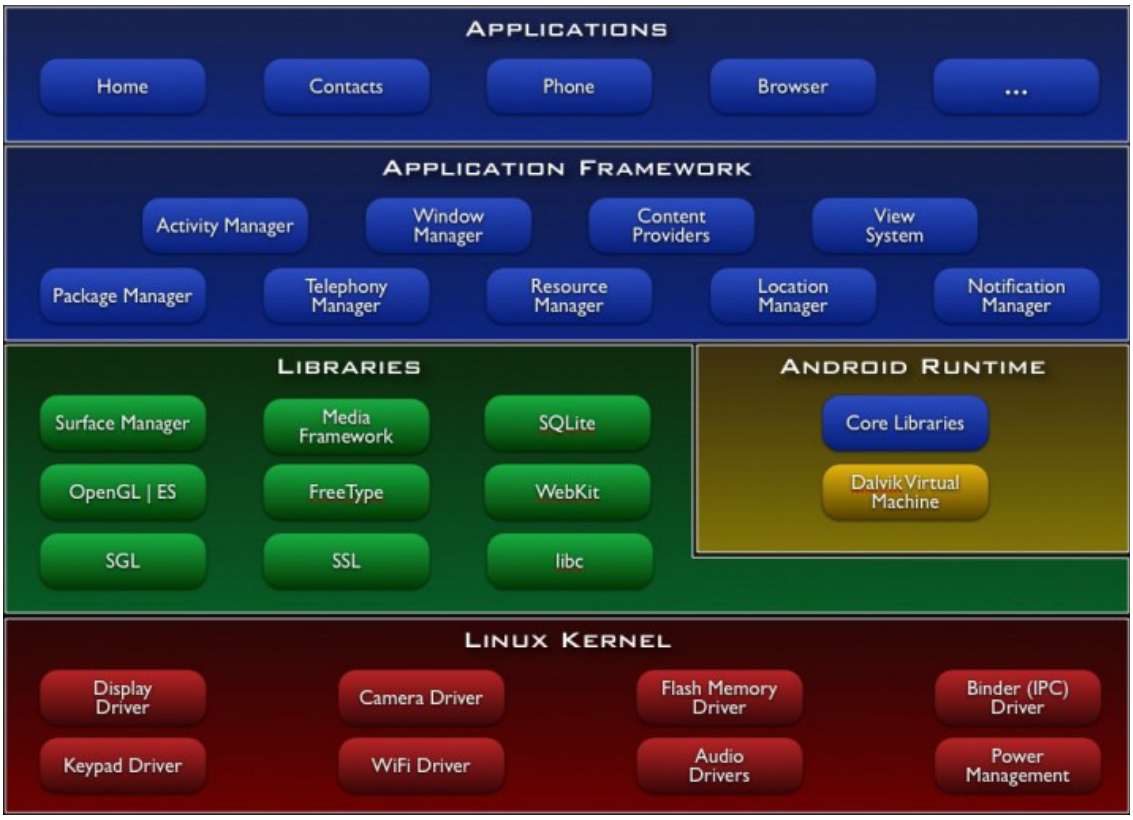

Figura 3.1 – Arquitetura do sistema Android

Fonte: Google [21]

O *Linux Kernel* é a base do sistema, que possui a versão 2.6 do Linux e inclui os serviços essenciais do Android, como o gerenciamento de memória, gerenciamento de energia, *drivers*, entre outros. A camada *Android Runtime* possui bibliotecas necessárias para as principais funcionalidades do sistema e uma Máquina Virtual Dalvik projetada para requerer pouca memória e ser instanciada várias vezes. Cada processo de uma aplicação Android roda em uma instância da Máquina Virtual Dalvik, auxiliando o sistema numa melhor gerência de memória. A camada *Libraries* possui bibliotecas bastante requisitadas para as funcionalidades da aplicação desenvolvida neste trabalho, como o *Media Librarie*, elaborada para suportar a reprodução de arquivos multimídia do tipo MPEG (*Moving Picture Experts Group*), PNG (*Portable Network Graphics*), entre outros. A camada *Application* é onde se encontram todas as aplicações Android, como a agenda telefônica e o *browser*. Nas próximas seções serão abordados alguns dos principais componentes fundamentais no desenvolvimento de aplicações que tenham o sistema Android como plataforma de execução.

3.1.2 Classe *Activity*

Baseada em uma linguagem de programação orientada a objetos, a utilização de classes é essencial no desenvolvimento de aplicações para o sistema Android. A classe *Activity* (*android.app.Activity*), é responsável por definir a exibição de informações ao usuário da aplicação. A figura a seguir representa o ciclo de vida de uma classe *Activity*.

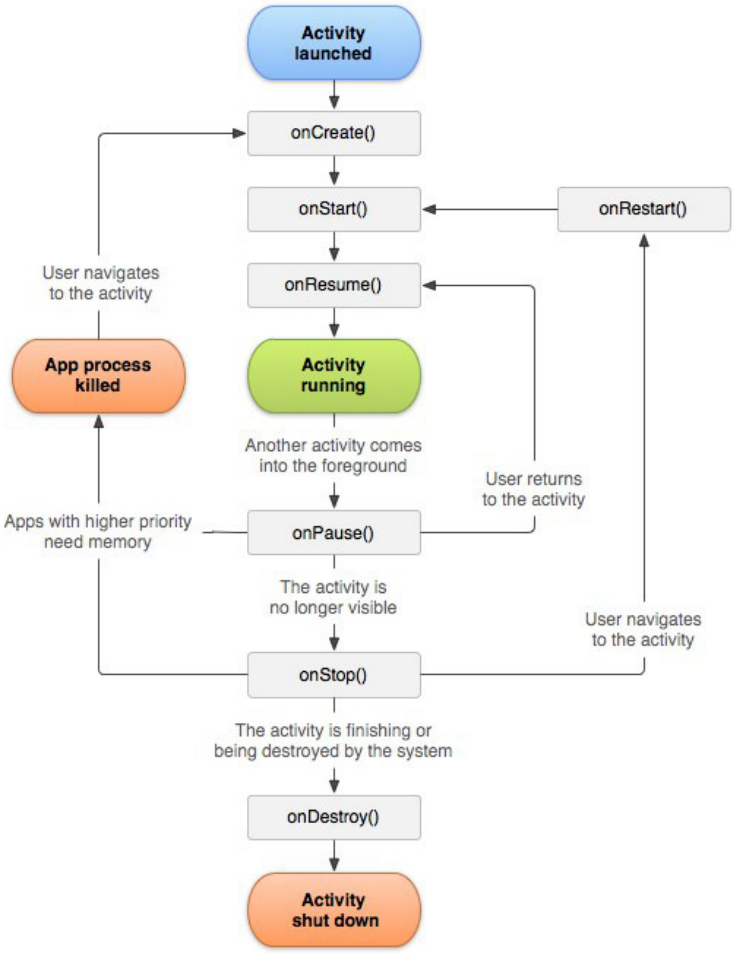

Figura 3.2 – Ciclo de vida da classe *Activity* do Android SDK

Fonte: Google [21]

Através da figura é possível notar um ciclo de vida dividido em 6 (seis) estados (métodos) intuitivos. O estado *onCreate* é assumido a partir da criação da *Activity e* permite a definição de parâmetros utilizáveis durante todo o ciclo de vida da Activity. O estado *onStart* é assumido quando a *Activity* está em evidência ao usuário. Assemelha-se bastante ao estado *onCreate* por também permitir a definição de parâmetros iniciais. Por isso, na prática o estado é geralmente desconsiderado. O estado *onResume* é assumido quando a *Activity* está retornando à execução após uma paralisação temporária, enquanto o estado *onPause* quando execução da *Activity* é paralisada, salvando o seu estado atual. O estado *onStop* é assumido quando a *Activity* é executada em segundo plano, isto é, quando alguma outra

aplicação é ativada pelo usuário e passa a ser aquela utilizada no momento por ele (primeiro plano). Ao voltar a executar a *Activity* anterior em primeiro plano, que estava no estado *onStop* quando executada em segundo plano, esta assume o estado *onStart* e, em seguida, o estado *onResume* (após assumir o estado *onPause* para recuperação do estado anterior), retornando ao ponto que se encontrava antes de assumir o estado *onStop*. Por fim, o estado *onDestroy* é assumido quando a *Activity* é finalizada, eliminando o estado salvo pelo estado *onPause* e permitindo alguns procedimentos antes de seu encerramento. O pacote de desenvolvimento do Android permite a implementação de cada um dos estados através de métodos herdados da classe *Activity*, conforme mostra a figura a seguir.

```
public class TestActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    protected void onStart() {
        super.onStart();
    }

    @Override
    protected void onResume() {
        super.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
    }

    @Override
    protected void onStop() {
        super.onStop();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
    }

}
```

Figura 3.3 – Implementação dos métodos da classe *Activity* do Android SDK

A figura mostra que cada método pode conter procedimentos executados quando o respectivo estado for assumido. Assim, o pacote de implementação proporciona maior liberdade e autonomia ao desenvolvedor para projetar funcionalidades de acordo como o ciclo de vida de cada classe *Activity* de sua aplicação.

3.1.3 Classe *Intent*

A comunicação em tempo de execução entre classes do tipo *Activity*, bem como alternância entre elas, são tarefas atribuídas à classe *Intent* (*android.content.Intent*). A figura abaixo mostra, como exemplo, a implementação de uma instância da classe *Intent* responsável pela passagem de uma classe *Activity* para outra.

```
Intent intent = new Intent().setClass(this, Test.class);
String hello = "Hello";
intent.putExtra("test", hello);
startActivity(intent);
```

Figura 3.4 – Exemplo de implementação da classe *Intent* do Android SDK

Através de uma instância classe *Intent* é possível realizar diversos procedimentos em uma comunicação entre duas classes do tipo *Activity*. Um exemplo é a passagem de dados, como cadeia de caracteres, números inteiros, entre outros, de uma para a outra. Além disso, a classe *Intent* pode ser utilizada para solicitação de tarefas como chamadas telefônicas, execução do navegador *web,* entre outras.

3.1.4 Classe *View*

Os componentes visuais presentes na visualização da aplicação, como botões, caixas de textos, entre outros, são encontrados na classe *View* (*android.view.View*), assim como a gerência customizada de *layouts.* Uma interface gráfica ou *view*, implementada através de arquivos XML em que podem ser inseridos os botões e as caixas de textos, por exemplo, é produzida para interagir diretamente com uma classe do tipo *Activity*, A figura a seguir mostra uma interface gráfica codificada em arquivo XML e sua respectiva visualização.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/title_main"/>

<Button android:layout_height="wrap_content"
    android:text="@string/button_tts"
    android:layout_width="fill_parent"
    android:id="@+id/btn_tts"/>

</LinearLayout>
```

Figura 3.5 – Exemplo de interface gráfica codificada em arquivo XML

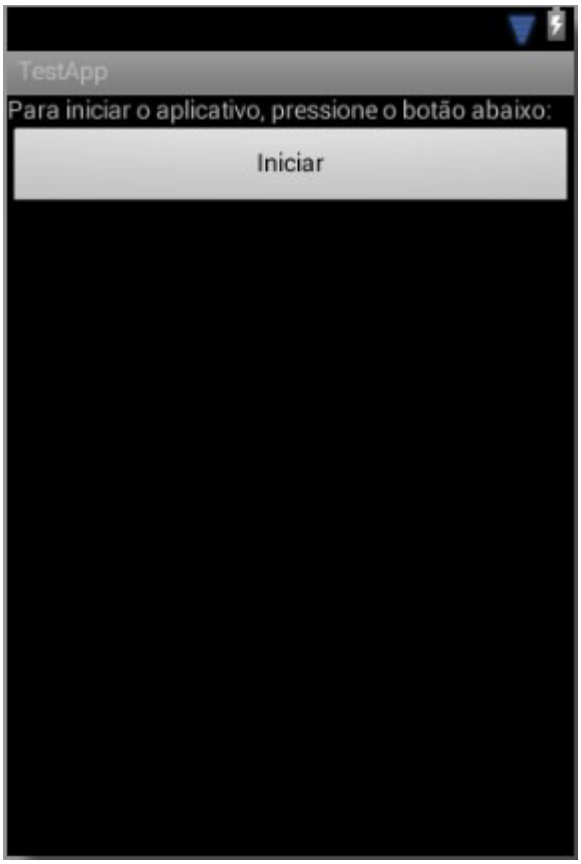

Figura 3.6 – Exemplo de interface gráfica visualizada pelo usuário

Os componentes visuais da figura podem ser manipulado através dos identificadores atribuídos a cada um deles. Assim, o nome inserido pelo usuário em um campo de texto pode ser tratada para ser inserida ao lado da mensagem “Seja bem-vindo, ” ou da forma que o desenvolvedor preferir. Esta funcionalidade será descrita na seção destinada à “Classe R”.

3.1.5 Padrão *Adapter*

Os objetos de uma classe *Activity* encontram incompatibilidade na interação com as interfaces gráficas por conta de suas origens, isto é, uma lista de objetos não é a mesma coisa que uma lista de elementos de uma interface gráfica, por exemplo. Uma solução foi a implementação de um padrão de projeto denominado *Adapte*r. A figura a seguir representa a utilização da classe Adaptar para o exemplo citado.

```
public class TestAdapter extends BaseAdapter{

    @Override
    public int getCount() {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public Object getItem(int arg0) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public long getItemId(int arg0) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public View getView(int arg0, View arg1, ViewGroup arg2) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Figura 3.7 – Exemplo de utilização do padrão *Adapter* no Android SDK

Através dos métodos herdados da classe *BaseAdapter* (*android.widget.BaseAdapter*) é possível realizar operações como obter o tamanho e determinado item da lista de elementos de uma interface gráfica, viabilizando ao desenvolvedor a interação com os dados disponibilizados pelos componentes visuais.

3.1.6 Classe *R*

A criação de uma interface gráfica através de um arquivo XML resulta na geração de uma constante do tipo inteiro com o nome do arquivo dado a mesma é criada em uma classe gerada automaticamente pelo *Android Development Tools*, *plugin* do sistema Android responsável pela conexão do ambiente de

desenvolvimento ao Android SDK, denominada classe *R*. A figura abaixo mostra um exemplo de classe do tipo *R*.

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.nuance.nmdp.sample;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int layout_bottomButtons=0x7f06000b;
        public static final int layout_resultsList=0x7f060009;
    }
    public static final class layout {
        public static final int main=0x7f030002;
        public static final int test=0x7f030003;
    }
    public static final class raw {
        public static final int beep=0x7f040000;
    }
    public static final class string {
        public static final int app_name=0x7f05000b;
        public static final int button_start=0x7f050006;
    }
}
```

Figura 3.8 – Exemplo de uma classe *R* do Android SDK

A classe *R* é necessária para que qualquer arquivo presente no projeto da aplicação seja encontrado o código da aplicação for compilado. Assim, referência do arquivo XML gerada automaticamente na classe *R* é passado como parâmetro no momento da criação da *Activity* ao método *setContentView*, responsável pela ligação entre a classe *Activity* e a interface gráfica.

3.1.7 Arquivo *AndroidManifest.xml*

O nome da aplicação e a classe *Activity* que a inicia são algumas das declarações contidas em um arquivo XML denominado *AndroidManifest*. A figura abaixo mostra um exemplo de arquivo *AndroidManifest.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name"
        android:debuggable="true">
        <activity android:name=".TestView"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".TestView"
                  android:label="@string/app_name">
        </activity>
    </application>

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.RECORD_AUDIO"></uses-permission>
<uses-permission android:name="android.permission.VIBRATE"></uses-permission>
<uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
<uses-permission android:name="android.permission.BLUETOOTH"></uses-permission>

<uses-sdk android:minSdkVersion="4"></uses-sdk>
</manifest>
```

Figura 3.9 – Exemplo de um arquivo *AndroidManifest.xml*

Através da figura é possível perceber que se trata de um arquivo de configuração. Nele estão descritos informações que vão desde o que é necessário para que a referida aplicação funcione corretamente, como a versão mínima do sistema Android, até como quais recursos do dispositivo móvel ela pode utilizar durante uma execução, como as permissões de uso.

3.1.8 Classe *Dialog*

A exibição de mensagens padrões de alerta ou operações com respostas do usuário do tipo “sim ou não”, “*ok* ou cancelar”, entre outros, por vezes torna-se mais do que necessária. A classe *Dialog* permite a geração de uma pequena janela através da qual o usuário pode, além de realizar uma decisão, inserir alguma informação solicitada. A figura a seguir representa um exemplo de utilização da classe *Dialog*.

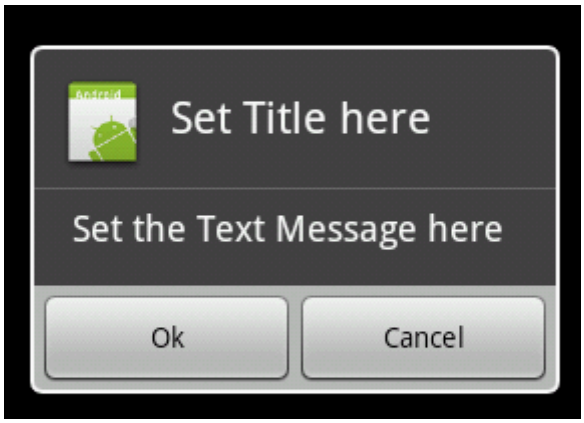

Figura 3.10 – Exemplo de utilização da classe *Dialog* do Android SDK

A figura mostra um exemplo simples de uma caixa de diálogo gerada pela classe *Dialog*. Através dela é possível confirmar ou não uma operação solicitada.

3.2 Dragon Mobile SDK

Encontrar ferramentas de desenvolvimento que ofereçam suporte ao reconhecimento de voz e à conversão de texto para fala na língua portuguesa é uma tarefa bastante árdua. Uma das poucas alternativas que suportam as duas funcionalidades em mais de 20 (vinte) idiomas, incluindo o português, é o pacote de desenvolvimento Dragon Mobile SDK [22], disponibilizado pela empresa Nuance com utilização sujeita a planos adquiridos gratuitamente e por determinados valores, dependendo das funcionalidades requeridas pelo desenvolvedor. A ferramenta

fornece os serviços de reconhecimento e síntese de texto para fala, respectivamente, entrada e saída de áudio. Para isto, dispõe de uma Interface de Programação de Aplicativos (do inglês, *Application Programming Interface*) de alto nível que executa automaticamente todas as tarefas necessárias para o reconhecimento e/ou síntese de voz, incluindo a gravação e reprodução de áudio, bem como o gerenciamento da conexão de rede, realziadas através de requisições aos serviços executados no servidor cuja utilização está associada a um identificador válido disponibilizado no momento da aquisição de um dos planos oferecidos. A figura a seguir ilustra a arquitetura do Dragon Mobile SDK.

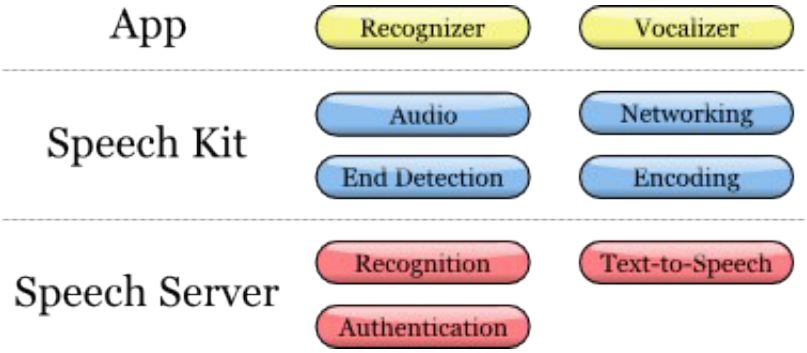

Figura 3.11 – Arquitetura do Dragon Mobile SDK

Fonte: [22]

Conforme a figura é possível identificar 3 (três) níveis, a saber, *App* (aplicação), *Speech Kit* (biblioteca) e *Speech Server* (servidor), além dos 2 (dois) componentes principais: *Recognition* (reconhecedor) e o sintetizador de texto para fala (*Text-to-Speech*). Há vários processos coordenados:

- Completa administração do sistema de áudio para gravação e reprodução (*Recognizer* e *Vocalizer*, respectivamente);
- O componente de rede (*Networking*) controla a ligação ao servidor e, no início de uma nova solicitação, automaticamente restabelece conexões.
- O detector de fim da fala (*End Detection*) determina quando o usuário parou de falar e paralisa automaticamente a gravação.

- O componente de codificação compacta (*Encoding*) e descompacta o *streaming* de áudio (*Audio*) para reduzir os requisitos de largura de banda e diminuir a latência.

O servidor é responsável pela maior parte do trabalho no ciclo de processamento de voz. O reconhecimento completo ou procedimento de síntese é realizada no servidor, consumindo ou produzindo o *streaming* de áudio. Além disso, é quem realiza a gerência da autenticação, conforme será visto na próxima seção.

3.2.1 Permissões de acesso e conexão com o servidor

Como discorrido na seção destinada ao sistema operacional móvel Android, no arquivo *AndroidManifest.xml* é possível adicionar permissões de acesso a recursos do dispositivo móvel. Para uso do Dragon Mobile SDK são necessárias permissões de acesso ao monitoramento de conexões, à Internet, ao microfone e aos fones de ouvido. O acréscimo da permissão de vibração só é necessário caso respectivo *prompt* seja utilizado. A figura a seguir mostra as permissões acrescentadas a um arquivo *AndroidManifest.xml*.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.RECORD_AUDIO"></uses-permission>
<uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
<uses-permission android:name="android.permission.VIBRATE"></uses-permission>
</ Manifest>
```

Figura 3.12 – Permissões de acesso para utilização do Dragon Mobile SDK

Fonte: [22]

Para que a conexão com o servidor seja estabelecida é necessária a validação da licença adquirido junto à biblioteca de implementação (*Speech Kit*). É inserido no código da aplicação através de um vetor estático de *bytes* armazenado em uma variável, conforme mostra o exemplo da figura a seguir.

```
static final byte[] SpeechKitApplicationKey = {(byte)0x12, (byte)0x34, ..., (byte)0x89};
```

Figura 3.13 – Exemplo de chave do Dragon Mobile SDK

Fonte: [22]

A conexão é inicializada através da chamada do método *initialize*, da classse *SpeechKit*, que recebe 6 (seis) parâmetros, a saber, o contexto da aplicação, o identificador, o endereço do servidor, uma porta, a configuração SSL e a chave. Todos estes dados são enviados via *email* no ato da aquisição de um plano oferecido pela Nuance. A figura a seguir mostra a chamada do método *initialize*.

```
SpeechKit sk = SpeechKit.initialize(context,
                                    speechKitAppId,
                                    speechKitServer,
                                    speechKitPort,
                                    speechKitSsl,
                                    speechKitApplicationKey);
```

Figura 3.14 – Exemplo de chamada do método *initialize* do Dragon Mobile SDK

Fonte: [22]

Após a verificação, a conexão finalmente é estabelecida através do método *connnect*, conforme mostra a figura a seguir.

```
sk.connect();
```

Figura 3.15 – Exemplo de chamada do método *connect* do Dragon Mobile SDK

Fonte: [22]

Se todos os dados forem validados, os serviços já se encontrarão prontos para serem utilizados.

3.2.2 Processo de conversão de texto para fala

A sequência básica do processo de texto para fala realizada pelo Dragon Mobile SDK pode ser vista na figura a seguir.

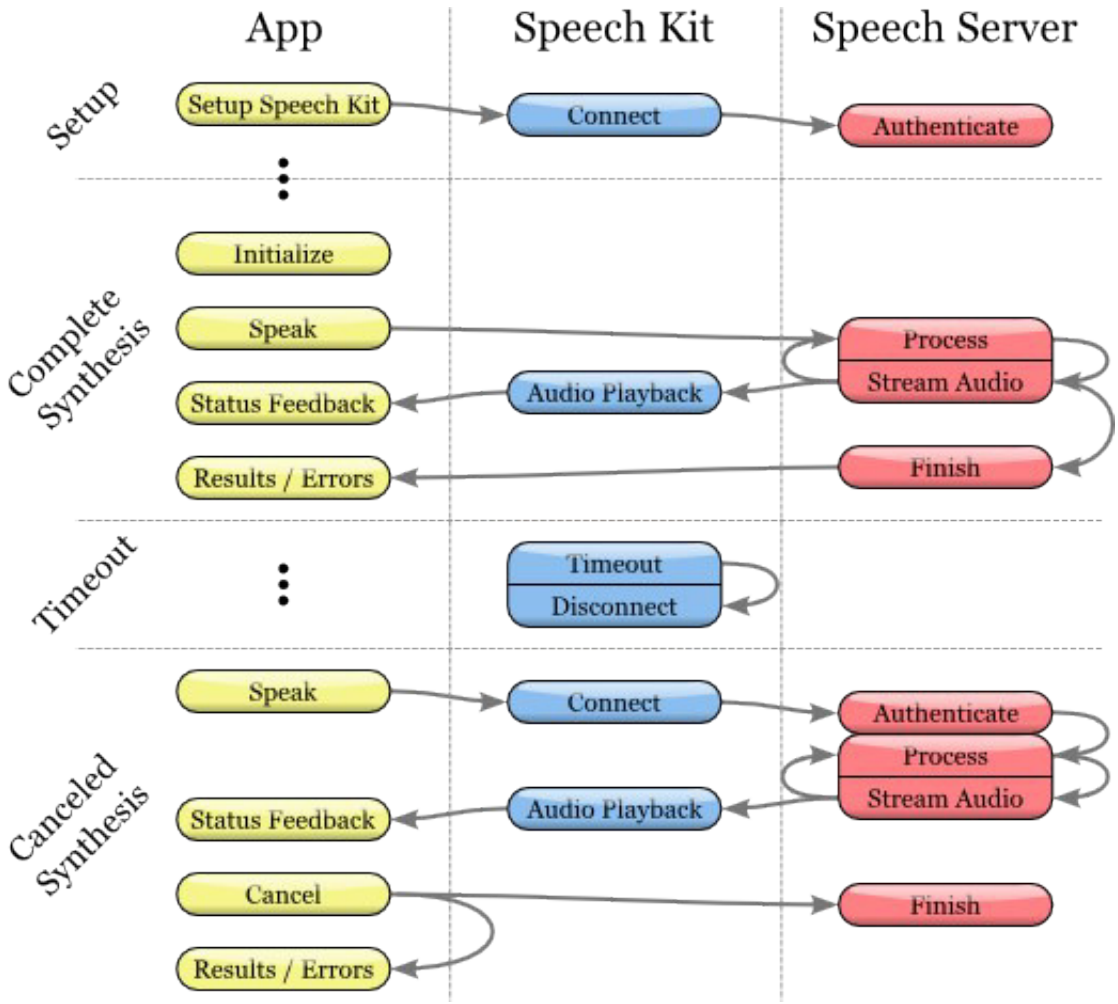

Figura 3.16 – Processo de conversão de texto para fala do Dragon Mobile SDK

Fonte: [22]

Através da figura é possível notar que no processo de texto para fala ocorrem 4 (quatro) fases distintas envolvendo os 3 (três) níveis. A fase *Setup* diz respeito à autenticação da licença necessária para utilização dos serviços, descrita no início da seção.

O processo de texto para fala ocorre na fase *Complete Synhtesis*. O método *createVocalizerWithLanguage*, da classe *Vocalizer*, conforme mostra o exemplo da

figura a seguir, inicializa um sintetizador de texto para fala no idioma escolhido recebendo como parâmetros o idioma, o contexto da aplicação e o manipulador de mensagens (*handler*) do processador, instanciado como um objeto comum na linguagem de programação Java.

```
Vocalizer voc = sk.createVocalizerWithLanguage("en_US", this, handler);
```

Figura 3.17 – Exemplo de inicialização do sintetizador de texto para fala do Dragon Mobile SDK

Fonte: [22]

A conversão de texto para fala é iniciada pelo método *speakString*, que recebe a cadeia de caracteres a ser convertida e a envia ao servidor dando início à transmissão e reprodução do áudio no aparelho. A figura seguir mostra o objeto do tipo *Vocalizer* criado na figura anterior.

```
voc.speakString("Hello world.", context);
```

Figura 3.18 – Exemplo de chamada do método sintetizador de texto para fala do Dragon Mobile SDK

Fonte: [22]

O método recebe o contexto da aplicação como parâmetro. Neste exemplo, o áudio referente à frase “Hello world” seria reproduzido através da voz preestabelecida para aquele referido idioma, que pode ser masculina ou feminina (no caso da língua portuguesa).

O serviço ainda permite a realização de procedimentos durante o processo de síntese de texto para fala. O método *onSpeakingBegin* é chamado quando a reprodução de áudio efetivamente se inicia, isto é, após o processo de conversão da cadeia de caracteres em fonemas equivalentes ao idioma escolhido. Após a conclusão da reprodução da pergunta, o método envia uma mensagem de sucesso, se a conversão e reprodução for bem-sucedida, ou nula em caso de erro. O método *onSpeakingDone*, por sua vez, é chamado após a reprodução do áudio referente à

cadeia de caracteres, adicionando uma variável de erro como parâmetro e possibilitando a chamada de procedimentos de acordo com a sua ocorrência. A figura a seguir mostra a implementação dos referidos métodos.

```
public void onSpeakingBegin(Vocalizer vocalizer, String text, Object context) {
        // update UI to indicate that text is being spoken
}
public void onSpeakingDone(Vocalizer vocalizer, String text, SpeechError error, Object context) {
    if (error != null) {
        // Present error dialog to user
    } else {
        // Update UI to indicate speech is complete
    }
}
```

Figura 3.19 – Implementação dos métodos *onSpeakingBegin* e *onSpeakingDone* do Dragon Mobile SDK

Fonte: [22]

Através destes métodos é possível realizar procedimentos adicionais no início e no fim da reprodução do áudio. Assim, é possível chamar uma nova *Activity* após a reprodução de uma mensagem de boas-vindas, por exemplo.

Durante todo o processo são gerados comentários (*feedback*) ao usuário de acordo com a situação ocorrida. A fase *Canceled Synthesis* consiste nas mesmas etapas descritas anteriormente, acrescentando, porém, a ocorrência de cancelamento da conversão durante o processamento.

3.2.3 Processo de conversão de fala para texto

O Dragon Mobile SDK realiza o processo de conversão de fala para texto de acordo com a sequência básica apresentada na figura a seguir.

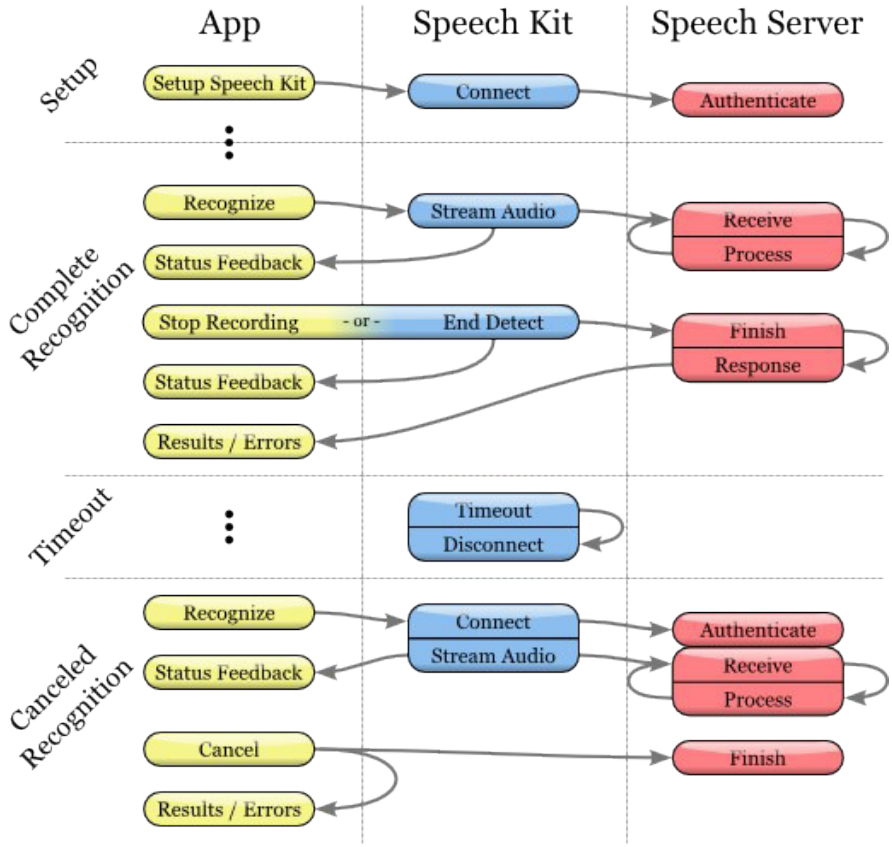

Figura 3.20 – Processo de reconhecimento de voz do Dragon Mobile SDK

Fonte: [22]

A figura mostra que o processo de reconhecimento de voz é semelhante ao processo de texto para fala. As fases *Setup* e *Timeout* são as mesmas e as demais fases também correspodem a um processo completado e interrompido.

O reconhecimento de voz propriamente dito ocorre na fase *Complete Recognition*. O método *createRecognizer* da classe *SpeechKit,* inicializa o processo de reconhecimento da fala recebendo como parâmetros a sequência de sons que vai compor a resposta do usuário, bem como a indicação de seu final (realizada pelo atributo *EndOfSpeechDetection* da classe *Recognition*), o idioma da fala (neste caso, o português), o contexto da aplicação e o manipulador de mensagens de resultado do reconhecedor, semelhante ao que ocorre no processo de conversão de texto para fala, responsáveis por gerar comentários ao usuário. A figura a seguir mostra um exemplo de chamada deste método.

```
recognizer = sk.createRecognizer(Recognizer.RecognizerType.Dictation,
                                 Recognizer.EndOfSpeechDetection.Short,
                                 "en_US", this, handler);
```

Figura 3.21 – Exemplo de chamada do método *createReconizer* do Dragon Mobile SDK

Fonte: [22]

O método *start,* por sua vez, é o responsável por habilitar o microfone para gravação da fala. Após a identificação de uma pausa gerada pela ausência da vibração das ondas de áudio, este método faz com que a sequência de sons seja convertida em pequenos arquivos de áudio e transferida para o servidor que, ao recebê-la, realiza um processamento interno de comparação de cada fonema no referido idioma. O método *onResults,* da classe *Recognizer.Listener,* permite a manipulação de uma lista de cadeias de caracteres que provavelmente correspondem ao conteúdo da resposta do usuário. A figura a seguir mostra um exemplo de implementação deste método.

```
public void onResults(Recognizer recognizer, Recognition results) {
        String topResult;
if (results.getResultCount() > 0) {
    topResult = results.getResult(0).getText();
    // do something with topResult...
}
}
```

Figura 3.22 – Exemplo de implementação do método *onResuls* do Dragon Mobile SDK

Fonte: [22]

O método recebe como parâmetros de execução o objeto reconhecedor e a lista de resultados encontrados por ele através da captação de áudio. Neste exemplo é escolhido o resultado constante na primeira posição da lista. Esta posição é privilegiada por estar reservada para o resultado considerado mais próximo do conteúdo da fala gerada pelo usuário. Para tal, a fórmula é baseada em pontuações

(*scores*) de probabilidades geradas de acordo com uma comparação das faixas de áudio.

O reconhecedor também está sujeito a erros. A lista inclui falha na conexão com o servidor, no processamento de texto para fala, de fala para texto e na indicação de um cancelamento de um dos serviços. O método *onError* é chamado no momento da ocorrência de um deles. A figura a seguir mostra a implementação do método *onError*.

```
public void onError(Recognizer recognizer, SpeechError error) {
    // Inform the user of the error and suggestion
}
```

Figura 3.23 – Implementação do método *onError* do Dragon Mobile SDK

Fonte: [22]

Por fim, os métodos *onRecordingBegin* e *onRecordingDone* são chamados no momento do início e do fim da habilitação do microfone para a gravação de áudio. A figura a seguir mostra um exemplo de implementação destes métodos.

```
public void onRecordingBegin(Recognizer recognizer) {
    // Update the UI to indicate the system is now recording
}
public void onRecordingDone(Recognizer recognizer) {
    // Update the UI to indicate that recording has stopped and the speech is still being processed
}
```

Figura 3.24 – Implementação dos métodos *onRecordingBegin* e *onRecordingDone* do Dragon Mobile SDK

Fonte: [22]

A utilização dos métodos *onResults*, *onErrors*, *onRecordingBegin* e *onRecordingDone* é particularmente importante no sentido de oferecer possibilidades interessantes ao desenvolvedor como, por exemplo, implantação de comandos de voz. A cadeia de caracteres do topo da lista de resultados como a palavra “sair”, por exemplo, pode chamar um procedimento através do método *onResults* para encerrar a aplicação. A ocorrência de um erro pode chamar outro

procedimento através do método *onError* para reproduzir o áudio referente ao texto “erro encontrado”, assim como os textos “gravando” e “processando” através dos métodos *onRecordingBegin e onRecordingDone*, respectivamente.

As demais fases correspondem aos casos alternativos do processo. A fase *Timeout* é responsável por encerrar a conexão após um longo período de inatividade do usuário, evitando recursos despendidos com conexões desnecessárias ao servidor. Por fim, a fase *Canceled Recognition* apenas acrescenta a ocorrência de cancelamento do reconhecimento durante o processamento, permanecendo as mesmas etapas descritas anteriormente.

Neste capítulo foram abordados diversos aspectos funcionais sobre o sistema operacional móvel Android e dos pacotes de desenvolvimento Android SDK e Dragon Mobile SDK, utilizado para o desenvolvimento de aplicações e implementação de sistemas de diálogo falado, respectivamente, para ambientes computacionais móveis. O capítulo seguinte detalha a aplicação Patient-Buddy-Build, assim como a análise e o projeto da interação humano-computador baseada em sistema de diálogo falado proposta para a referida aplicação.

4 APLICAÇÂO COLABORATIVA NA ÁREA DA SAÚDE

A aplicação Patient-Buddy-Build [23] é desenvolvida utilizando recursos do *middleware* MobileHealthNet e tem como especificidade a aplicação de redes sociais móveis na área da saúde. Ela consiste de uma ferramenta para a geração de aplicações móveis, contendo um questionário customizado, para o acompanhamento à distância de pacientes com doenças crônicas, escolhidos como público-alvo por possuírem um alto custo de tratamento e, na sua maioria, paciente de risco [24].

A customização dos questionários ocorre a partir de informações de uma base de conhecimento onde estão contidos dados sobre o paciente a partir do perfil na rede social, os sensores disponíveis, a doença e o modo de acompanhamento. Estes questionários, aliados às informações extraídas dos sensores, gera a informação necessária para um acompanhamento médico levando-se em consideração as hipóteses de que o paciente dispõe de um *smartphone* com sensores para captura da respectiva informação de contexto, do acesso constante à rede de dados, do diagnóstico prévio da doença, do compromisso do profissional de saúde em consultar periodicamente os dados recebidos de seus pacientes e da constatação de sintomas a partir de sensores móveis ou de perguntas feitas ao paciente.

Para alcançar seu objetivo, são analisados representações de conhecimento médico e de dados de contexto através da utilização de ontologias [25], descritas utilizando o modelo OWL (do inglês, *Ontology Web Language*). São elas:

- **Ontologia da doença**: Contém dados sobre a doença relevantes para aplicação no sentido de monitoramento do paciente. Por exemplo, informações de sintomas, nome da doença e descrição;

- **Ontologia do questionário**: Descreve características das perguntas, como por exemplo, o tamanho da fonte a ser utilizada, cor do texto, entre outras. Além disso, determina os tipos de respostas permitidos, por exemplo, múltipla escolha, caixa de texto ou utilizando o acelerômetro disponibilizado pelo

dispositivo móvel. Neste último caso são criados padrões de movimentação do dispositivo móvel, e detectados pelo acelerômetro, para representar as respostas;

- **Ontologia do ambiente**: Descreve quais informações de contexto pervasivo serão capturadas do ambiente. Essas informações podem ser obtidas a partir de sensores ou serviços. Nestes serviços, coordenadas adquiridas a partir do GPS são passadas a fim de obter como resposta informações do local onde o usuário se encontra, tais como clima, temperatura, umidade do ar, entre outras;

- **Ontologia de controle**: Define os possíveis estados em que o paciente pode se encontrar durante o seu acompanhamento. Como representado na figura a seguir, estes estados são representados como: Saudável, Pré-crítico e Crítico. Esta ontologia descreve como o conhecimento de todas as outras ontologias será aplicado para monitorar o paciente.

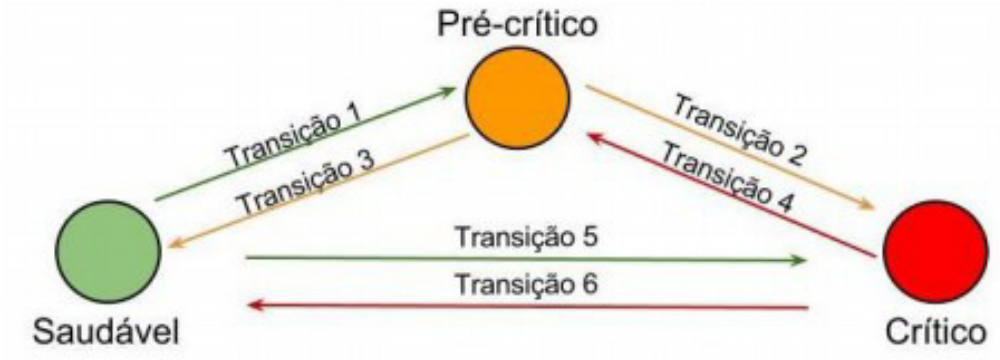

Figura 4.1 – Ontologia de controle da aplicação Patient-Buddy-Build

Fonte: ALMEIDA, V. P et al. [26]

Dependendo do estado em que o paciente se encontre, a aplicação se comporta de uma dada maneira e gera um determinado questionário, o qual é diferenciado pelas informações coletadas. Isso ocorre devido cada estado caracterizar um quadro de preocupações diferentes sobre o paciente. Por exemplo,

o estado Pré-crítico pode significar que o paciente esteja em uma altitude maior do que 2400 (dois mil e quatrocentos metros) metros e a aplicação deverá estar preocupada em perguntar se o paciente sente algum sintoma relacionado a altitude, tais como dor de cabeça, náuseas, febre, entre outros, ou até mesmo se este paciente faz uso de algum tipo de anticoagulante. Caso ele faça uso de algum anticoagulante, a aplicação passa para o estado Crítico e imediatamente o seu comportamento muda e pode dar um retorno ao paciente para que ele desça a uma altitude menor do que 2400 metros.

Ressalta-se que o objetivo do processo de geração do questionário é colher informações do paciente, para que o profissional da saúde responsável possa realizar diagnósticos. A aplicação não faz nenhum tipo de diagnóstico, ela somente sugere alguns quadros possíveis e os envia ao profissional da saúde, junto com todas as informações coletadas para que ele possa dar a palavra final e, por exemplo, notifique o paciente para que ele compareça ao hospital.

A existência dos estados na ontologia de controle tem o objetivo de guiar a aplicação no processo de coleta dos dados sobre o paciente. Eles definem quais dados a aplicação irá se preocupar em monitorar. A aplicação irá se comportar de acordo com a informação descrita na ontologia de controle. Através dela, o profissional da saúde pode definir a maneira a qual ele considera mais apropriada para monitorar seu paciente.

Assim, o resultado consiste em um *framework* para geração de aplicações móveis, cada uma responsável pelo acompanhamento de um paciente específico que é realizado através da coleta de informações do ambiente e em resposta aos questionários gerados automaticamente. A ideia é que estes questionários possam ser respondidos de forma rápida e prática, sem entrada excessiva de texto e predominância de respostas de múltipla escolha. Neste cenário, a utilização de uma interface baseada em sistema de diálogo falado se mostra como uma opção pontual diante da interface textual encontrada na maioria dos dispositivos móveis. As informações colhidas são enviadas ao profissional de saúde para uma avaliação do estado do paciente e, através de um sistema de notificações, a sua presença pode ser solicitada em um hospital para, por exemplo, uma mudança de tratamento.

4.1 Interação humano-computador baseada em sistemas de diálogo falado

A alternativa de interação humano-computador apresentada ao preenchimento do questionário da aplicação Patient-Buddy-Build é baseada em sistemas de diálogo falado e implementada através da utilização dos pacotes de desenvolvimento Android SDK e Dragon Mobile SDK visando a plataforma Android como ambiente de execução móvel. O resultado pode ser conferido através da metodologia adotada em cada componente considerando a arquitetura de módulos que rege um sistema de diálogo falado (ver figura 1), assim como os diagramas de classes e de sequência apresentados na notação UML (do inglês, *Unifield Modeling Language*).

4.1.1 Arquitetura de módulos do sistema de diálogo falado

A proposta da interação humano-computador baseada em sistemas de diálogo falado basicamente faz com que o dispositivo móvel interaja com o usuário fazendo-lhe uma pergunta do questionário e ouvindo sua respectiva alternativa, isto é, reproduz o conteúdo correspondente à questão e capta o áudio da voz referente à resposta através de alto-falantes e microfone embutidos, respectivamente. Em outras palavras, o ciclo do diálogo inicia-se com o módulo gerenciador de diálogo, cuja funcionalidade é de responsabilidade da classe *DialogManager*, incorporando também as atribuições dos módulos gerador e de compreensão da linguagem. Por isso, é nesta classe que estão dispostas as cadeias de caracteres referentes às perguntas e respostas que compõem o questionário a ser respondido, sendo incumbida de selecionar as palavras a serem transformadas pelo módulo seguinte, a saber, o módulo de síntese texto-para-fala, funcionalidade disponibilizada pela classe *SpeechVocalizer*. De início, os caracteres que compõe a primeira pergunta do questionário são selecionados e, após a sintetização, reproduzidas ao usuário através das saídas de som do dispositivo móvel, como alto-falantes ou fones de ouvido.

A classe *SpeechRecognition* é responsável pelo módulo de reconhecimento de voz. Nela existe uma sequência de procedimentos necessários para transformação do áudio em texto. Após a realização do reconhecimento de voz e a sua transformação em sequências de caracteres torna-se necessária a atribuição de significados a estas palavras, tarefa esta que é responsabilidade do módulo de compreensão da linguagem, também de responsabilidade da classe *DialogManager*. De posse da resposta do usuário esta classe trabalha na lista de possíveis textos resultantes do áudio capturado da voz do usuário. Esta lista é então comparada com uma das respostas previamente geradas para aquela pergunta. Se uma das possíveis respostas do usuário for válida, isto é, coincidir com alguma das repostas preestabelecidas, a pergunta é considerada respondida e a próxima pergunta é gerada, reiniciando-se o ciclo. Caso contrário, a aplicação exibe o erro pelo qual a resposta não foi entendida e refaz a leitura da pergunta, procedimento realizado até que uma reposta do usuário seja considerada válida, isto é, corresponda a uma de suas alternativas.

Para melhor entendimento, cada componente será detalhado a partir dos diagramas de classe e de sequência analisados nas próximas seções.

4.1.2 Diagrama de classes

Com a implementação pautada no paradigma da programação orientada a objetos, a estrutura da interação humano-computador proposta é constituída por classes, conforme ilustra o diagrama a seguir.

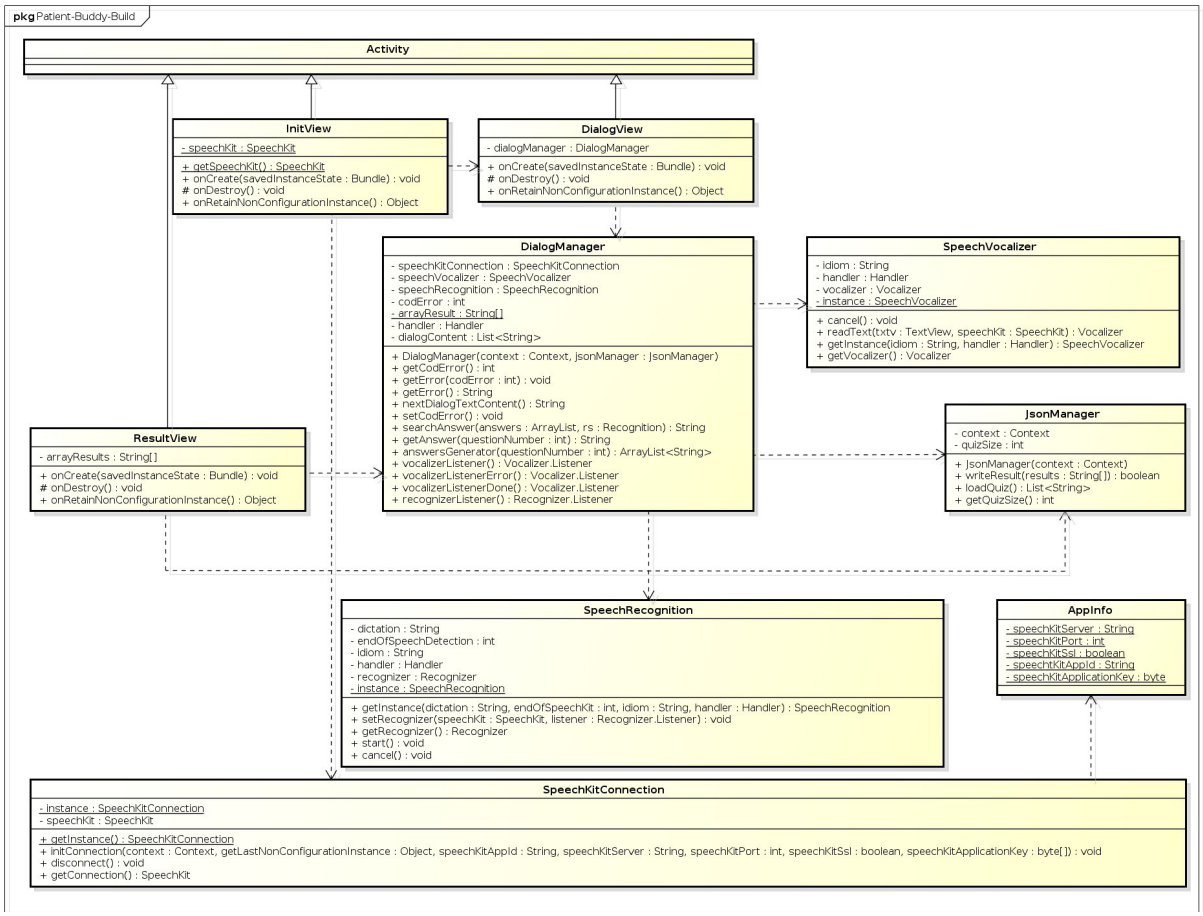

Figura 4.2 – Diagrama de classes da interação humano-computador proposta

Através do diagrama é possível notar a relação de dependência entre as classes. A visualização da aplicação é constituída por 3 (três) classes, a saber, *InitView*, *DialogView* e *ResultView* (pertencentes ao pacote *example.pbb.sds.view*), que herdam as funcionalidades da classe *Activity*. A classe *InitView* é responsável por iniciar a aplicação através do método *getSpeechKit*, que por sua vez inicializa o processo de habilitação dos serviços do Dragon Mobile SDK. Para tal, uma instância da classe *SpeechKitConnection* é gerada após a validação dos dados referentes à licença adquirida, constantes na classe *AppInfo* (ambas as classes pertencentes ao pacote *example.pbb.sds.connection*). Uma vez constatada a validação, os serviços de reconhecimento de voz e de texto para fala estarão habilitados e acessíveis através de instâncias das classes *SpeechRecognition* (*example.pbb.sds.recognition*) e *SpeechVocalizer* (*example.pbb.sds.tts*), respectivamente.

A classe *DialogManager* (*example.pbb.sds.manager*) *é* mais complexa da aplicação por ser responsável pelo controle de todo o diálogo ocorrido com o usuário. O questionário gerado pela aplicação Patient-Buddy-Build é disponibilizado em um arquivo de texto denominado JSON (*JavaScript Object Notation*) [27], uma formatação leve de troca de dados baseada em um subconjunto da linguagem de programação JavaScript constituído em duas estruturas: uma coleção de pares nome/valor, como objetos, estruturas, entre outras, e uma lista ordenada de valores, como vetores, listas ou sequências. A figura a seguir mostra questionário representado em um arquivo no formato JSON.

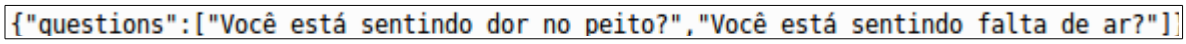

Figura 4.3 – Questionário representado em arquivo no formato JSON

O questionário se encaixa na segunda estrutura do formato JSON descrita anteriormente. O campo *questions* é o identificador do vetor de cadeias de caracteres correspondentes às perguntas do questionário, dispostas entre aspas e separadas por vírgulas, indicando suas respectivas posições.

Arquivos no formato JSON precisam ser tratados para terem seu conteúdo disponível na linguagem Java. A classe *JsonManager* (*example.pbb.sds.manager*) é responsável pela leitura e escrita de arquivos JSON através dos métodos implementados pela biblioteca *json-simple*, que converte arquivos JSON em objetos Java e vice-versa. A classe *JsonManager* recebe um arquivo denominado *quiz.json*, semelhante ao apresentado na figura anterior, através do método *loadQuiz* e o converte em uma lista de cadeias de caracteres, referentes a cada pergunta. De posse desta lista, a classe *DialogManager* constrói outra lista de cadeias de caracteres, denominada *dialogContent*, referente à sequência de frases do diálogo a serem lidas pelo método *readText*, da classe *SpeechVocalizer*, acessado através do respectivo objeto instanciado. Esta sequência é criada com uma mensagem de boas-vindas ao usuário inserida na primeira posição da lista. Em seguida, todas as perguntas são adicionadas por ordem numérica. Por fim, uma mensagem de

agradecimento, a ser lida quando o questionário for respondido por inteiro pelo usuário, é inserida na última posição da lista.

O método *nextDialogTextContent* é responsável pela condução do fluxo do diálogo, que é iniciado com a leitura da mensagem de boas-vindas ao usuário. Posteriormente, a primeira pergunta do questionário é lida e o reconhecimento de voz é habilitado logo em seguida através do método *start* da classe *SpeechRecognition*, também através do respectivo objeto instanciado. O reconhecimento ficará ativo até a detecção do fim da fala, indicado pelo atributo *endOfSpeechDetection* da mesma classe, quando será processada uma lista de cadeias de caracteres da classe *Recognition* que possivelmente correspondem ao que foi falado pelo usuário. O método *answersGenerator* é responsável pela geração da lista de respostas predefinidas para cada pergunta, isto é, as possíveis alternativas para o usuário. Qualquer resposta que não corresponda a uma destas alternativas fará com que a pergunta não seja respondida. Esta verificação é atribuída ao principal método da classe *DialogManager*, a saber, *searchAnswer*, que recebe justamente como parâmetros as listas de respostas predefinidas pelo método *answersGenerator* e de resultados encontrados no processamento da fala do usuário. De posse dos dois parâmetros, o método realiza uma varredura das cadeias de caracteres pertencentes à lista de resultados encontrados comparando-as com cada resposta constante na lista de respostas predefinidas. No caso da lista de respostas predefinidas para a primeira pergunta ser constituída das cadeias de caracteres “Sim” e “Não”, por exemplo, o método buscará em cada posição da lista de respostas encontradas pelo reconhecedor de fala se existe alguma ocorrência destas duas palavras. Este processo resulta em 3 (três) diferentes situações: a resposta é encontrada, a resposta é parcialmente encontrada (no caso de respostas com mais de uma palavra como “Sim, frequentemente”, por exemplo) ou a resposta não é encontrada. Estas duas últimas situações serão sinalizadas através do método *getError,* responsável por armazenar a mensagem do respectivo erro que será lida ao usuário, podendo também indicar o cancelamento do reconhecimento (realizada pelo próprio usuário) e outros resultantes do processamento do próprio Dragon Mobile SDK, conforme visto na seção destinada ao pacote de desenvolvimento. Nestes casos (exceto no cancelamento do reconhecimento, onde

a pergunta é lida novamente somente por comando do usuário), o método *nextDialogTextContent* envia um sinal à classe *DialogView* para que a pergunta seja lida novamente e o reconhecimento habilitado logo em seguida. Em caso de resposta encontrada com sucesso, o sinal é para que seja exibida uma mensagem indicando o reconhecimento de resposta bem-sucedido e para que a próxima pergunta seja lida em seguida, reiniciando-se o ciclo após o armazenamento da pergunta e da resposta no vetor de resultados, a saber, *arrayResult*.

Ao final do questionário é exibida a mensagem de agradecimento e disponibilizado o acesso ao resultado através da classe *ResultView*, que acessa o conteúdo do vetor *arrayResult* e aciona o método *writeResult*, da classe *JsonManager*, responsável por gerar o arquivo de saída da aplicação no formato JSON denominado *result.json*. Um exemplo da estrutura deste arquivo é apresentada na figura a seguir.

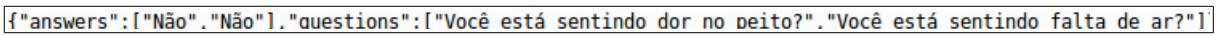

Figura 4.4 – Resultado representado em arquivo no formato JSON

Neste arquivo, o campo *answers* é adicionado seguido dos respectivos valores encontrados para cada pergunta do campo *questions*, constantes no arquivo de entrada da aplicação.

4.1.3 Diagrama de sequência

As sequências de execução da aplicação segue os fluxos apresentados no diagrama a seguir.

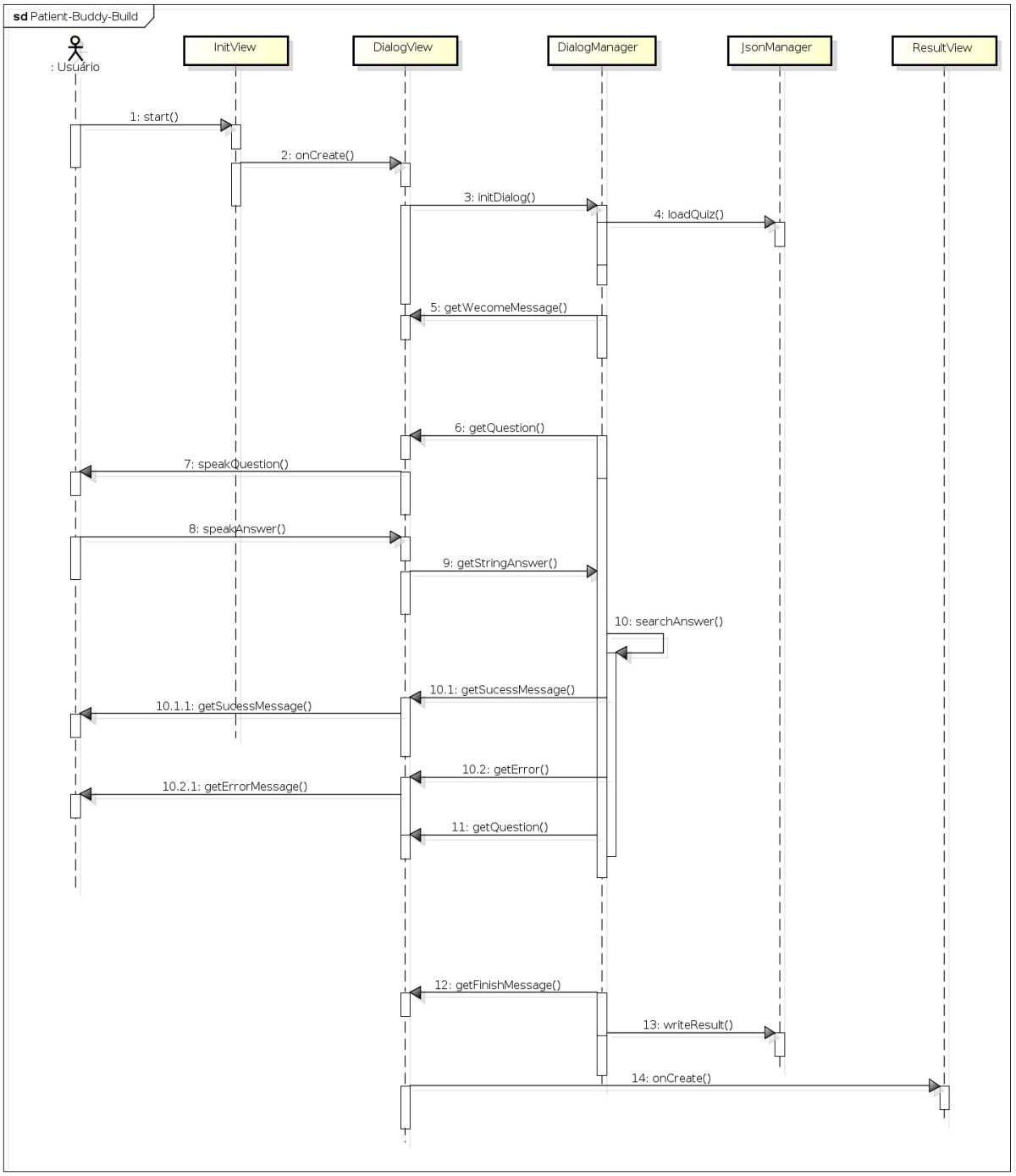

Figura 4.5 – Diagrama de sequência da interação humano-computador proposta

Considerando que as classes *AppInfo* e *SpeechKitConnection* basicamente realizam a conexão do Dragon Mobile SDK e instâncias das classes *SpeechVocalizer SpeechRecognition* são invocadas pela classe *DialogManager* apenas para a leitura e reconhecimento, respectivamente, o diagrama apresenta a

sequência realizada para cada pergunta existente no questionário. O usuário inicia a aplicação através da *InitView,* responsável pela exibição da tela inicial composta de uma mensagem de boas-vindas e um botão para a validação dos dados referentes à licença do Dragon Mobile SDK e, consequentemente, para a inicialização do questionário. Após a validação, a tela de diálogo é exibida através da *DialogView*, que por sua vez aciona a classe *DialogManager*, responsável por receber da classe *JsonManager* o questionário no formato JSON. A partir de então, o seguinte fluxo se repete até a última pergunta do questionário: uma pergunta é lida, o reconhecimento habilitado e a resposta analisada. Caso a resposta seja equivalente a uma das respostas preestabelecidas, uma mensagem indicando que a resposta foi reconhecida é exibida e a próxima pergunta é lida; caso contrário, uma mensagem de erro é exibida e a pergunta atual é lida novamente. O processo se repete até que a última pergunta seja lida e sua respectiva resposta seja reconhecida, o que faz com que as respostas sejam escritas em um arquivo de saída no formato JSON e a tela de resultado seja exibida, através das classes *JsonManager* e *ResultView*, respectivamente.

4.1.4 Interfaces gráficas

Por se tratar de uma aplicação baseada em sistemas de diálogo falado, a interface gráfica foi projetada de forma simples em razão da voz ser o principal modo de interação. A aplicação tem início a partir da tela a seguir.

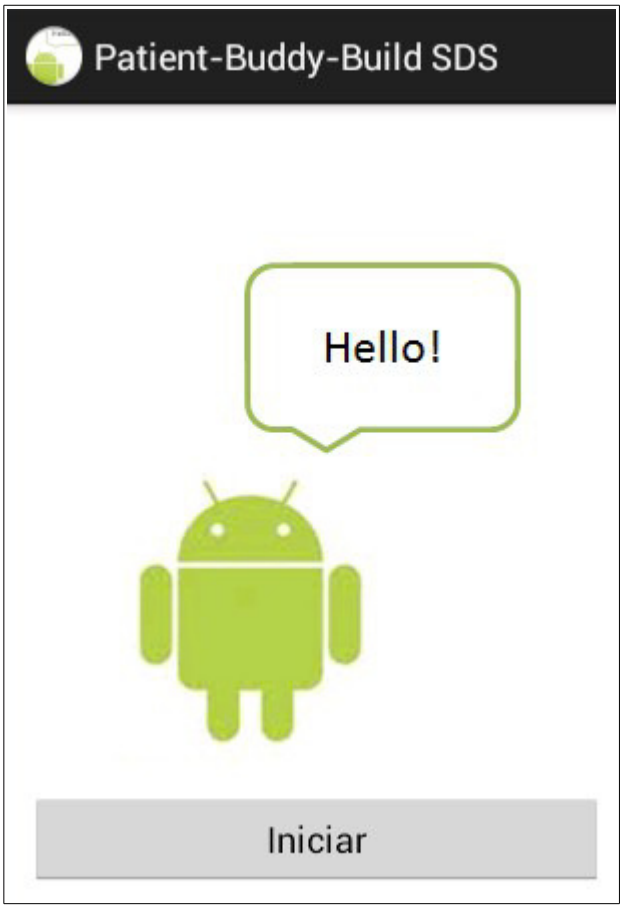

Figura 4.6 – Tela inicial

Conforme mostra a figura, a tela inicial (e as demais) apresenta um único botão, denominado Iniciar. Ela é construída através da classe *InitView*, responsável por inicializar a conexão com o Dragon Mobile SDK. Após a validação, a tela de diálogo é exibida junto à mensagem de boas-vindas, que é lida logo em seguida. A figura a seguir mostra a tela de diálogo nesta situação.

Figura 4.7 – Tela de diálogo exibindo mensagem de boas-vindas

O texto referente a mensagem de boas-vindas é o primeiro a ser exibido na tela de diálogo para, logo em seguida, ser lido ao usuário. É possível notar também que, além do botão e da mensagem, existem outros componentes nesta tela, a saber, o indicador de estado do microfone, indicando se o mesmo está habilitado ou não, e o texto de interação com o usuário, que indica ações como aguardar e falar, além de mensagens de erros e confirmações de reconhecimento de resposta. Vale ressaltar que o microfone permanece desabilitado enquanto a síntese texto-para-fala estiver sendo executada e habilitado quando o reconhecedor é iniciado, desabilitando-o novamente após a detecção do fim da fala do usuário.

Após a leitura da mensagem de boas-vindas, a primeira pergunta do questionário é exibida e lida ao usuário logo em seguida. A figura a seguir mostra um exemplo da tela de diálogo na referida situação.

Figura 4.8 – Tela de diálogo exibindo uma pergunta

Após a leitura da pergunta, a figura do microfone é exibida na cor verde junto à mensagem “Fale agora!” para indicar ao usuário que o reconhecimento de voz está habilitado. A partir de então, o usuário dispõe de duas opções: falar ou cancelar o reconhecimento através do botão “Parar”, que também é habilitado paralelamente ao início do reconhecimento. Caso o usuário opte pelo cancelamento, o reconhecimento é cancelado, a mensagem “Reconhecimento cancelado” é exibida e lida ao usuário e o botão “Repetir pergunta” é habilitado, bastando ser acionado para que a pergunta atual volte a ser lida e o reconhecimento novamente habilitado na sequência. A figura a seguir mostra um exemplo desta situação.

Figura 4.9 – Tela de diálogo exibindo mensagem de reconhecimento cancelado

Além do cancelamento do reconhecimento por opção do usuário, o mesmo pode ocorrer por conta de algum erro no serviço do Dragon Mobile SDK. Neste caso, é exibida e lida a seguinte mensagem: “Erro no serviço de reconhecimento. Por favor, tente novamente.” Após a leitura, a pergunta é lida novamente e o reconhecimento novamente habilitado na sequência. A figura a seguir mostra um exemplo da referida situação.

Figura 4.10 – Tela de diálogo exibindo mensagem de erro no serviço de reconhecimento

Caso o usuário opte por falar após a habilitação do reconhecimento, são possíveis duas situações: resposta desconhecida ou reconhecida. A primeira situação ocorre quando o resultado do reconhecimento de áudio da voz do usuário não confere com nenhuma das opções de resposta predefinidas para a pergunta em questão. Neste caso, são possíveis mais duas ocorrências: resposta parcial ou totalmente desconhecida. A figura a seguir demonstra um exemplo destas situações.

Figura 4.11 – Tela de diálogo exibindo mensagem de resposta desconhecida

As situações que levam à ocorrência deste erro são as seguintes: nenhuma das respostas encontradas pelo reconhecedor de voz corresponde às respostas preestabelecidas para a pergunta em questão ou a resposta foi parcialmente reconhecida (apenas o “sim” das respostas “sim, um pouco” e “sim, às vezes”). No entanto, o complemento não foi possível ser identificado (“um pouco” ou “às vezes”). Assim, a pergunta então é lida novamente e o reconhecimento habilitado logo em seguida.

Após a conclusão do questionário, é exibida e lida a mensagem de agradecimento habilitando o botão para geração do relatório logo em seguida. A figura a seguir mostra a referida situação.

Figura 4.12 – Tela de diálogo exibindo mensagem de conclusão do questionário

Ao ser acionado, o botão “Gerar Relatório” permite a exibição da tela de resultados construída pela classe *ResultView*, responsável também pela criação do mesmo conteúdo em arquivo no formato JSON. A tela exibe todas as perguntas do questionário com as respectivas respostas do usuário com a opção de encerramento da aplicação através do botão “Sair”. Um exemplo desta tela pode ser visto na figura a seguir.

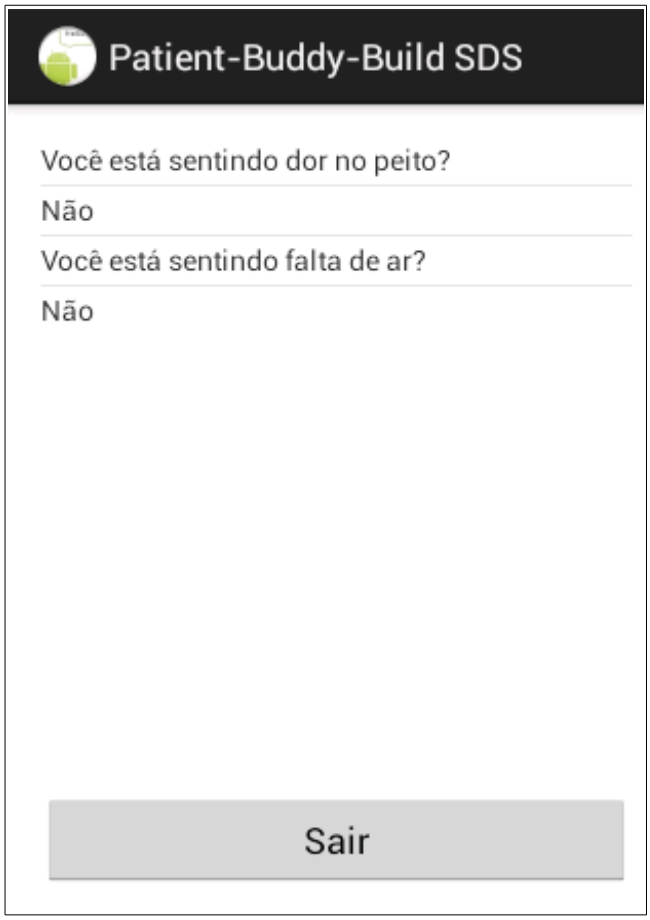

Figura 4.13 – Tela de resultados

Neste capítulo foram abordados detalhes da aplicação Patient-Buddy-Build e da interação humano-computador baseada em sistemas de diálogo como alternativa para os questionários customizados gerados por ela.

5 CONCLUSÕES

A motivação para a realização desta aplicação surgiu a partir da necessidade de pacientes serem avaliados sem obrigatoriamente estarem em um centro de tratamento diante de profissionais de saúde. Para tal, foi realizado um estudo de viabilidade da utilização da fala como forma de interação com o intuito de contornar as limitações que alguns pacientes possuem, especialmente aqueles integrantes de comunidades carentes, ao lidar com as tradicionais interfaces textuais. Além de apresentar uma alternativa mais comum aos usuários, a aplicação contribui para a redução do constante número de deslocamentos para a realização de consultas junto a estes profissionais que, por sua vez, podem realizar diagnósticos à distância com base nas respostas de seus pacientes. Considerando que alguns destes deslocamentos constituem grandes empecilhos para muitos pacientes, a aplicação se apresenta como uma boa alternativa no acompanhamento realizado junto aos profissionais de saúde.

Os objetivos propostos para este trabalho foram atingidos durante um árduo processo, visto que são muitos os desafios envolvidos na implantação de interações humano-computador baseadas em sistemas de diálogo falado. A atual e considerável escassez de ferramentas de reconhecimento e reprodução de áudio com suporte à língua portuguesa, aliada à complexidade de se processar algo tão peculiar como a voz humana e os meios onde ela se propaga, constantemente afetados por ruídos indesejados que dificultam ainda mais o trabalho, são algumas das barreiras a serem transpostas.

As metas estabelecidas para trabalhos futuros envolvem o processamento da linguagem natural como forma de coesão entre as mais variadas maneiras de se referir a um mesmo termo referente a uma resposta fazendo uso da sinonímia, por exemplo, e a incrementação do *feedback* entre a aplicação e o usuário visando maior entendimento do fluxo do diálogo, além de testes a serem realizados em situações reais envolvendo pacientes em tratamento e respectivos profissionais de saúde responsáveis como forma de avaliação de funcionalidades e revelação de possíveis alterações em sua estrutura.

REFERÊNCIAS

[1] THE NIELSEN COMPANY. *Consumidores online ao redor do mundo e meios de comunicação multi-telas: hoje e amanhã*. São Paulo: Nielsen, 2012. Disponível em: <http://www.br.nielsen.com/reports/documents/Consumidor_Online_Maio2012.pdf/> [Acesso em: 05 de Agosto de 2013]

[2] G1. *Número de smartphones triplicará no mundo até 2018, diz pesquisa*. Disponível em: <http://g1.globo.com/tecnologia/noticia/2013/03/numero-de-smartphones-triplicara-no-mundo-ate-2018-diz-pesquisa.html/> [Acesso em: 05 de Agosto de 2013]

[3] INTERNATIONAL DATA CORPORATION. *IDC apresenta as previsões para o mercado de Tecnologia da Informação e Comunicação em 2012*. Disponível em: < http://www.idclatin.com/news.asp?ctr=bra&id_release=2212/> [Acesso em: 05 de Agosto de 2013]

[4] UOL. *Em 2014, banda larga móvel deve atingir 124 milhões de acesso*. Disponível em: <http://economia.uol.com.br/ultimas-noticias/infomoney/2012/03/20/em-2014-banda-larga-movel-deve-atingir-124-milhoes-de-acessos.jhtm/> [Acesso em: 05 de Agosto de 2013]

[5] Kaplan, A. M; Haenlein, M. (2010). *Users of the world, unite! The challenges and opportunities of social media*. Business Horizons, 53(1): 59 – 68.

[6] Xin Lin; Qin Kede. *Embeddedness, Social Network Theory and Social Capital Theory: Antecedents and Consequence,* Management and Service Science (MASS), 2011 International Conference on, pp.1-5, 12-14 Aug. 2011.

[7] Yu, W.D; Siddiqui, A. *Towards a Wireless Mobile Social Network System Design in Healthcare,* Multimedia and Ubiquitous Engineering, 2009. MUE ’09. Third International Conference on, pp.429-436, 4-6 June 2009.

[8] Juwel Rana; Johan Kristiansson; Josef Hallberg; Kare Synnes. *An Architecture for Social Mobile Networking Applications*, 2009.

[9] R H Istepanian; S Laxminarayan, C. S. P. (2006). *MHealth: Emerging Mobile Health Systems*. Springer, New York, NY, USA.

[10] Demiris G. *The diffusion of virtual communities in health care: concepts and challenges*. Patient Education and Counseling—Elsevier. 2006; 178-188.

[11] Jain, R; Sonnen, D. *Social Life Networks, IT Professional*, vol.13, no.5, pp.8-11, Sept.-Oct. 2011.

[12] Wagner, E. H; Austin, B. T; Davis, C; Hindmarsh, M; Schaefer, J; Bonomi, A. *Improving chronic illness care: translating evidence into action*. Health Affairs, 20(6): 64–78. 2001.

[13] M. F. McTear; T. V. Ramam. *Spoken Dialogue Technology*: *Toward the Conversational User Interface*. First Edition. Springer, 2004.

[14] TELES, A.S; GONÇALVES, J.F; SILVA, F; PINHEIRO, V.; ENDLER, M. *Infraestrutura e Aplicações de Redes Sociais Moveis para Colaboração em Saúde*. Trabalho apresentado ao XIII Congresso Brasileiro de Informática em Saúde, ISSN 2178-2857, Curitiba, Novembro, 2012.

[15] BATISTA, R. (2012). *Infraestrutura de comunicação para o middleware MobileHealthNet*. Master's thesis, Universidade Federal do Maranhão.

[16] TELES, A. S.; PINHEIRO, D. N.; GONÇALVES, J. F.; BATISTA, R.; DA SILVA E SILVA, F. J.; PINHEIRO, V.; ENDLER, M. (2013). *Redes sociais móveis: Conceitos, aplicações e aspectos de segurança e privacidade*. In 31o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos.

[17] Da Saúde, M. (2006). *Política Nacional de Atenção Básica*. Editora do Ministério da Saúde, 4a edição.

[18] Leimeister JM, Daum M, Krcmar H. Mobile virtual healthcare communities: An approach to community engineering for cancer patients. In: European Conference on Information Systems (ECIS). 2002; 1626-1637.

[19] Object Management Group. Data Distribution Service for Real-Time Systems Specification. 2001.

[20] Gao, H.; Hu, J.; Huang, T.; Wang, J.; Chen, Y. (2011). Security issues in online social networks. Internet Computing, IEEE, 15(4):56 –63.

[21] GOOGLE. *Google Android*. Disponível em:<http://developer.android.com/guide/> [Acesso em: 05 de Outubro de 2013]

[22] NUANCE. *Dragon Mobile SDK Reference*. Disponível em: < http://dragonmobile.nuancemobiledeveloper.com//public/Help/DragonMobileSDKReference_Android/Introduction.html/> [Acesso em: 15 de Agosto de 2013]

[23] PINHEIRO, V; ENDLER, M; HAEUSLER, E. *Patient-Buddy-Build: Acompanhamento remoto móvel customizável de pacientes com doenças crônicas*. Trabalho apresentado ao XIII Congresso Brasileiro de Informática em Saúde, ISSN 2178-2857, Curitiba, Novembro, 2012.

[24] Asmi A; Ragavan L, C. J. Pervasive asthma monitoring system. pams. a health systems approach to remote monitoring of asthma. 2007.

[25] Gruber, Thomas. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal Human-Computer Studies Vol. 43, Issues 5-6, Novemer 1995, p.907-928.

[26] ALMEIDA, V. P.; ENDLER, M.; Haeusler, E. H. (2013). Patient- buddy-build: Customized mobile monitoring for patients with chronicle diseases. In Proceedings of the 5th International Conference on eHealth, Telemedicine, and Social Medicine, eTELEMED ’13, Nice, France.

[27] JSON. *Introduction JSON*. Disponível em: <http://www.json.org/> [Acesso em: 09 de Outubro de 2013]